1 A primer on kernel methods

Jean-Philippe Vert
Koji Tsuda
Bernhard Schölkopf

Kernel methods in general, and support vector machines (SVMs) in particular, are increasingly used to solve various problems in computational biology. They offer versatile tools to process, analyze, and compare many types of data, and offer state-of-the-art performance in many cases. This self-contained introduction to positive definite kernels and kernel methods aims at providing the very basic knowledge and intuition that the reader might find useful in order to fully grasp the technical content of this book.

1.1 Introduction

Kernel methods in general and SVMs in particular have been successfully applied to a number of real-world problems and are now considered state-of-the-art in various domains, although it was only fairly recently that they became part of the mainstream in machine learning and empirical inference. The history of methods employing positive definite kernels, however, can be traced back at least a few decades. Aronszajn (1950) and Parzen (1962) were some of the first to employ these methods in statistics. Subsequently, Aizerman et al. (1964) used positive definite kernels in a way which was already closer to what people now call the *kernel trick*. They employed radial basis function kernels to reduce a convergence proof for the *potential function classifier* to the linear perceptron case. To do this, they had to argue that a positive definite kernel is identical to a dot product in another space (sometimes called the *feature space*), in which their algorithm reduced to the perceptron algorithm. They did not, however, use the feature space view to design new algorithms.

Kernel methods

The latter was done some thirty years later by Boser et al. (1992), to construct the SVMs, a generalization of the so-called *optimal hyperplane* algorithm. Initially, it

was thought that the main strength of SVMs compared to the optimal hyperplane algorithm was that they allowed the use of a larger class of similarity measures. Just as optimal hyperplanes, however, they were only used on vectorial data. But soon it was noted (Schölkopf, 1997) that kernels not only increase the flexibility by increasing the class of allowed similarity measures but also make it possible to work with nonvectorial data. This is due to the fact that kernels automatically provide a vectorial representation of the data in the feature space. The first examples of nontrivial kernels defined on nonvectorial data were those of Haussler (1999) and Watkins (2000) (see also Cristianini and Shawe-Taylor, 2000). Moreover, it was pointed out (Schölkopf et al., 1998) that kernels can be used to construct generalizations of any algorithm that can be carried out in terms of dot products, and the last 5 years have seen a large number of "kernelizations" of various algorithms (Graepel and Obermayer, 1998; Weston et al., 1999; Tsuda, 1999; Ruján and Marchand, 2000; Herbrich et al., 2000; Fyfe and Lai, 2000; Rosipal and Trejo, 2001; Akaho, 2001; Harmeling et al., 2001; Girolami, 2002; Suykens et al., 2002; Weston et al., 2003; Vert and Kanehisa, 2003; Kuss and Graepel, 2003).

Further threads of kernel work can be identified in approximation theory and statistics (Berg et al., 1984; Micchelli, 1986; Wahba, 2002; Poggio and Girosi, 1990), as well as in the area of Gaussian process prediction and related fields such as kriging, where kernels play the role of *covariance functions* (see, e.g., Weinert, 1982; Williams, 1998; MacKay, 1998).

This chapter is structured as follows. Section 1.2 is devoted to the presentation of kernels and some of their basic properties. Kernel methods are then introduced in section 1.3, SVMs being treated in more detail in section 1.4. We then discuss more advanced kernel topics relevant to this book, including the presentation of several families of kernels in section 1.6, and an introduction to the emergent field of kernel design in section 1.7.

1.2 Kernels

Kernels are the basic ingredient shared by all kernel methods. They provide a general framework to represent data, and must satisfy some mathematical conditions. These conditions give them a number of properties useful to bear in mind when it comes to understanding the intuition behind kernel methods and kernel design.

1.2.1 The Issue of Data Representation

Let us denote by $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ a set of n objects to be analyzed. We suppose that each object $\mathbf{x}_i$ is an element of a set $\mathcal{X}$, which may, for example, be the set of all possible images if one wants to analyze a set of images, or the set of all possible molecules in a biological context. In order to design data analysis methods, the first question to be addressed is how to represent the data set $\mathcal{S}$ for further processing.

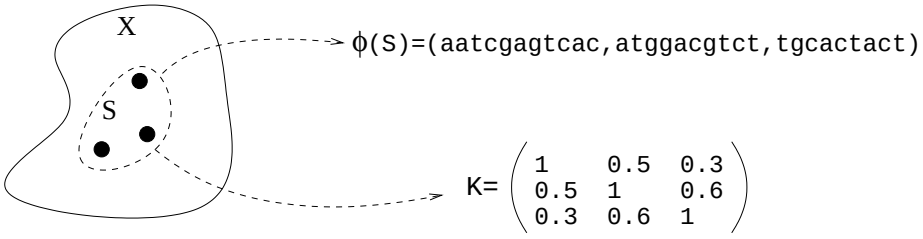

Figure 1.1 Two different representations of the same dataset. $\mathcal{X}$ is supposed to be the set of all oligonucleotides, and $\mathcal{S}$ is a data set of three particular oligonucleotides. The classic way to represent $\mathcal{S}$ is first to define a representation $\phi(\mathbf{x})$ for each element of $\mathbf{x} \in \mathcal{X}$, for example, as a sequence of letters to represent the succession of nucleotides, and then to represent $\mathcal{S}$ as the set $\phi(\mathcal{S})$ of representations of its elements (*upper part*). Kernel methods are based on a different representation of $\mathcal{S}$, as a matrix of pairwise similarity between its elements (*lower part*).

The vast majority of data analysis methods, outside kernel methods, have a natural answer to this question: first define a representation for each object, and then represent the set of objects by the set of their representations. Formally, this means that a representation $\phi(\mathbf{x}) \in \mathcal{F}$ is defined for each possible object $\mathbf{x} \in \mathcal{X}$, where the representation can, for example, be a real-valued vector ($\mathcal{F} = \mathbb{R}^p$), a finite-length string ($\mathcal{F}$ is then the set of all finite-length strings), or a more complex representation that can be processed by an algorithm. The data set $\mathcal{S}$ is then represented as the set of individual object representations, $\phi(\mathcal{S}) = (\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_n))$, and the algorithm is designed to process such data. As an example, if a protein is represented by a sequence of letters that corresponds to its primary structure, then a set of proteins can be represented by a set of sequences.

Kernel representation

Kernel methods are based on a radically different answer to the question of data representation. Data are not represented individually anymore, but only through a set of pairwise comparisons. In other words, instead of using a mapping $\phi : \mathcal{X} \to \mathcal{F}$ to represent each object $\mathbf{x} \in \mathcal{X}$ by $\phi(\mathbf{x}) \in \mathcal{F}$, a real-valued "comparison function" $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is used, and the data set $\mathcal{S}$ is represented by the $n \times n$ matrix of pairwise comparisons $k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. All kernel methods are designed to process such square matrices. The difference between both approaches is represented in figure 1.1.

Several comments can already be made at this point. First, the representation as a square matrix does not depend on the nature of the objects to be analyzed. They can be images, molecules, or sequences, and the representation of a data set is always a real-valued square matrix. This suggests that an algorithm developed to process such a matrix can analyze images as well as molecules or sequences, as long as valid functions k can be defined. This also suggests that a complete modularity exists between the design of a function k to represent data on the one hand, and the design of an algorithm to process the data representations on the other hand. These properties turn out to be of utmost importance in fields like computational

biology, where data of different nature need to be integrated and analyzed in a unified framework.

Second, the size of the matrix used to represent a dataset of n objects is always $n \times n$, whatever the nature or the complexity of the objects. For example, a set of ten tissues, each characterized by thousands of gene expression levels, is represented by a 10×10 matrix, whatever the number of genes. Computationally, this is very attractive in the case when a small number of complex objects are to be processed.

Third, there are many cases where comparing objects is an easier task than finding an explicit representation for each object that a given algorithm can process. As an example, many data analysis algorithms, such as least squares regression or neural networks, require an explicit representation of each object $\mathbf{x}$ as a vector $\phi(\mathbf{x}) \in \mathbb{R}^p$. There is no obvious way to represent protein sequences as vectors in a biologically relevant way, however, while meaningful pairwise sequence comparison methods exist.

1.2.2 General Definition

As the reader might guess, the comparison function k is a critical component of any kernel method, because it defines how the algorithm "sees" the data. Most kernel methods described below can only process square matrices, which are symmetric *positive definite*. This means that if k is an $n \times n$ matrix of pairwise comparisons, it should satisfy $k_{i,j} = k_{j,i}$ for any $1 \leq i, j \leq n$, and $\boldsymbol{c}^\top k \boldsymbol{c} \geq 0$ for any $\boldsymbol{c} \in \mathbb{R}^n$.[1] This motivates the following definition:

Positive definite kernel

Definition 1.1 *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* positive definite kernel *iff it is symmetric, that is, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ for any two objects $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, and positive definite, that is,*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for any $n > 0$, any choice of n objects $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathcal{X}$, and any choice of real numbers $c_1, \cdots, c_n \in \mathbb{R}$.

From now on, we only focus on positive definite kernels, and simply call them *kernels*. Definition 1.1 ensures that if k is a kernel, then any pairwise similarity matrix built from k is symmetric positive definite.

Imposing the condition that a comparison function be a kernel clearly restricts the class of functions one can use. For example, the local alignment scores widely used in computational biology to assess the similarity between sequences are not in general kernels (see chapter **??**). However, this restriction is often worth the cost

1. In mathematics, such a matrix is usually called positive semidefinite, because $\mathbf{c}^\top k \mathbf{c}$ can be zero and not strictly positive. However, in this chapter, we follow the notation in approximation theory (Wahba, 2002), which omits "semi" for simplicity.

because it opens the door to the use of kernel methods (see section 1.3). Moreover, an increasing number of "tricks" are being developed to derive kernels from virtually any comparison function (see section 1.7).

Before we describe kernel methods in more detail, however, let us try to get a better intuition about kernels themselves. Kernels have several properties which one should bear in mind in order to fully understand kernel methods, and to understand the motivations behind the kernels developed in this book.

1.2.3 Kernels as Inner Product

Let us start with a simple example that leads to a fundamental property of kernels. Suppose the data to be analyzed are real vectors, that is, $\mathcal{X} = \mathbb{R}^p$ and any object is written as $\mathbf{x} = (x_1, \cdots, x_p)^\top$. One is tempted to compare such vectors using their inner product: for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^p$,

Linear kernel

$$k_L(\mathbf{x}, \mathbf{x}') := \mathbf{x}^\top \mathbf{x}' = \sum_{i=1}^{p} x_i x_i'. \tag{1.1}$$

This function is a kernel. Indeed, it is symmetric ($\mathbf{x}^\top \mathbf{x}' = \mathbf{x}'^\top \mathbf{x}$), and the positive definiteness results from the following simple calculation, valid for any $n > 0$, $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^p$, and $c_1, \cdots, c_n \in \mathbb{R}$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_L(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \mathbf{x}_i^\top \mathbf{x}_j = ||\sum_{i=1}^{n} c_i \mathbf{x}_i||^2 \geq 0. \tag{1.2}$$

The inner product between vectors is the first kernel we encounter. It is usually called the *linear kernel*. An obvious limitation of this kernel is that it is only defined when the data to be analyzed are vectors. For more general objects $\mathbf{x} \in \mathcal{X}$, however, this suggests a way to define kernels in a very systematic manner, by first representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathbb{R}^p$, and then defining a kernel for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ by

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}'). \tag{1.3}$$

Following the same line of computation as in (1.2), the reader can easily check that the function k defined in (1.3) is a valid kernel on the space $\mathcal{X}$, which does not need to be a vector space.

Any mapping $\phi : \mathcal{X} \to \mathbb{R}^p$ for some $p \geq 0$ results in a valid kernel through (1.3). Conversely, one might wonder whether there exist more general kernels than these. As the following classic result of Aronszajn (1950) shows, the answer is negative, at

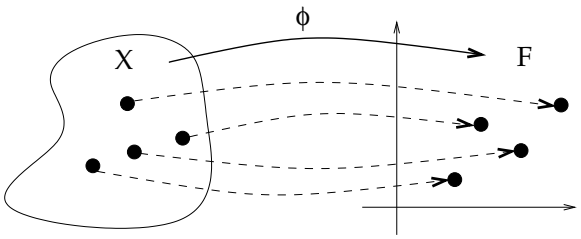

Figure 1.2 Any kernel on a space $\mathcal{X}$ can be represented as an inner product after the space $\mathcal{X}$ is mapped to a Hilbert space $\mathcal{F}$, called the feature space.

least if one allows $\mathbb{R}^p$ to be replaced by an eventually infinite-dimensional Hilbert space[2]:

Theorem 1.2 *For any kernel k on a space $\mathcal{X}$, there exists a Hilbert space $\mathcal{F}$ and a mapping $\phi : \mathcal{X} \to \mathcal{F}$ such that*

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \quad \textit{for any } \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \tag{1.4}$$

where $\langle u, v \rangle$ represents the dot product in the Hilbert space between any two points $u, v \in \mathcal{F}$.

