1 Measuring Similarity with Kernels

1.1 Introduction

Over the last ten years, estimation and learning methods utilizing positive definite kernels have become rather popular, particularly in machine learning. Since these methods have a stronger mathematical slant than earlier machine learning methods (e.g., neural networks), there is also significant interest in the statistical and mathematical community for these methods. The present chapter aims to summarize the state of the art on a conceptual level. In doing so, we build on various sources (including Vapnik (1998); Burges (1998); Cristianini and Shawe-Taylor (2000); Herbrich (2002) and in particular Schölkopf and Smola (2002)), but we also add a fair amount of recent material which helps in unifying the exposition.

The main idea of all the described methods can be summarized in one paragraph. Traditionally, theory and algorithms of machine learning and statistics have been very well developed for the linear case. Real-world data analysis problems, on the other hand, often require nonlinear methods to detect the kind of dependences that allow successful prediction of properties of interest. By using a positive definite kernel, one can sometimes have the best of both worlds. The kernel corresponds to a dot product in a (usually high-dimensional) feature space. In this space, our estimation methods are linear, but as long as we can formulate everything in terms of kernel evaluations, we never explicitly have to work in the high-dimensional feature space.

1.2 Kernels

1.2.1 An Introductory Example

Suppose we are given empirical data

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}. \tag{1.1}$$

Here, the domain $\mathcal{X}$ is some nonempty set that the *inputs* x_i are taken from; the $y_i \in \mathcal{Y}$ are called *targets*. Here and below, $i, j = 1, \ldots, n$.

Note that we have not made any assumptions on the domain $\mathcal{X}$ other than it being a set. In order to study the problem of learning, we need additional structure. In