Feature space

This result, illustrated in figure 1.2, provides a first useful intuition about kernels: they can all be thought of as dot products in some space $\mathcal{F}$, usually called the *feature space*. Hence, using a kernel boils down to representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathcal{F}$, and computing dot products. There is, however, an important difference with respect to the explicit representation of objects as vectors, discussed in subsection 1.2.1: here the representation $\phi(\mathbf{x})$ does not need to be computed explicitly for each point in the data set $\mathcal{S}$, since only the pairwise dot products are necessary. In fact, there are many cases where the feature space associated with a simple kernel is infinite-dimensional, and the image $\phi(\mathbf{x})$ of a point $\mathbf{x}$ is tricky to represent even though the kernel is simple to compute.

This intuition of kernels as dot products is useful in order to provide a geometric interpretation of kernel methods. Indeed, most kernel methods possess such an interpretation when the points $\mathbf{x} \in \mathcal{X}$ are viewed as points $\phi(\mathbf{x})$ in the feature space.

1.2.4 Kernels as Measures of Similarity

In this book, as well as in the kernel methods community, kernels are often presented as measures of similarity, in the sense that $k(\mathbf{x}, \mathbf{x}')$ is "large" when $\mathbf{x}$ and $\mathbf{x}'$

2. A Hilbert space is a vector space endowed with a dot product (a strictly positive and symmetric bilinear form), that is complete for the norm induced. $\mathbb{R}^p$ with the classic inner product is an example of a finite-dimensional Hilbert space.

are "similar." This motivates the design of kernels for particular types of data or applications, because particular prior knowledge might suggest a relevant measure of similarity in a given context. As an example, the string and graph kernels presented in chapters **??** and **??** are motivated by a prior intuition of relevant notions of similarity: the fact that two biological sequences are similar when there exist good alignments between them, on the one hand, and the fact that two graphs are similar when they share many common paths, on the other.

The justification for this intuition of kernels as measures of similarity is not always obvious, however. From subsection 1.2.3 we know that kernels are dot products in a feature space. Yet the notion of dot product does not always fit one's intuition of similarity, which is more related to a notion of distance. There are cases where these notions coincide. Consider, for example, the following kernel on $\mathcal{X} = \mathbb{R}^p$, called the

RBF kernel

Gaussian radial basis function (RBF) kernel:

$$k_G(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{d\left(\mathbf{x}, \mathbf{x}'\right)^2}{2\sigma^2}\right), \tag{1.5}$$

where σ is a parameter and d is the Euclidean distance. This is a valid kernel (see subsection 1.7.2), which can be written as a dot product $k_G\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \phi\left(\mathbf{x}\right), \phi\left(\mathbf{x}'\right)\right\rangle$ by theorem 1.2. The feature space is a functional space, and an explicit form of the map ϕ is not obvious. By (1.5), we see that this kernel is a decreasing function of the Euclidean distance between points, and therefore has a relevant interpretation as a measure of similarity: the larger the kernel $k_G(\mathbf{x}, \mathbf{x}')$, the closer the points $\mathbf{x}$ and $\mathbf{x}'$ in $\mathcal{X}$.

For more general kernels k on a space $\mathcal{X}$, basic linear algebra in the feature space associated with k by theorem 1.2 shows that the following holds for any two objects $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$k(\mathbf{x}, \mathbf{x}') = \frac{\left|\left|\phi\left(\mathbf{x}\right)\right|\right|^2 + \left|\left|\phi\left(\mathbf{x}'\right)\right|\right|^2 - d\left(\phi\left(\mathbf{x}'\right), \phi\left(\mathbf{x}'\right)\right)^2}{2}, \tag{1.6}$$

where d is the Hilbert distance defined by $d(u, v)^2 = \left\langle (u - v), (u - v)\right\rangle$ and $||.||$ is the Hilbert norm ($||u||^2 = \langle u, u\rangle$). Equation (1.6) shows that the kernel $k\left(\mathbf{x}, \mathbf{x}'\right)$ measures the similarity between $\mathbf{x}$ and $\mathbf{x}'$ as the opposite of the square distance $d\left(\phi\left(\mathbf{x}\right), \phi\left(\mathbf{x}'\right)\right)^2$ between their images in the feature space, up to the terms $\left|\left|\phi\left(\mathbf{x}\right)\right|\right|^2$ and $\left|\left|\phi\left(\mathbf{x}'\right)\right|\right|^2$. If all points have the same length in the feature space, meaning $\left|\left|\phi\left(\mathbf{x}\right)\right|\right|^2 = k\left(\mathbf{x}, \mathbf{x}\right) = \textit{constant}$ for all $\mathbf{x} \in \mathcal{X}$, then the kernel is simply a decreasing measure of the distance in the feature space. This is, for example, the case for all translation-invariant kernels of the form $k\left(\mathbf{x}, \mathbf{x}'\right) = \psi\left(\phi\left(\mathbf{x}\right) - \phi\left(\mathbf{x}'\right)\right)$ such as the Gaussian RBF kernel (1.5), because in this case $k\left(\mathbf{x}, \mathbf{x}\right) = \psi(0)$ for any $\mathbf{x} \in \mathcal{X}$. For more general kernels, one should keep in mind the slight gap between the notion of dot product and similarity.

The conclusion of this section is that it is generally relevant to think of a kernel as a measure of similarity, in particular when it is constant on the diagonal. This intuition is useful in designing kernels and in understanding kernel methods.

For example, methods like SVMs (see section 1.4), which predict the values of a function at a given point from the observation of its values at different points, base their prediction on the hypothesis that "similar" points are likely to have similar values. The "similarity" between points mentioned here is precisely the similarity determined by the kernel.

1.2.5 Kernels as Measures of Function Regularity

Let k be a kernel on a space $\mathcal{X}$. In this section we show that k is associated with a set of real-valued functions on $\mathcal{X}$, $\mathcal{H}_k \subset \{f : \mathcal{X} \to \mathbb{R}\}$, endowed with a structure of Hilbert space (in particular, with a dot product and a norm). Understanding the functional space $\mathcal{H}_k$ and the norm associated with a kernel often helps in understanding kernel methods and in designing new kernels, as we illustrate in section 1.3.

Let us start with two examples of kernels and their associated functional spaces. Consider first the linear kernel (1.1) on a vector space $\mathcal{X} = \mathbb{R}^p$. The corresponding functional space is the space of linear function $f : \mathbb{R}^d \to \mathbb{R}$:

$$\mathcal{H}_k = \left\{ f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \quad : \mathbf{w} \in \mathbb{R}^p \right\}, \tag{1.7}$$

and the associated norm is just the slope of the linear function,

$$||f||_{\mathcal{H}_k} = ||\mathbf{w}|| \quad \text{for } f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}. \tag{1.8}$$

As a second example, consider the Gaussian RBF kernel (1.5) on the same vector space $\mathcal{X} = \mathbb{R}^p$. The associated functional space is the set of functions $f : \mathbb{R}^d \to \mathbb{R}$ with Fourier transform $\hat{f}$ that satisfies

$$N(f) = \frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \int_{\mathbb{R}^p} \left| \hat{f}(\omega) \right|^2 e^{\frac{\sigma^2}{2}||\omega||^2} d\omega < +\infty,$$

and the norm in $\mathcal{H}_k$ is precisely this functional: $||f||_{\mathcal{H}_k} = N(f)$. Hence $\mathcal{H}_k$ is a set of functions with Fourier transforms that decay rapidly, and the norm $||.||_{\mathcal{H}_k}$ quantifies how fast this decay is.

In both examples, the norm $||f||_{\mathcal{H}_k}$ decreases if the "smoothness" of f increases, where the definition of smoothness depends on the kernel. For the linear kernel, the smoothness is related to the slope of the function: a smooth function is a flat function. For the Gaussian RBF kernel, the smoothness of a function is measured by its Fourier spectrum: a smooth function has little energy at high frequencies. These examples of smoothness turn out to be very general, the precise definition of smoothness depending on the kernel considered. In fact, the notion of smoothness is dual to the notion of similarity discussed in subsection 1.2.4: a function is "smooth" when it varies slowly between "similar" points.

Let us now sketch the systematic construction of the functional space $\mathcal{H}_k$ from the kernel k. The set $\mathcal{H}_k$ is defined as the set of function $f : \mathcal{X} \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}), \tag{1.9}$$

for $n > 0$, a finite number of points $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathcal{X}$, and a finite number of weights $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, together with their limits under the norm:

$$||f||^2_{\mathcal{H}_k} := \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right). \tag{1.10}$$

It can be checked that this norm is independent of the representation of f in (1.9). $\mathcal{H}_k$ is in fact a Hilbert space, with a dot product defined for two elements $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$ and $g(\mathbf{x}) = \sum_{j=1}^{m} \alpha'_i k(\mathbf{x}'_i, \mathbf{x})$ by

Reproducing kernel Hilbert space

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \alpha'_j k\left(\mathbf{x}_i, \mathbf{x}'_j\right).$$

An interesting property of this construction is that the value $f\left(\mathbf{x}\right)$ of a function $f \in \mathcal{H}_k$ at a point $\mathbf{x} \in \mathcal{X}$ can be expressed as a dot product in $\mathcal{H}_k$,

$$f\left(\mathbf{x}\right) = \langle f, k\left(\mathbf{x}, .\right) \rangle. \tag{1.11}$$

In particular, taking $f\left(.\right) = k\left(\mathbf{x}', .\right)$, we derive the following reproducing property valid for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \langle k\left(\mathbf{x}, .\right), k\left(\mathbf{x}', .\right) \rangle. \tag{1.12}$$

For this reason, the functional space $\mathcal{H}_k$ is usually called the reproducing kernel Hilbert space (RKHS) associated with k. The equality in (1.12) also shows that the Hilbert space $\mathcal{H}_k$ is one possible feature space associated with the kernel k, when we consider the mapping $\phi : \mathcal{X} \to \mathcal{H}_k$ defined by $\phi\left(\mathbf{x}\right) := k\left(\mathbf{x}, .\right)$. Indeed, (1.4) is exactly equivalent to (1.12) in this case. The construction of $\mathcal{H}_k$ therefore provides a proof of theorem 1.2.

Regularization

Aside from the technicalities of this section, the reader should keep in mind the connection between kernels and norms on functional spaces. Most kernel methods have an interpretation in terms of functional analysis. As an example, we show in the next sections that many kernel methods, including SVMs, can be defined as algorithms that, given a set of objects $\mathcal{S}$, return a function that solves the equation

$$\min_{f \in \mathcal{H}_k} R(f, \mathcal{S}) + c||f||_{\mathcal{H}_k}, \tag{1.13}$$

where $R(f, \mathcal{S})$ is small when f "fits" the data well, and the term $||f||_{\mathcal{H}_k}$ ensures that the solution of (1.13) is "smooth." In fact, besides fitting the data well and being smooth, the solution to (1.13) turns out to have special properties that are useful for computational reasons, which are discussed in theorem 1.3.3 below.

1.3 Some Kernel Methods

Having discussed the notions of data representation and kernels in section 1.2, let us now turn our attention to the algorithms that process the data to perform some particular tasks, such as clustering, computing various properties, inferring a regression or classification function from its observation on a finite set of points, and so on. We focus here on a class of algorithms called kernel methods, which can roughly be defined as those for which the data to be analyzed only enter the algorithm through the kernel function; in other words, algorithms that take as input the similarity matrix defined by a kernel.

Recent years have witnessed the development of a number of kernel methods, which we do not have the ambition to survey in full generality in this short introduction. Historically, the first kernel method recognized as such is the SVM (Boser et al., 1992), which has found many applications in computational biology (see survey in chapter **??**), and which we describe in detail in section 1.4. Before this, let us try briefly to give a flavor of the two concepts that underlie most kernel methods: the *kernel trick* and the *representer theorem*.

1.3.1 The Kernel Trick

The kernel trick is a simple and general principle based on the property of kernels discussed in subsection 1.2.3, namely that they can be thought of as inner product. It can be stated as follows.

Kernel trick

Proposition 1.3 *Any algorithm for vectorial data that can be expressed only in terms of dot products between vectors can be performed implicitly in the feature space associated with any kernel, by replacing each dot product by a kernel evaluation.*

The kernel trick is obvious but has huge practical consequences that were only recently exploited. It is first a very convenient trick to transform linear methods, such as linear discriminant analysis (Hastie et al., 2001) or principal component analysis (PCA; Jolliffe, 1986), into nonlinear methods, by simply replacing the classic dot product by a more general kernel, such as the Gaussian RBF kernel (1.5). Nonlinearity is then obtained at no computational cost, as the algorithm remains exactly the same. The operation that transforms a linear algorithm into a more general kernel method is often called *kernelization*.

Second, the combination of the kernel trick with kernels defined on nonvectorial data permits the application of many classic algorithms on vectors to virtually any type of data, as long as a kernel can be defined. As an example, it becomes natural to perform PCA on a set of sequences, thanks to the availability of kernels for sequences such as those discussed in chapter **??** or chapter **??**, or to search for structure-activity relationships on chemical compounds using least squares regression without computing an explicit representation of molecules as vectors, thanks to the graph kernel presented in chapter **??**.

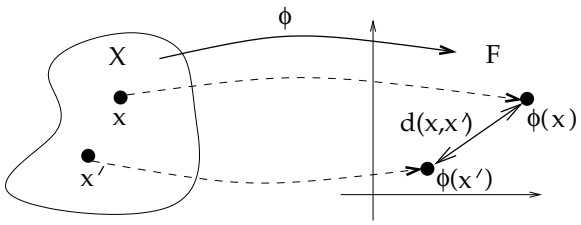

Figure 1.3 Given a space $\mathcal{X}$ endowed with a kernel, a distance can be defined between points of $\mathcal{X}$ mapped to the feature space $\mathcal{F}$ associated with the kernel. This distance can be computed without explicitly knowing the mapping ϕ thanks to the kernel trick.

1.3.2 Example: Computing Distances between Objects

Let us illustrate the kernel trick on the very simple problems of computing distances between points and clouds of points, which we attempt in general sets $\mathcal{X}$ endowed with a kernel k. Recall from theorem 1.2 that the kernel can be expressed as a dot product $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ in a dot product space $\mathcal{F}$ for some mapping $\phi : \mathcal{X} \to \mathcal{F}$.

As a starter consider two objects $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, such as two sequences or two molecules. These points are mapped to two vectors $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$ in $\mathcal{F}$, so it is natural to define a distance $d(\mathbf{x}_1, \mathbf{x}_2)$ between the objects as the Hilbert distance between their images,

$$d(\mathbf{x}_1, \mathbf{x}_2) := ||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)||. \tag{1.14}$$

This definition is illustrated in figure 1.3. At first sight, it seems necessary to explicitly compute the images $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$ before computing this distance. However, the following simple equality shows that the distance (1.14) can be expressed in terms of dot products in $\mathcal{F}$:

$$||\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)||^2 = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_1) \rangle + \langle \phi(\mathbf{x}_2), \phi(\mathbf{x}_2) \rangle - 2\langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle. \tag{1.15}$$

Applying the kernel trick in (1.15) and plugging the result into (1.14) shows that the distance can be computed only in terms of the kernel,

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{k(\mathbf{x}_1, \mathbf{x}_1) + k(\mathbf{x}_2, \mathbf{x}_2) - 2k(\mathbf{x}_1, \mathbf{x}_2)}. \tag{1.16}$$

The effect of the kernel trick is easily understood in this example: it is possible to perform operations implicitly in the feature space. This is of utmost importance for kernels that are easy to calculate directly, but correspond to complex feature spaces, such as the Gaussian RBF kernel (1.5).

Let us now consider the following slightly more general problem. Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a fixed finite set of objects, and $\mathbf{x} \in \mathcal{X}$ a generic object. Is it possible to assess how "close" the object $\mathbf{x}$ is to the set of objects $\mathcal{S}$?

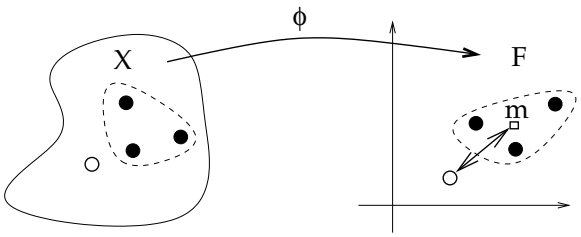

Figure 1.4 The distance between the white circle and the set of three black circles in the space $\mathcal{X}$ endowed with a kernel (*on the left*) is defined as the distance in the feature space between the image of the white circle and the centroid m of the images of the black circles. The centroid m might have no preimage in $\mathcal{X}$. This distance can nevertheless be computed implicitly with the kernel trick.

This question might be of interest in different contexts. For example, in binary classification, one observes two sets of objects $\mathcal{S}_1$ and $\mathcal{S}_2$ having two different properties, and one is asked to predict the property of a new object $\mathbf{x}$. A natural way to achieve this is to predict that $\mathbf{x}$ has the property of the objects in $\mathcal{S}_1$ if it is closer to $\mathcal{S}_1$ than $\mathcal{S}_2$, and the other property otherwise. A second example was recently proposed by Gorodkin et al. (2001) in the context of multiple sequence alignment. Given a set of biological sequences to align jointly, the authors proposed a pairwise alignment score as a kernel from which they derived a ranking of the sequences, from the most central to the most peripheral with respect to the whole set. This ranking can then be used to improve a greedy multiple alignment.

Having mapped the data set $\mathcal{S}$ and the object $\mathbf{x}$ in the feature space with the function ϕ, a natural way to measure the distance from $\mathbf{x}$ to $\mathcal{S}$ is to define it as the Euclidean distance between $\phi(\mathbf{x})$ and the centroid of $\mathcal{S}$ in the feature space, where the centroid is defined as

$$m = \frac{1}{n}\sum_{i=1}^{n} \phi(\mathbf{x}_i).$$

In general, there is no reason why m should be the image of an object $\mathbf{x} \in \mathcal{X}$ by ϕ, but still it is well defined as an element of $\mathcal{F}$. As illustrated in figure 1.4, we can now define the distance from $\mathbf{x}$ to $\mathcal{S}$ as follows:

$$\mathrm{dist}(\mathbf{x}, \mathcal{S}) = ||\phi(\mathbf{x}) - m|| = ||\phi(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n} \phi(\mathbf{x}_i)||. \tag{1.17}$$

Expanding the square distance (1.17) in terms of dot products in the feature space as we did in (1.15), and using the kernel trick, we obtain

$$\mathrm{dist}(\mathbf{x}, \mathcal{S}) = \sqrt{k(\mathbf{x},\mathbf{x}) - \frac{2}{n}\sum_{i=1}^{n} k(\mathbf{x},\mathbf{x}_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(\mathbf{x}_i,\mathbf{x}_j)}. \tag{1.18}$$

This shows that the distance from $\mathbf{x}$ to $\mathcal{S}$ can be computed entirely from the values of the kernels between pairs of points in $\{\mathbf{x}\} \cup \mathcal{S}$, even though it is defined as a distance in the feature space between $\phi(\mathbf{x})$ and a point m that does not necessarily even have a preimage $\phi^{-1}(m)$ in $\mathcal{X}$.

As a slight generalization to (1.18), interested readers can now easily verify that the kernel trick allows them to define the following functional as a distance between two sets of points $\mathcal{S}_1$ and $\mathcal{S}_2$:

$$\sqrt{\frac{1}{|\mathcal{S}_1|^2}\sum_{\mathbf{x},\mathbf{x}'\in\mathcal{S}_1} k(\mathbf{x},\mathbf{x}') + \frac{1}{|\mathcal{S}_2|^2}\sum_{\mathbf{x},\mathbf{x}'\in\mathcal{S}_2} k(\mathbf{x},\mathbf{x}') - \frac{2}{|\mathcal{S}_1||\mathcal{S}_2|}\sum_{\mathbf{x}\in\mathcal{S}_1,\mathbf{x}'\in\mathcal{S}_2} k(\mathbf{x},\mathbf{x}')}.$$

1.3.3 The Representer Theorem

The kernel trick is straightforward when one thinks of kernels as inner products (see subsection 1.2.3), and is a convenient guideline to deriving kernel methods from linear algorithms. When one thinks of kernels as regularization operators (see subsection 1.2.5), a simple but deep theorem can help understand many kernel methods in a different light. This theorem, called the representer theorem, was first stated less generally by Kimeldorf and Wahba (1971):

Representer theorem

Theorem 1.4 *Let $\mathcal{X}$ be a set endowed with a kernel k, and $\mathcal{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \subset \mathcal{X}$ a finite set of objects. Let $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ be a function of $n+1$ arguments, strictly monotonic increasing in its last argument. Then any solution of the problem*

$$\min_{f\in\mathcal{H}_k} \Psi\left(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n), ||f||_{\mathcal{H}_k}\right), \tag{1.19}$$

where $(\mathcal{H}_k, ||.||_{\mathcal{H}_k})$ is the RKHS associated with k, admits a representation of the form

$$\forall \mathbf{x} \in \mathcal{X}, \quad f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{1.20}$$

Proof With the notations of theorem 1.4, let us call $\xi(f, \mathcal{S})$ the function to be minimized in (1.19), and let

$$\mathcal{H}_k^{\mathcal{S}} = \left\{ f \in \mathcal{H}_k : f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}), (\alpha_1, \cdots, \alpha_n) \in \mathbb{R}^n \right\} \subset \mathcal{H}_k.$$

Any function $f \in \mathcal{H}_k$ can be decomposed as $f = f_{\mathcal{S}} + f_\perp$, where $f_{\mathcal{S}} \in \mathcal{H}_k^{\mathcal{S}}$ is the orthogonal projection of f onto the subspace $\mathcal{H}_k^{\mathcal{S}}$ and $f_\perp \perp \mathcal{H}_k^{\mathcal{S}}$. By (1.11) it follows that $f_\perp(\mathbf{x}_i) = \langle f_\perp, k(\mathbf{x}_i, .)\rangle = 0$ for $i = 1, \cdots, n$, because each function $k(\mathbf{x}_i, .)$ is an element of $\mathcal{H}_k^{\mathcal{S}}$ and $f_\perp$ is orthogonal to each element of $\mathcal{H}_k^{\mathcal{S}}$ by definition. Therefore $f(\mathbf{x}_i) = f_{\mathcal{S}}(\mathbf{x}_i)$ for each $i = 1, \cdots, n$. Moreover, Pythagoras theorem in $\mathcal{H}_k$ states that $||f||^2_{\mathcal{H}_k} = ||f_{\mathcal{S}}||^2_{\mathcal{H}_k} + ||f_\perp||_{\mathcal{H}_k}$. This shows that $\xi(f, \mathcal{S}) \geq \xi(f_{\mathcal{S}}, \mathcal{S})$, with equality iff $||f_\perp||_{\mathcal{H}_k} = 0$, or $f_\perp = 0$, because Ψ is strictly monotonic increasing

in its last argument. As a result, any minimum f of $\xi(f, \mathcal{S})$ must belong to $\mathcal{H}_k^{\mathcal{S}}$, which concludes the proof. ■

Theorem 1.4 shows the dramatic effect of regularizing a problem by including a dependency in $||f||_{\mathcal{H}_k}$ in the function to optimize. As pointed out in subsection 1.2.5, this penalization makes sense because it forces the solution to be smooth, which is usually a powerful protection against overfitting of the data. The representer theorem shows that this penalization also has substantial computational advantages: any solution to (1.19) is known to belong to a subspace of $\mathcal{H}_k$ of dimension at most n, the number of points in $\mathcal{S}$, even though the optimization is carried out over a possibly infinite-dimensional space $\mathcal{H}_k$. A practical consequence is that (1.19) can be reformulated as an n-dimensional optimization problem, by plugging (1.20) into (1.19) and optimizing over $(\alpha_1, \cdots, \alpha_n) \in \mathbb{R}^n$.

Most kernel methods can be seen in light of the representer theorem. Indeed, as we show in the next examples, they often output a function of the subspace $\mathcal{H}_k^{\mathcal{S}}$; indeed one can often explicitly write the functional that is minimized, which involves a norm in $\mathcal{H}_k$. This observation can serve as a guide to choosing a kernel for practical applications, if one has some prior knowledge about the function the algorithm should output: it is in fact possible to design a kernel such that a priori desirable functions have a small norm.

1.3.4 Example: Kernel Principal Component Analysis

PCA

PCA (Jolliffe, 1986) is a powerful method to extract features from a set of vectors and to visualize them. Let us first suppose that $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is a set of centered vectors,

$$\sum_{i=1}^{n} \mathbf{x}_i = 0.$$

The orthogonal projection onto a direction $\boldsymbol{w} \in \mathbb{R}^p$ is the function $h_{\boldsymbol{w}} : \mathcal{X} \to \mathbb{R}$ defined by

$$h_{\boldsymbol{w}}(\mathbf{x}) = \mathbf{x}^\top \frac{\boldsymbol{w}}{||\boldsymbol{w}||}. \tag{1.21}$$

As illustrated in figure 1.5, PCA finds successive directions $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_p$ for which the projections $h_{\boldsymbol{w}_i}$ have maximum empirical variance and $\boldsymbol{w}_i$ is orthogonal to $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{i-1}$, for $i = 1, \ldots, p$. Here the empirical variance of a projection $h_{\boldsymbol{w}}$ is defined by

$$\hat{var}(h_{\boldsymbol{w}}) := \frac{1}{n} \sum_{i=1}^{n} h_{\boldsymbol{w}}(\mathbf{x}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} \frac{\left(\mathbf{x}_i^\top \boldsymbol{w}\right)^2}{||\boldsymbol{w}||^2}. \tag{1.22}$$

There may be ambiguity in this definition if two different directions have the same empirical variance, which we will not discuss further in the hope of keeping the attention of the reader on the kernelization of PCA.

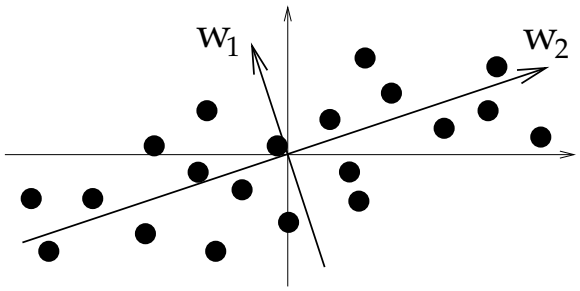

Figure 1.5 For centered vectorial data, principal component analysis (PCA) finds the orthogonal directions of largest variations.

Let us now rephrase PCA in terms of functional optimization (Schölkopf and Smola, 2002). Let k_L be the linear kernel (1.1), and $\mathcal{H}_k$ the RKHS (1.7) associated with k_L. Given any direction $\boldsymbol{w} \in \mathbb{R}^d$, we use (1.8) to associate the function $f_{\boldsymbol{w}} \in \mathcal{H}_k$ defined by $f_{\boldsymbol{w}}(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x}$, with norm $||f_{\boldsymbol{w}}|| = ||\boldsymbol{w}||$. Here, $||f_{\boldsymbol{w}}||$ is understood as the norm of f in $\mathcal{H}_k$, while $||\boldsymbol{w}||$ is the Euclidean norm of w in $\mathbb{R}^d$. The empirical variance (1.22) of the projection onto $\boldsymbol{w}$ can therefore be expressed in terms of $f_{\boldsymbol{w}}$;

$$\forall \boldsymbol{w} \in \mathbb{R}^p, \quad \hat{var}\left(h_{\boldsymbol{w}}\right) = \frac{1}{n||f_{\boldsymbol{w}}||^2} \sum_{i=1}^{n} f_{\boldsymbol{w}}(\mathbf{x}_i)^2.$$

Moreover, orthogonality of two directions $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{R}^p$ is equivalent to the orthogonality of the corresponding functions $f_w, f'_w \in \mathcal{H}_k$ with respect to the dot product of $\mathcal{H}_k$. Linear PCA can therefore be rephrased as finding successive functions $f_1, \ldots, f_p \in \mathcal{H}_k$ defined recursively as follows: for $i = 1, \ldots, p$, f_i maximizes the functional

$$\forall f \in \mathcal{H}_k, \quad \Psi(f) := \frac{1}{n||f||^2} \sum_{j=1}^{n} f(\mathbf{x}_j)^2, \tag{1.23}$$

under the constraints of orthogonality with respect to $f_1, \ldots, f_{i-1}$. Here again, the definition has some ambiguity if several functions have the same value.

The functional (1.23) satisfies the conditions of theorem 1.4: it is a strictly decreasing function of $||f||$, and only depends on the values that f takes on the points $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Theorem 1.4 shows that the successive directions f_i, for $i = 1, \ldots, p$, admit a representation

$$\forall \mathbf{x} \in \mathcal{X}, \quad f_i(\mathbf{x}) = \sum_{j=1}^{n} \alpha_{i,j} k(\mathbf{x}_j, \mathbf{x}), \tag{1.24}$$

for some $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \ldots, \alpha_{i,n})^\top \in \mathbb{R}^n$ [indeed, the proof of theorem 1.4 remains valid when the optimization (1.19) is performed on a subspace of $\mathcal{H}_k$, such as a subspace

defined by orthogonality conditions]. Combining (1.24) and (1.10), we can express the norm $||f_i||$ in terms of $\boldsymbol{\alpha}_i$ using matrix notations,

$$||f_i||^2 = \boldsymbol{\alpha}_i^\top k \boldsymbol{\alpha}_i, \tag{1.25}$$

which with (1.24) yields

$$\sum_{i=1}^{n} f_i(\mathbf{x}_i)^2 = \boldsymbol{\alpha}_i^\top k^2 \boldsymbol{\alpha}_i. \tag{1.26}$$

Plugging (1.25) and (1.26) into (1.23), we obtain a dual formulation of PCA which consists in finding $\alpha_1, \ldots, \alpha_p \in \mathbb{R}^n$ defined recursively: for $i = 1, \ldots, p$, α_i maximizes the function

$$\frac{\boldsymbol{\alpha}^\top k^2 \boldsymbol{\alpha}}{n \boldsymbol{\alpha}^\top k \boldsymbol{\alpha}}, \tag{1.27}$$

under the constraints $\boldsymbol{\alpha}_i k \boldsymbol{\alpha}_j = 0$, for $j = 1, \ldots, i-1$. The principal components are then recovered by (1.24).

Classic linear algebra (Schölkopf et al., 1998) shows that the solutions $\boldsymbol{\alpha}_i$ of this problem are precisely the eigenvectors of k. In order to recover the projections (1.21) onto the principal directions, the eigenvector $\boldsymbol{\alpha}_i$ of k with eigenvalue λ_i must be scaled to ensure $1 = ||\boldsymbol{w}_i|| = \boldsymbol{\alpha}_i^\top k \boldsymbol{\alpha}_i = \lambda_i ||\boldsymbol{\alpha}_i||^2$, or $||\boldsymbol{\alpha}_i|| = 1/\sqrt{\lambda_i}$. Of course this conputation is only valid if the data are centered in the feature space. This is not a restriction, however, because any kernel matrix k can be transformed into a matrix $\tilde{k}$ corresponding to the inner products of the same points after centering in the feature space, using the kernel trick. The reader can check that $\tilde{k}$ is obtained by the formula $\tilde{k} = (I - e/n)k(I - e/n)$, where I is the identity matrix and e is the singular matrix with all entries equal to 1.

This representation of PCA only involves the diagonalization of the $n \times n$ matrix of pairwise comparisons with the linear kernel. It can therefore be kernelized by simply replacing this matrix with the same $n \times n$ matrix of pairwise comparisons obtained from a different kernel. In that case, linear PCA is performed implicitly in the corresponding feature space. The resulting method, called *kernel PCA*, is a useful tool to extract features from a set of objects in a space endowed with a kernel. For example, PCA involving a kernel defined on strings can be a useful visualization tool for sets of biological sequences. By projecting the sequences onto the first two or three principal components, one can observe the structure of the set of points, such as the presence of clusters or outliers, as shown in figure 1.6.

Kernel PCA

1.4 Support Vector Machines

Suppose that the data set $\mathcal{S}$ consists of a series of objects $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, together with a series of labels $y_1, \ldots, y_n \in \mathcal{Y}$ associated with the objects. SVMs are kernel

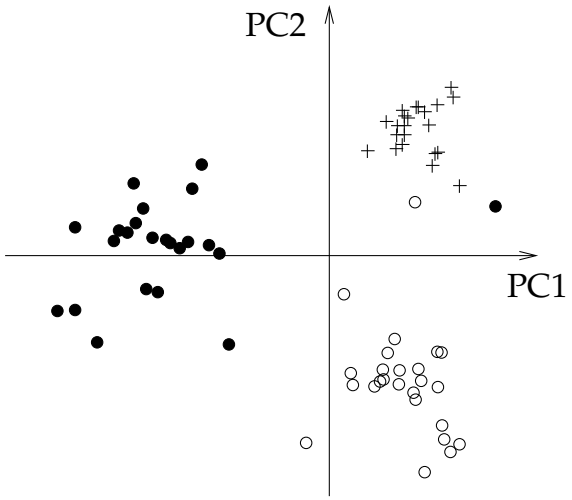

Figure 1.6 An example of kernel PCA. A set of 74 human tRNA sequences is analyzed using a kernel for sequences (the second-order marginalized kernel based on SCFG (Kin et al., 2002)). This set of tRNAs contains three classes, called Ala-AGC (*white circles*), Asn-GTT (*black circles*) and Cys-GCA (*plus symbols*). This plot shows the 74 sequences projected onto the first two principal components. By visual inspection, the three classes appear to be well separated in the feature space associated with the kernel. See also Vert (2002) for another example of kernel PCA application to biological data.

methods to learn a function $f : \mathcal{X} \to \mathcal{Y}$ from $\mathcal{S}$, which can be used to predict the label of any new object $\mathbf{x} \in \mathcal{X}$ by $f(\mathbf{x})$.

Pattern recognition

In this tutorial we only consider the simple case where each object is classified into one of two classes, indicated by the label $y \in \{-1, +1\}$. This simple problem, called *binary classification* or *pattern recognition* in the machine learning community, turns out to be very useful in practice. Examples of pattern recognition problems in computational biology include predicting whether a protein is secreted or not from its amino acid sequence, predicting whether a tissue is healthy from a gene profiling experiment, or predicting whether a chemical compound can bind a given target or not from its structure. In each case, a positive prediction is associated with the label $+1$, and a negative prediction with the label -1. In order to perform pattern recognition, one needs a data set of objects with known tags, such as a database of proteins known to be secreted or not, in order to learn a prediction function that can then be applied to proteins without annotation.

As usual with kernel methods, we begin with a description of the algorithm when objects are vectors, $\mathcal{X} = \mathbb{R}^p$. In this case, the SVM tries to separate the two classes of points using a linear function of the form $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$, with $\boldsymbol{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$. Such a function assigns a label $+1$ to the points $\mathbf{x} \in \mathcal{X}$ with $f(\mathbf{x}) \geq 0$, and a label -1 to the points $\mathbf{x} \in \mathcal{X}$ with $f(\mathbf{x}) < 0$. The problem is therefore to learn such a function f from a data set of observations $\mathcal{S}$.

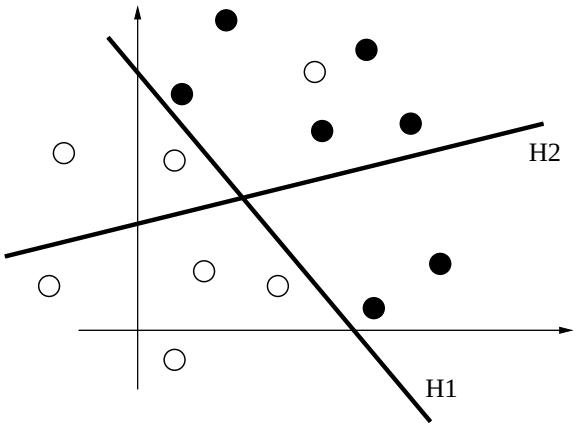

Figure 1.7 The hyperplane $H1$ discriminates the white circles from the black ones with 1 mistake. The hyperplane $H2$ separates these points with 5 mistakes. The empirical risk minimization principle states that one should choose a hyperplane with a minimal number of errors on the training set, which is $H1$ in this case

For a candidate function $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$, one can check for each observation $(\mathbf{x}_i, y_i)$ whether it is correctly classified by f, that is, whether $y_i f(\mathbf{x}_i) \geq 0$ or not. A natural criterion to choose f might be to minimize the number of classification errors on $\mathcal{S}$; the number of indices $i \in [1, n]$ such that $y_i f(\mathbf{x}_i) \geq 0$. This general principle, called *empirical risk minimization*, is illustrated in figure 1.7. This can, for example, be accomplished using the linear perceptron. As shown in figure 1.8, however, this usually does not define a unique solution, even when it is possible to perfectly separate the points.

Empirical risk minimization

SVMs are unique in that they focus more on the confidence of the classifications than on the number of misclassifications. This emphasis stems from general results on learning theory, developed in particular by Vapnik and Chervonenkis since the late 1960s (Vapnik and Chervonenkis, 1968, 1971, 1974). One way to formalize it with linear classifiers is shown in figure 1.9. The linear function $f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b$ defines two half-spaces of points classified positively and negatively with large confidence, namely the sets $h^+ = \{\mathbf{x} : f(\mathbf{x}) \geq 1\}$ and $h^- = \{\mathbf{x} : f(\mathbf{x}) \leq -1\}$. The distance between these two half-spaces, called the *margin*, is exactly equal to $2/||\boldsymbol{w}||$. If possible, one might require all points in the training set $\mathcal{S}$ to be correctly classified with strong confidence by a linear function f with largest possible margin. This would correspond to the problem of maximizing $2/||\boldsymbol{w}||$ under the constraints $y_i\left(\boldsymbol{w}^\top \mathbf{x} + b\right) \geq 1$ for $i = 1, \ldots, n$. In order to accommodate the cases when the training set cannot be correctly separated by a linear hyperplane, SVMs slightly modify this problem by softening the constraints using the continuous hinge loss function shown in figure 1.10,

Margin

Hinge loss

$$c(f, \mathbf{x}, y) = \max(0, 1 - y f(\mathbf{x})). \tag{1.28}$$

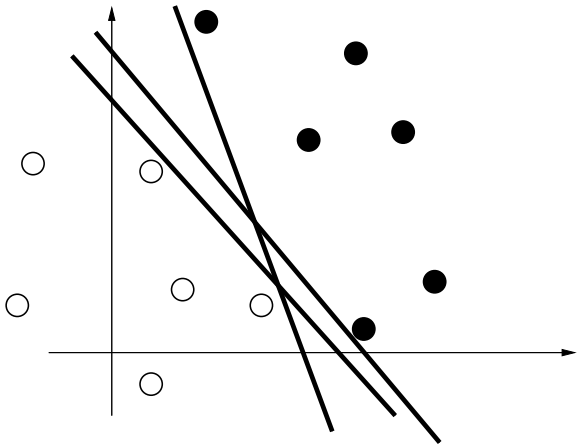

Figure 1.8 Even when the training data are linearly separable, the empirical risk minimization principle does not define a unique solution.

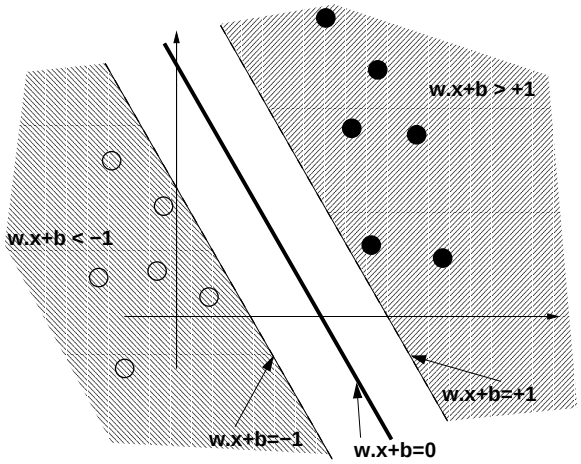

Figure 1.9 An affine function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ defines two half-spaces where points are classified with large confidence: $h^+ = \{\mathbf{x} : f(\mathbf{x}) \geq 1\}$ for the positive points (*black circles*) and $h^- = \{\mathbf{x} : f(\mathbf{x}) \leq -1\}$ (*white circles*). The distance between the half-spaces is equal to $1/\|\mathbf{w}\|$.

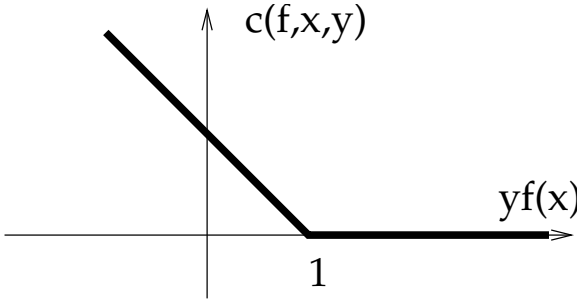

Figure 1.10 The hinge loss function. As long as $yf(\mathbf{x}) \geq 1$, the point $\mathbf{x}$ is correctly classified by the function f with large confidence and the hinge loss is null. When $yf(\mathbf{x}) < 1$, $\mathbf{x}$ is either correctly classified with small confidence ($0 \leq yf(\mathbf{x}) < 1$), or misclassified ($yf(\mathbf{x}) < 0$). In these cases the hinge loss is positive, and increases as $1 - yf(\mathbf{x})$. SVMs find a linear separating function with a large margin and small average hinge loss on the training set.

If a point $(\mathbf{x}, y)$ is correctly classified by f with large confidence, then $c(f, \mathbf{x}, y) = 0$. If this is not the case, then $c(f, \mathbf{x}, y)$ increases with the distance from $\mathbf{x}$ to the correct half-space of large confidence.

SVMs combine the requirements of large margin (i.e., small $||\boldsymbol{w}||$), and few misclassifications or classifications with little confidence on the training set, by solving the problem

$$\underset{f(\mathbf{x})=\boldsymbol{w}^\top \mathbf{x}+b}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{n} c(f, \mathbf{x}_i, y_i), \tag{1.29}$$

where C is a parameter that controls the tradeoff between the two requirements. Larger values of C might lead to linear functions with smaller margin but more examples correctly classified with strong confidence (the choice of this parameter is discussed in subsection 1.5.3).

Stated as (1.29), the reader might observe that this problem is very close to the minimization of a functional on a RKHS satisfying the hypothesis of theorem 1.4, with the slight difference that we consider here affine functions f, and not only linear functions of the form (1.7). It turns out that the representer theorem can be adapted to this case (see, e.g., theorem 4.3 in Schölkopf and Smola, 2002), and any $\boldsymbol{w}$ solution of (1.29) has an expansion as a linear combination of $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Let us now directly demonstrate this property and highlight several interesting properties of the solution of (1.29).

1.4.1 Solving the Optimization Problem

The hinge loss function (1.28) is not differentiable, so direct minimization of (1.29) is not straightforward. To overcome this issue let us introduce n new variables $\xi_1, \ldots, \xi_n$, called slack variables, and rewrite (1.29) as the problem of minimizing:

$$\underset{\boldsymbol{w},b,\xi_1,\ldots,\xi_n}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i, \tag{1.30}$$

under the constraints $\xi_i \geq c(f, \mathbf{x}_i, , y_i)$ for $i = 1, \ldots, n$. These two problems are equivalent, because the minimization of (1.30) with respect to ξ_i is obtained when ξ_i takes its minimal value, namely $c(f, \mathbf{x}_i, y_i)$. By definition of the hinge loss (1.28), the constraint $\xi_i \geq c(f, \mathbf{x}_i, y_i)$ is equivalent to the two constraints $\xi_i \geq 0$ and $\xi_i \geq 1 - y_i(\boldsymbol{w}^\top \mathbf{x}_i + b)$. We have therefore shown that (1.29) is equivalent to the quadratic programming problem

$$\min_{\boldsymbol{w},b,\xi_1,\ldots,\xi_n} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i, \tag{1.31}$$

under the constraints

$$\text{for } i = 1, \ldots, n, \quad \begin{cases} \xi_i \geq 0, \\ \xi_i - 1 + y_i(\boldsymbol{w}^\top \mathbf{x}_i + b) \geq 0. \end{cases} \tag{1.32}$$

Lagrange multipliers

This constrained optimization problem can be processed using *Lagrange multipliers*, which are written as $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \geq 0$ for each of the constraints $y_i\,(\mathbf{w}.\mathbf{x}_i + b) \geq 1 - \xi_i$, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \geq 0$ for each of the constraints $\xi_i \geq 0$. We can then introduce the Lagrangian,

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left[\xi_i - 1 + y_i\left(\boldsymbol{w}^\top \mathbf{x}_i + b\right)\right] - \sum_{i=1}^{n}\beta_i\xi_i. \tag{1.33}$$

In order to solve (1.31) we need to find the unique saddle point of L, which is a minimum with respect to $(\mathbf{w}, b, \boldsymbol{\xi})$ and a maximum with respect to $(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq 0$.

For fixed $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ let us first minimize the Lagrangian as a function of $(\mathbf{w}, b, \xi)$. This is done by setting the partial derivatives to 0;

$$\frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w} - \sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i = 0, \tag{1.34}$$

$$\frac{\partial L}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{n} y_i\alpha_i = 0, \tag{1.35}$$

$$\frac{\partial L}{\partial \xi_i}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = C - \alpha_i - \beta_i = 0, \quad \text{for } i = 1, \ldots, n. \tag{1.36}$$

With (1.34) we recover the representer theorem that states $\boldsymbol{w}$ is a linear combination of the $\mathbf{x}_1, \ldots, \mathbf{x}_n$. More precisely, we get

$$\boldsymbol{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i. \tag{1.37}$$

Plugging (1.37) into (1.33) and using (1.35), we obtain the value of the Lagrangian when minimized with respect to $(\mathbf{w}, b, \mathbf{x})$,

$$\forall (\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq 0, \quad \inf_{\mathbf{w}, b, \boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i, \tag{1.38}$$

under the constraints (1.35) and (1.36) on $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (if these constraints are not fulfilled, the infimum is equal to $-\infty$).

The function (1.38) has to be maximized with respect to $\boldsymbol{\alpha} \geq 0$ and $\boldsymbol{\beta} \geq 0$. But $\boldsymbol{\beta}$ does not appear in this function, so we just need to maximize (1.38) as a function of $\boldsymbol{\alpha}$ and to check that there exists some $\boldsymbol{\beta} \geq 0$ for which (1.36) holds. This is the case iff $\alpha_i \leq C$ for $i = 1, \ldots, N$, because only in this case can we find $\beta_i \geq 0$ such that $\beta_i + \alpha_i = C$.

As a result, the initial problem (1.31) is equivalent to the following *dual problem*:

Dual problem

find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ which minimizes

$$W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i, \tag{1.39}$$

under the constraints

$$\begin{cases} \sum_{i=0}^{n} y_i \alpha_i = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n. \end{cases}$$

Once $\boldsymbol{\alpha}$ is found one recovers the other dual vector $\boldsymbol{\beta}$ with the constraint

$$\beta_i = C - \alpha_i, \quad \text{for } i = 1, \ldots, N.$$

The vector $\boldsymbol{w}$ is then obtained from (1.37). In order to recover b, we can use the Karush-Kuhn-Tucker conditions which state that the constraints corresponding to non-zero Lagrange multipliers are met at the saddle point of the Lagrangian. As a result, for any $0 \leq i \leq n$ with $0 < \alpha_i < C$ (which implies $\boldsymbol{\beta} > 0$), the constraints $\xi_i = 0$ and $\xi_i - 1 + y_i \left(\boldsymbol{w}.\mathbf{x}_i + b\right)$ hold. We thus obtain

$$b = y_i - \boldsymbol{w}^\top \boldsymbol{x}_i = y_i - \sum_{j=1}^{n} y_j \alpha_j \mathbf{x}_j^\top \mathbf{x}_i. \tag{1.40}$$

Figure 1.11 shows a typical linear function learned by an SVM, together with the Lagrange multipliers α_i and β_i associated to each point. The constraint $\alpha_i < C$ implies $\beta_i > 0$, and therefore $\xi_i = 0$. These points are are correctly classified with large confidence. The constraints $0 < \alpha_i < C$ imply $\beta_i > 0$, and therefore

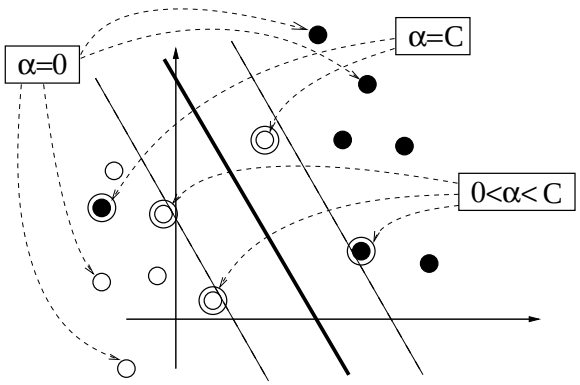

Figure 1.11 A linear function learned by an SVM to discriminate black from white circles, and the corresponding Lagrange multipliers $\boldsymbol{\alpha}$. Each point correctly classified with large confidence ($yf(\mathbf{x}) > 1$) has a null multiplier. Other points are called support vector. They can be on the boundary, in which case the multiplier satisfies $0 \leq \alpha \leq C$, or on the wrong side of this boundary, in which case $\alpha = C$.

$\boldsymbol{w}^\top \boldsymbol{x}_i + b = \boldsymbol{y}_i$. These points are correctly classified, but at the limit of the half-space of large confidence. Points not correctly classified with large confidence correspond to $\xi_i > 0$, and therefore $\beta_i = 0$ and $\alpha_i = C$.

Support vectors

The points with positive Lagrange multiplier $\alpha_i = 0$ are called *support vectors*. From (1.37) we see that $\boldsymbol{w}$ is a linear combination of the support vectors alone. Moreover, the solution found by the SVM does not change when non-support vectors are removed from the training set. Thus the set of support vectors contains all the information about the data set used by SVM to learn a discrimination function. This can easily be seen when it comes to predicting the class of a new object $\mathbf{x} \in \mathcal{X}$. Indeed we must then form the linear function

$$f(\mathbf{x}) = \boldsymbol{w}^\top \mathbf{x} + b = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^\top \mathbf{x} + b, \tag{1.41}$$

and predict that the class of $\mathbf{x}$ is -1 or $+1$ depending on the sign of this function. The sum in (1.41) only involves support vectors.

1.4.2 General SVMs

From (1.39), (1.40), and (1.41), we see that learning a linear classifier and predicting the class of a new point only involves the points in the training set through their dot products. The kernel trick can therefore be applied to perform the SVM algorithm in the feature space associated with a general kernel. It can be stated as follows: find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ which minimizes

$$W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n} \alpha_i, \tag{1.42}$$

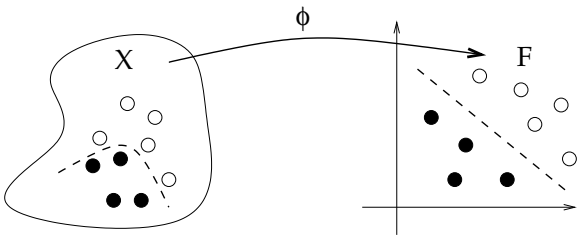

Figure 1.12 SVMs perform a linear discrimination of a training set of labeled points in the feature space associated with a kernel. The resulting separation can be nonlinear in the original space.

under the constraints

$$\begin{cases} \sum_{i=0}^{n} y_i \alpha_i = 0, \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, n. \end{cases}$$

Next, find an index i with $0 < \alpha_i < C$, and set:

$$b = y_i - \sum_{j=1}^{n} y_j \alpha_j k\left(\mathbf{x}_j, \mathbf{x}_i\right).$$

The classification of a new object $\mathbf{x} \in \mathcal{X}$ is then based on the sign of the function

$$f\left(\mathbf{x}\right) = \sum_{i=1}^{n} y_i \alpha_i k\left(\mathbf{x}_i, \mathbf{x}\right) + b. \tag{1.43}$$

The resulting function, although linear in the feature space associated with the kernel, can of course be nonlinear if the initial space is a vector space and the kernel is nonlinear. An example of nonlinear separation is illustrated in figure 1.12.

1.4.3 Variants and Extensions

Many variants of the basic SVM algorithm presented in the preceding sections have been proposed. Among the many variants surveyed in Schölkopf and Smola (2002), let us mention here a few directions to generalize the basic SVM algorithm. First, in the case of binary classification, several interesting modifications are obtained by changing the function (1.29) being optimized. Different norms on $\boldsymbol{w}$ together with different cost functions lead to interesting variants, which can be more or less difficult to solve. Lack of space prevents us from being exhaustive, so we will have to skip interesting approaches such as the *leave-one-out machine* (Weston, 1999; Weston and Herbrich, 2000) and the *Bayes point machines* (Ruján and Marchand, 2000; Herbrich et al., 2001; Rychetsky et al., 2000), as well as algorithms for tasks which are different from pattern recognition, such as regression estimation (Vapnik, 1995) and novelty detection (Schölkopf et al., 2001).

1.4.4 ν-SVMs

An alternative realization of a soft margin SVM (1.29) uses the ν-parametrization (Schölkopf et al., 2000). In this approach, the parameter C is replaced by $\nu \in [0,1]$, which can be shown to lower- and upper-bound the number of examples that will be support vectors and that will come to lie on the wrong side of the hyperplane, respectively. In many situations, this provides a more natural parameterization than that using the somewhat unintuitive parameter C. The so-called ν-SVM uses a primal objective function with the error term $\frac{1}{\nu m}\sum_i \xi_i - \rho$, and separation constraints

ν-SVM

$$y_i \cdot ((\boldsymbol{w} \cdot \boldsymbol{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m. \tag{1.44}$$

The margin parameter ρ is a variable of the optimization problem. The dual can be shown to consist of maximizing the quadratic part of (1.39), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$.

1.4.5 Linear Programming Machines

The idea of linear programming (LP) machines is to use the kernel expansion $f(x) = \sum_{i=1}^m \upsilon_i k(x, x_i) + b$ [cf. (1.43)] as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector (Mangasarian, 1965; Frieß and Harrison, 1998; Mattera et al., 1999; Bennett, 1999; Weston et al., 1999). The main motivation for this is that this regularizer is known to induce sparse solutions. This amounts to the objective function

$$R_{\mathrm{reg}}[g] := \frac{1}{m}\|\upsilon\|_1 + C\, R_{\mathrm{emp}}[g], \tag{1.45}$$

where $\|\upsilon\|_1 = \sum_{i=1}^m |\upsilon_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\mathrm{emp}}[g] = \frac{1}{m}\sum_i \xi_i, \tag{1.46}$$

with slack terms

$$\xi_i = \max\{1 - y_i f(x_i), 0\}. \tag{1.47}$$

We thus obtain the LP problem

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\xi} \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{m}\sum_{i=1}^m (\alpha_i + \alpha_i^*) + C\sum_{i=1}^m \xi_i, \tag{1.48}$$

subject to

$$\begin{cases} y_i f(x_i) \geq 1 - \xi_i, \\ \alpha_i, \alpha_i^*, \xi_i \geq 0. \end{cases}$$

Here, the ℓ_1-norm has been componentwise split into positive and negative parts, that is, $v_i = \alpha_i - \alpha_i^*$. The solution differs from (1.43) in that each expansion pattern no longer necessarily has a weight $\alpha_i y_i$ with a sign equal to its class label; nor do the expansion patterns lie on or beyond the margin — in LP machines they can basically be anywhere, a feature which is reminiscent of the *relevance vector machine* (Tipping, 2001).

LP machines can also benefit from the ν-trick. In this case, the programming problem can be shown to take the following form (Graepel et al., 1999b):

$$\min_{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b,\rho\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho, \tag{1.49}$$

subject to

$$\begin{cases} \frac{1}{m}\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) = 1, \\ y_i f(\boldsymbol{x}_i) \geq \rho - \xi_i, \\ \alpha_i, \alpha_i^*, \xi_i, \rho \geq 0. \end{cases}$$

1.5 SVMs in Practice

SVM implementations

A number of SVM implementations are freely or commercially available: For instance, SVM light,[3] LIBSVM, [4] and mySVM [5] are popular in the machine learning community, and a more complete and up-to-date list is available on the kernel method community website `http://www.kernel-machines.org`. While newcomers may feel that these programs can solve the learning tasks automatically, it in fact remains challenging to apply SVMs in a fully automatic manner. Questions regarding the choice of kernel, of parameters, of data representation, or of different flavors of SVMs, remain largely empirical in real-world applications. While default setting and parameters are generally useful as a starting point, big improvements can result from careful tuning of the algorithm. As an example, Hsu et al. (2003) report an accuracy improvement from 36% to 85.2% by an appropriate tuning.

1.5.1 Multiclass Problems

The basic SVM algorithm for pattern recognition is designed for classification of objects into two classes, but many real-world applications deal with more than two classes. This is, for example, the case when one wants to assign a function or a structure to a protein sequence, or a disease family to a tissue from gene expression

3. Available from `http://svmlight.joachims.org/`
4. Available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
5. Available from `http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/`

experiments. One way to use SVMs in this context is to apply an implementation that specifically solves multiclass problems (see, e.g., chapter **??**). However, these implementations remain rare and cannot handle more than a few classes.

The most widely used method for multiclass problems involves reformulating them as a number of binary classification problems, and solving these problems with binary SVMs. The resulting SVMs must then be combined to form a multiclass prediction algorithm. The most common way to perform this split and combination is called the *One-against-all* scheme. It consists in first finding a discrimination between each class and all the others, thus transforming a problem with N classes into N binary problems. The scores output by each SVM are then combined by a *max rule*: an object is assigned to the class corresponding to the SVM that outputs the largest score. As an example, let us consider a three-class classification problem with the following training set labels:

One-against-all

$$\boldsymbol{y} = (1, 1, 1, 2, 2, 2, 3, 3, 3).$$

In the one-against-all scheme this problem is decomposed as three binary problems with the following class assignments:

$$\begin{aligned} \boldsymbol{y}_1 &= (1, 1, 1, -1, -1, -1, -1, -1, -1) \\ \boldsymbol{y}_2 &= (-1, -1, -1, 1, 1, 1, -1, -1, -1) \\ \boldsymbol{y}_3 &= (-1, -1, -1, -1, -1, -1, 1, 1, 1) \end{aligned} \tag{1.50}$$

Three SVMs are trained on the three class labels respectively. When an unknown sample is classified, the outputs of SVMs are compared and the sample is assigned to the class with the largest output.

We conclude this subsection by noting that several other methods for multiclass problems have been proposed.

- In *pairwise classification*, one classifier is learned for each possible pair of classes (see Friedman, 1996; Schmidt and Gish, 1996; Kreßel, 1999).
- In *error-correcting output codes*, a set of classifiers is trained, each one solving the task of separating the union of certain classes from the complement. By cleverly choosing the classes, the outputs of several such classifiers code the class membership of a given test point rather robustly (Allwein et al., 2000)
- *Multiclass objective functions* capture a multiclass problem by defining an objective function that simultaneously trains all the classifiers involved (Weston and Watkins, 1999). While this may be the most elegant approach, it tends to be too expensive to train all classifiers simultaneously, if the problem size is large.

1.5.2 Kernel Normalization

When the data set is a set of vectors, it is often effective to linearly scale each attribute to zero mean and unit variance, and then apply the Gaussian RBF kernel

or polynomial kernel (Hsu et al., 2003). The main advantage of this normalization is to avoid attributes in larger numeric ranges dominating those in smaller ranges.

For more general kernels such as string or graph kernels, the kernel matrix is often directly obtained without feature vectors. In this case, it is considered effective to normalize the kernel matrix such that all the diagonal elements are 1. If k_{ij} denotes the (i, j)th element of a kernel matrix k, this normalization is

$$k'_{ij} = \frac{k_{ij}}{\sqrt{k_{ii}k_{jj}}}.$$

More advanced methods for kernel normalization are described by Schölkopf et al. (2002)

1.5.3 Parameter Setting

In order to use a basic SVM for binary classification, two kinds of parameters have to be determined:

- The regularization parameter C of the SVM
- The kernel and its parameters

A proper choice of these parameters is crucial to the good performance of the algorithm. A temptation to be avoided is to set the parameters based on the performance of the SVM on the training set, because this is likely to lead to overfitting: the performance increases on the training set used, but decreases on new samples.

Cross-validation

A standard way to fix parameters is to use cross-validation. Let us denote by γ a parameter of a kernel to be set, for instance, the width of the Gaussian RBF kernel, and C the parameter of the algorithm. Given specific values of C and γ, the k-fold cross-validation error is calculated as follows: first of all, the training set $\mathcal{Z} = \{x_i, y_i\}_{i=1}^n$ is randomly divided into k subsets $\mathcal{Z}_1, \cdots, \mathcal{Z}_k$ of approximately equal size. The SVM is trained on $k-1$ subsets and its error rate on the remaining subset is computed. Repeating this process k times such that each subset is tested once, the cross-validation error is determined by the average of the test errors. When $k = n$, the cross-validation error is especially called the *leave-one-out error*.

Grid search

In this scheme C and γ are determined so as to minimize the cross-validation error. This goal is approximately achieved by a *Grid search*. A set of candidate values are chosen both for C and γ, and the cross-validation error is computed for every possible combination of them. If n_c and n_γ are the number of candidate values, then the cross-validation error is computed $n_c n_\gamma$ times, which means that the SVM is trained $k n_c n_\gamma$ times in total. Typically, users do not have any idea about the optimal values for C and γ, so the candidate values must cover a very large domain. Hsu et al. (2003) suggest the candidate values be determined as an exponentially growing sequence (e.g., $C = 2^{-5}, 2^{-3}, \cdots, 2^{15}, \gamma = 2^{-15}, \cdots, 2^3$). When there are more than two parameters, the grid search becomes difficult as the

number of grid points grows exponentially. In such cases, one can use a gradient search to minimize an upper bound on the leave-one-out error (Chapelle et al., 2002).

Model selection can lead to costly computations. For example, when $k = n_c = n_\gamma = 10$, the SVM must be trained 1000 times to choose C and γ, which might be prohibitive for large data sets. However, this process can be easily parallelized, which alleviates the burden of cross-validation.

1.6 More Kernels

In section 1.3, we presented some of the possibilities for data analysis offered by kernel methods. They are completely modular, in the sense that each method can be applied to any kernel. In this section we present classic or recently developed kernels that the reader might find it useful to be aware of.

1.6.1 Kernels for Vectors

A few kernels for vectors have gained considerable attention in the SVM community. The *linear kernel* which we already met,

$$k_L(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}',$$

Polynomial kernels

is a particular instance of the *polynomial kernels* defined for $d \geq 0$ by

$$k_{Poly1}(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}'\right)^d,$$

or

$$k_{Poly2}(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top \mathbf{x}' + c\right)^d,$$

where d is the degree of the polynomial and c is a constant in the second kernel. The polynomial kernel k_{Poly1} of degree 2 corresponds to a feature space spanned by all products of 2 variables, that is, $\left\{x_1^2, x_1 x_2, x_2^2\right\}$. It is easy to see that the kernel k_{Poly2} of degree 2 corresponds to a feature space spanned by all products of at most 2 variables, that is, $\left\{1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\right\}$. More generally the kernel k_{Poly1} corresponds to a feature space spanned by all products of exactly d variables, while the kernel k_{Poly2} corresponds to a feature space spanned by all products of at most d variables.

Gaussian RBF kernel

The *Gaussian RBF kernel*

$$k_G(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right),$$

where σ is a parameter, is one of the most frequently used kernels in practice, thanks to its capacity to generate nonparametric classification functions. Indeed, the discriminant function learned by an SVM has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}||^2}{2\sigma^2}\right)$$

and is therefore a sum of Gaussian centered on the support vectors. Almost any decision boundary can be obtained with this kernel. Observe that the smaller the parameter σ, the more peaked the Gaussians are around the support vectors, and therefore the more complex the decision boundary can be. Larger σ corresponds to a smoother decision boundary.

Sigmoid kernel

The *sigmoid kernel* is defined by

$$k(\mathbf{x}, \mathbf{x}') = \tanh\left(\kappa \mathbf{x}^\top \mathbf{x}' + \theta\right),$$

where $\kappa > 0$ and $\theta < 0$ are parameters respectively called *gain* and *threshold*. The main motivation behind the use of this kernel is that the decision function learned by an SVM,

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \tanh\left(\kappa \mathbf{x}_i^\top \mathbf{x} + \theta\right),$$

is a particular type of two-layer sigmoidal neural network. In fact the sigmoid kernel is not always positive definite, but has still been successfully used in practice.

1.6.2 Kernels for Strings

Computational biology is a field rich in strings, such as peptide or nucleotide strings. As a result, much work has been devoted recently to the problem of making kernels for strings, as illustrated in chapters **??**, **??**, and **??**.

String kernels differ in the information about strings they encode, their implementation, and their complexity. In order to give a flavor of string kernels, we present a particular string kernel proposed by Lodhi et al. (2002) in the context of natural language processing, which is one of the earliest string kernels. The basic idea is to count the number of subsequences up to length n in a sequence, and compose a high-dimensional feature vector by these counts. The string kernel is defined as a dot product between such feature vectors.

More precisely, let Σ be the set of symbols. A string s of length $|s|$ is defined as $s = s_1, \ldots, s_{|s|} \in \Sigma^{|s|}$. The set of all strings is $\mathcal{X} = \bigcup_{i=0}^{\infty} \Sigma^i$. An index set $\boldsymbol{i}$ of length l is an l-tuple of positions in s,

$$\boldsymbol{i} = (i_1, \ldots, i_l), \quad 1 \leq i_1 < \ldots < i_l \leq |s|,$$

and we denote by $s[\boldsymbol{i}] = s_{i_1}, \ldots, s_{i_l}$ the subsequence of s corresponding to the indices in $\boldsymbol{i}$. Let us define the weight of the index set $\boldsymbol{i}$ by

$$\lambda^{l(\boldsymbol{i})}, \text{ where } \quad l(\boldsymbol{i}) = i_l - i_1 + 1,$$

where $\lambda < 1$ is a predetermined constant. Thus, for a given subsequence length l, the weight decreases exponentially with the number of gaps in the subsequence.

For each sequence $u \in \Sigma^k$, where k is fixed, let us now define a feature $\Phi_u : \mathcal{X} \to \mathbb{R}$ as

$$\forall s \in \mathcal{X}, \quad \Phi_u(s) = \sum_{\boldsymbol{i}: s[\boldsymbol{i}]=u} \lambda^{l(\boldsymbol{i})}.$$

Considering all sequences u of length n, we can map each sequence $s \in \mathcal{X}$ to a $|\Sigma|^n$-dimensional feature space by the mapping $s \to (\Phi_u(s))_{u \in \Sigma^n}$. We can then define a kernel for strings as the dot product between these representations,

$$\begin{aligned} \forall s, t \in \mathcal{X}, \quad k_n(s,t) &= \sum_{u \in \Sigma^n} \sum_{\boldsymbol{i}: u=s[\boldsymbol{i}]} \lambda^{l(\boldsymbol{i})} \sum_{\boldsymbol{j}: u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{j})} \\ &= \sum_{u \in \Sigma^n} \sum_{\boldsymbol{i}: u=s[\boldsymbol{i}]} \sum_{\boldsymbol{j}: u=t[\boldsymbol{j}]} \lambda^{l(\boldsymbol{i})+l(\boldsymbol{j})}. \end{aligned} \tag{1.51}$$

Calculating each feature is hopeless because of the high dimensionality. However, it has been shown that a recursive algorithm can calculate k_n efficiently with a time complexity $O(n|s||t|)$ (Lodhi et al., 2002), using dynamic programming.

1.6.3 The Fisher Kernel

Probabilistic models are convenient to represent families of complex objects that arise in computational biology. Typically, such models are useful when one wants to characterize a family of objects $\mathbf{x}$ that belong to a big set $\mathcal{X}$, but only span a very small subset of $\mathcal{X}$. The models can then be used to infer a probability distribution on $\mathcal{X}$ concentrated on the objects observed or likely to be observed. For example, hidden Markov models (HMMs) are a central tool for modeling protein families or finding genes from DNA sequences (Durbin et al., 1998). More complicated models called stochastic context-free grammars (SCFGs) are useful for modeling RNA sequences (Sakakibara et al., 1994).

The success of a particular probabilistic model requires that the distribution of actual objects be well characterized by that model. The Fisher kernel (Jaakkola and Haussler, 1999) provides a general principle to design a kernel for objects well modeled by a probabilistic distribution, or more precisely a parametric statistical model. Denote by $p(\mathbf{x}|\boldsymbol{\theta}),\ \mathbf{x} \in \mathcal{X}, \boldsymbol{\theta} \in \Re^p$ a parametric statistical model with a p-dimensional parameter $\boldsymbol{\theta}$ on the measurable space;

$$\forall \theta \in \Theta, \quad \int_{\mathcal{X}} p(\mathbf{x}|\theta) d\mathbf{x} = 1.$$

Moreover, $(\mathbf{x}, \theta) \to p(\mathbf{x}|\theta)$ is required to be smooth enough for all following computations to make sense.

Given a sample $\mathcal{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, suppose a parameter $\hat{\boldsymbol{\theta}}$ is estimated to model $\mathcal{S}$, for example, by maximum likelihood. The Fisher kernel is then defined as

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}})^\top J^{-1} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}'|\hat{\boldsymbol{\theta}}),$$

where

$$\nabla_{\boldsymbol{\theta}} = \left(\frac{\partial}{\partial \theta_1}, \cdots, \frac{\partial}{\partial \theta_p} \right)^\top$$

is a gradient vector with respect to $\boldsymbol{\theta}$ and J is the Fisher information matrix;

$$J = \int_{\mathbf{x} \in \mathcal{X}} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\hat{\boldsymbol{\theta}})^\top p(\mathbf{x}|\hat{\boldsymbol{\theta}}) dx.$$

The Fisher kernel can be understood intuitively when the parametric model is an exponential family. An exponential family of densities is written as

$$p(\mathbf{x}|\boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^\top \boldsymbol{s}(\mathbf{x}) + \phi(\boldsymbol{\theta})),$$

where $\boldsymbol{s} : \mathcal{X} \to \Re^p$ is a vector-valued function and ϕ is a normalization factor to ensure that $\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}|\boldsymbol{\theta}) = 1$. The function $\boldsymbol{s}$, commonly called "sufficient statistics," plays a role of feature extraction from x, because $p(x|\boldsymbol{\theta})$ depends on x solely through $\boldsymbol{s}$. The Fisher kernel can recover this "hidden" feature of x because

$$\nabla_{\boldsymbol{\theta}} \log p(x|\hat{\theta}) = \boldsymbol{s}(x) + \nabla_{\boldsymbol{\theta}} \phi(\hat{\boldsymbol{\theta}}),$$

and the second term is a constant independent of x. Usually the Fisher kernel is applied to complicated probability distributions which do not belong to the exponential family (e.g., HMMs). However, the Fisher kernel can still effectively reveal the features implicitly used in a probabilistic model (see Tsuda et al., 2004, for details).

The first application of the Fisher kernel was in the context of the protein remote homology detection, in combination with SVMs, where it outperformed all other state-of-the-art methods (Jaakkola et al., 2000). Extensions to the Fisher kernel idea can be seen, for example, in Tsuda et al. (2002a,b), Sonnenburg et al. (2002), and Seeger (2002).

1.7 Designing Kernels

As suggested in the previous section, a wide choice of kernels already exists. Many data or applications may still benefit from the design of particular kernels, adapted specifically to a given task. In this section, we review several useful results and principles when one wants to design a new kernel, or even "learn" a kernel from the observed data.

1.7.1 Operations on Kernels

The class of kernel functions on a set $\mathcal{X}$ has several useful closure properties. It is a convex cone, which means that if k_1 and k_2 are two kernels, then any linear combination,

$$\lambda_1 k_1 + \lambda_2 k_2,$$

with $\lambda_1, \lambda_2 \geq 0$ is a kernel.

The set of kernels is also closed under the topology of pointwise convergence, which means that if one has a family of kernel $(k_i)_{i\in\mathbb{N}}$ that converges in a pointwise fashion to a function,

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \lim_{n\to\infty} k_n(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}'),$$

then k is a kernel.

Other useful properties include closure under the pointwise multiplication, also called the Schur product (Schur, 1911): if k_1 and k_2 are two kernels, then

$$k(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}')$$

is also a kernel. From this and the closure under pointwise limit we can deduce a useful corollary: if $f(z) = \sum_{i=0}^{\infty} a_i z_i$ is holomorphic in $\{z \in \mathbb{C} : |z| < \rho\}$, and if k is a kernel such that $|k(\mathbf{x}, \mathbf{x}')| < \rho$ for any $\mathbf{x}, \mathbf{x}'$, then $f \circ k$ is a valid kernel. As an example, for any kernel k, $\exp(k)$ is a valid kernel, and for any bounded kernel $|k| < \rho$, $(\rho - \mathbf{x})^{-1}$ is a valid kernel.

On the other hand, other operations on kernels are in general forbidden. For example, if k is a kernel, then $\log(k)$ is not positive definite in general, and neither is k^β for $0 < \beta < 1$. In fact these two operations are linked by the following result: k^β is positive definite for any $\beta > 0$ iff $\log(k)$ is *conditionally positive definite*:

Conditionally positive definite

$$\sum_{i,j=1}^{n} c_i c_j \log(k(\mathbf{x}_i, \mathbf{x}_j)) \geq 0$$

for any $n > 0$, $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and $c_1, \ldots, c_n \in \mathbb{R}$ with the additional constraint that $\sum_{i=1}^{n} c_i = 0$. Such a kernel is called *infinitely divisible*. These considerations are, for example, discussed in chapter **??**.

1.7.2 Translation-Invariant Kernels and Kernels on Semi-Groups

When $\mathcal{X} = \mathbb{R}^p$, the class of translation-invariant kernels is defined as the class of kernels of the form

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x} - \mathbf{x}'),$$

for some function $\psi : \mathbb{R}^p \to \mathbb{R}$. The Gaussian kernel (1.5) is an example of a translation-invariant kernel. These kernels are particular examples of *group kernels*:

Group kernel

if $(\mathcal{X}, .)$ is a group,[6] then group kernels are defined as functions of the form $k(\mathbf{x}, \mathbf{x}') = \psi\left(\mathbf{x}^{-1}\mathbf{x}'\right)$, with $\psi : \mathcal{X} \to \mathbb{R}$. Conditions on ψ to ensure that k is a symmetric positive definite kernel have been studied in relation to harmonic analysis on groups and semigroups; the interested reader should consult Berg et al. (1984) for a complete treatment of these conditions for Abelian groups and semigroups. In the case $(\mathcal{X}, .) = (\mathbb{R}^p, +)$, the classic Bochner theorem states that if ψ is continuous, then k is a valid kernel iff ψ is the Fourier transform of a nonnegative finite measure. This is, for example, the case for the Gaussian RBF kernel. If $(\mathcal{X}, .)$ is a discrete Abelian semi-group, then k is a kernel iff ψ is the Fourier transform of a non-negative Radon measure. Such results can be extended to more general groups and semi-groups, and suggest principled ways to design kernels on sets with a group structure, such as the set of permutations, or on sets endowed with a group action. These kernels are related to the diffusion kernel presented in chapter **??**, which can be considered an efficient way to compute a group kernel if the graph is considered as the Cayley graph of a group.

1.7.3 Combining Kernels

Rather than design a kernel from scratch, one might be tempted to generate a kernel from a family of available kernels. In such cases, multiple kernel matrices $k_1, k_2, \cdots, k_c$ for the same set of objects are available. We might then wish to use kernel methods to combine this heterogeneous information; in other words, we would like to design a single kernel k from several basic kernels $k_1, \ldots, k_c$. A simple way to achieve this is to take the sum of the kernels:

$$k = \sum_{i=1}^{c} k_i.$$

This is clearly a valid kernel that can be interpreted as taking the direct product of the feature spaces of the basic kernels as a feature space. This approach was proposed in Pavlidis et al. (2002) in the context of functional genomics, and validated as a useful way to integrate heterogeneous data.

A slight generalization of this approach is to take a weighted sum,

$$k = \sum_{i=1}^{c} \mu_i k_i.$$

A nontrivial question is how to chose the weights automatically. Several approaches have been pioneered recently and are presented in forthcoming chapters: semidefinite programming in chapter **??**, kernel canonical correlation analysis in chapter **??**, and an information geometry-based approach in chapter **??**.

6. A group is a set with an associative operation, a neutral element, and such that any element has an inverse. If the operation is commutative, the group is called Abelian.

1.7.4 From Similarity Scores to Kernels

Another typical situation which arises when designing a kernel in computational biology, as well as in other fields, is when one has a "good" function to measure the similarity between objects, but which is unfortunately not a kernel. Such an example is, for instance, treated in detail in chapter **??**, where a kernel for biological sequences is built to mimic well-known measures of similarity between sequences. Other examples include the design of a kernel for molecular 3D structures from measures of structural similarity (see chapter **??**).

Empirical kernel map

Again, there is no single answer but rather a number of approaches that have been proposed and tested recently. Let $\mathcal{X}$ be a set and $s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a measure of similarity. One principled way to convert s into a valid kernel is called the *empirical kernel map* (Tsuda, 1999). It consists in first choosing a finite set of objects $t_1, \cdots, t_r \in \mathcal{X}$ called *templates*. An object $\mathbf{x} \in \mathcal{X}$ is then represented by a vector of similarity with respect to the template samples:

$$\mathbf{x} \in \mathcal{X} \to \phi(\mathbf{x}) = (s(x, t_1), \ldots, s(x, t_r))^\top \in \mathbb{R}^p.$$

The kernel is then defined as the dot product between two similarity vectors:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}) = \sum_{i=1}^{r} s(\mathbf{x}, t_i)\, s(\mathbf{x}', t_i).$$

Liao and Noble (2002) successfully applied this technique to transform an alignment score between protein sequences into a powerful kernel for remote homology detection However, one drawback of this method is that the results depend on the choice of template samples, as well as the fact that it can be computationally prohibitive.

In some cases, all objects to be processed by kernel methods are known in advance. This is the case, for example, when kernel PCA is performed on a finite set of objects, or when an SVM is trained in a transductive framework, that is, when the unannotated objects to be classified are known in advance. A good example of a transductive problem is in functional genomics on an organism: given the knowledge we have about the functions of some genes of an organism, can we predict the functions of the unannotated genes, which we know in advance because we know the whole genome.

In such cases, the problem boils down to making a symmetric positive definite matrix kernel matrix out of a pairwise similarity matrix. A natural way to perform this is by eigendecomposition of the similarity matrix (which is supposed to be symmetric), and removal of negative eigenvalues (Graepel et al., 1999a; Roth et al., 2003). Recently Roth et al. (2003) pointed out that this method preserves clusters in data, and showed promising experimental results in classifying protein sequences based on the FASTA scores.

Removal of negative eigenvalues

1.8 Conclusion

This short introductory tour of positive definite kernels and kernel methods only contains a very brief and partial summary of a field that has a long history, but was only recently investigated in depth in the context of empirical inference and machine learning. As highlighted by the different chapters of this book, this field is very active nowadays, with promising applications in computational biology.

References

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

S. Akaho. A kernel method for canonical correlation analysis. In *Proceedings of the 2000 Workshop on Information-Based Induction Sciences (IBIS2000),* 17–18 July 2000, Izu, Japan, pages 123–128, 2001.

E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

K. P. Bennett. Combining support vector and mathematical programming methods for induction. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—SV Learning*, pages 307–326, Cambridge, MA, MIT Press, 1999.

C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions.* New York, Springer Verlag, 1984.

B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, ACM Press, 1992.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis—Probabilistic Models of Proteins and Nucleic Acids.* Cambridge, UK, Cambridge University Press, 1998.

J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA, 1996.

T.-T. Frieß and R. F. Harrison. Linear programming support vector machines for pattern classification and regression estimation and the set reduction algorithm.

TR RR-706, University of Sheffield, Sheffield, UK, 1998.

C. Fyfe and P. L. Lai. ICA using kernel canonical correlation analysis. In *Proceedings of International Workshop on Independent Component Analysis and Blind Signal Separation (ICA2000)*, pages 279–284, Helsinki, 2000.

M. Girolami. Mercer kernel based clustering in feature space. *IEEE Transactions on Neural Networks*, 13, 2002.

J. Gorodkin, R.B. Lyngso, and G.D. Stormo. A mini-greedy algorithm for faster structural RNA stem-loop search. In H. Matsuda, S. Miyano, T. Takagi, and L. Wong, editors, *Genome Informatics 2001*, pages 184–193. Tokyo, Universal Academy Press, 2001.

T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 438–444. Cambridge, MA, MIT Press, 1999a.

T. Graepel, R. Herbrich, B. Schölkopf, A. J. Smola, P. L. Bartlett, K. Müller, K. Obermayer, and R. C. Williamson. Classification on proximity data with LP-machines. In *Ninth International Conference on Artificial Neural Networks*, Conference Publications No. 470, pages 304–309, London, Institution of Electrical Engineers (IEE), 1999b.

T. Graepel and K. Obermayer. Fuzzy topographic kernel clustering. In *Proceedings of the Fifth GI Workshop Fuzzy Neuro Systems*, pages 90–97, Munich, Germany, March 19–20, 1998.

S. Harmeling, A. Ziehe, M. Kawanabe, B. Blankertz, and K.-R. Müller. Nonlinear blind source separation using kernel feature spaces. In T.-W. Lee, editor, *Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation (ICA2001)*, pages 102–107, 2001.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, Springer Verlag, 2001.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, 1999.

R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, August 2001.

R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, MIT Press, 2000.

C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2): 95–114, 2000.

T.S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M.S. Kearns, S.A. Solla, and D.A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 487–493. Cambridge, MA, MIT Press, 1999.

I.T. Jolliffe. *Principal Component Analysis*. New York, Springer-Verlag, 1986.

G.S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

T. Kin, K. Tsuda, and K. Asai. Marginalized kernels for RNA sequence data analysis. In R.H. Lathtop, K. Nakai, S. Miyano, T. Takagi, and M. Kanehisa, editors, *Genome Informatics 2002*, pages 112–122. Tokyo, Universal Academic Press, 2002.

U. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods—Support Vector Learning*, pages 255–268, Cambridge, MA, MIT Press, 1999.

M. Kuss and T. Graepel. The geometry of kernel canonical correlation analysis. Technical Report 108, Max-Planck-Institut für biologische Kybernetik, Tübingen, Germany, 2003.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In G. Myers, S. Hannenhalli, D. Sankoff, S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the Sixth Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 225–232, New York, ACM Press, 2002.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

D. J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 133–165. Berlin, Springer-Verlag, 1998.

O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

D. Mattera, F. Palmieri, and S. Haykin. Simple and robust methods for support vector expansions. *IEEE Transactions on Neural Networks*, 10(5):1038–1047, 1999.

C. A. Micchelli. Algebraic aspects of interpolation. *Proceedings of Symposia in Applied Mathematics*, 36:81–102, 1986.

E. Parzen. Extraction and detection problems and reproducing kernel Hilbert spaces. *Journal of the Society for Industrial and Applied Mathematics. Series A, On control*, 1:35–62, 1962.

P. Pavlidis, J. Weston, J. Cai, and W. S. Noble. Learning gene functional classifications from multiple data types. *Journal of Computational Biology*, 9(2):401–411, 2002.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), September 1990.

R. Rosipal and L. J. Trejo. Kernel partial least squares regression in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 2:97–123, 2001.

V. Roth, J. Laub, J.M. Buhmann, and K.-R. Müller. Going metric: Denoising pairwise data. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 817–824. Cambridge, MA, MIT Press, 2003.

P. Ruján and M. Marchand. Computing the Bayes kernel classifier. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 329–347, Cambridge, MA, MIT Press, 2000.

M. Rychetsky, J. Shawe-Taylor, and M. Glesner. Direct Bayes point machines. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, San Francisco, Morgan Kaufmann, 2000.

Y. Sakakibara, M. Brown, R. Hughey, I.S Mian, K. Sjölander, R.C. Underwood, and D. Haussler. Stochastic context free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120, 1994.

M. Schmidt and H. Gish. Speaker identification via support vector classifiers. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 96)*, pages 105–108, Atlanta, May 1996.

B. Schölkopf. *Support vector learning*. Munich, Oldenbourg Verlag, 1997.

B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

B. Schölkopf, A. Smola, R.C. Williamson, and P.L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. Cambridge, MA, MIT Press, 2002.

B. Schölkopf, A.J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W. S. Noble. A kernel approach for learning from almost orthogonal patterns. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proceedings of ECML 2002, 13th European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages 511–528. Heidelberg, Springer Verlag, 2002.

I. Schur. Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen. *Journal für die Reine und Angewandte Mathematik*, 140: 1–29, 1911.

M. Seeger. Covariance kernels from Bayesian generative models. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, MIT Press, 2002.

S. Sonnenburg, G. Rätsch, A. Jagota, and K.-R. Müller. New methods for splice site recognition. In J.R. Dorronsoro, editor, *Artificial Neural Networks—ICANN 2002*, pages 329–336. Heidelberg, Springer Verlag, 2002.

J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. River Edge, NJ, World Scientific, 2002.

M.E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

K. Tsuda. Support vector classification with asymmetric kernel function. In M. Verleysen, editor, *Proceedings of the Seventh European Symposium on Artificial Neural Networks*, pages 183–188, 1999.

K. Tsuda, S. Akaho, M. Kawanabe, and K.-R. Müller. Asymptotic properties of the Fisher kernel. *Neural Computation*, 16:115–137, 2004.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K.R. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002a.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18:S268–S275, 2002b.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. New York, Springer Verlag, 1995. ISBN 0-387-94559-8.

V.N. Vapnik and A.Y. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Doklady Akademii nauk SSSR*, 181: 915–918, 1968.

V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

V.N. Vapnik and A.Y. Chervonenkis. *Theory of Pattern Recognition [In Russian]*. Moscow, Nauka, 1974.

J.-P. Vert. A tree kernel to analyze phylogenetic profiles. *Bioinformatics*, 18:S276–S284, 2002.

J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 1425–1432. Cambridge, MA, MIT Press, 2003.

G. Wahba. Soft and hard classification by reproducing kernel Hilbert space methods. *Proceedings of the National Academy of Sciences of the United States of America*, 99(26):16524 – 16530, 2002.

C. Watkins. Dynamic alignment kernels. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

H. L. Weinert, editor. *Reproducing Kernel Hilbert Spaces—Applications in Statistical Signal Processing*. Stroudsburg, PA, Hutchinson Ross, 1982.

J. Weston. Leave-one-out support vector machines. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 727–733. San Francisco, Morgan Kaufmann, 1999.

J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 873–880. Cambridge, MA, MIT Press, 2003.

J. Weston, A. Gammerman, M. Stitson, V.N. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 293–305. MIT Press, Cambridge, MA, 1999.

J. Weston and R. Herbrich. Adaptive margin support vector machines. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 281–296, Cambridge, MA, 2000. MIT Press.

J. Weston and C. Watkins. Multi-class support vector machines. In M. Verleysen, editor, *Proceedings of the Seventh European Symposium on Artificial Neural Networks*. Brussels, D Facto, 1999.

C. K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M.I. Jordan, editor, *Learning and Inference in Graphical Models*. Boston, Kluwer Academic Publishers, 1998.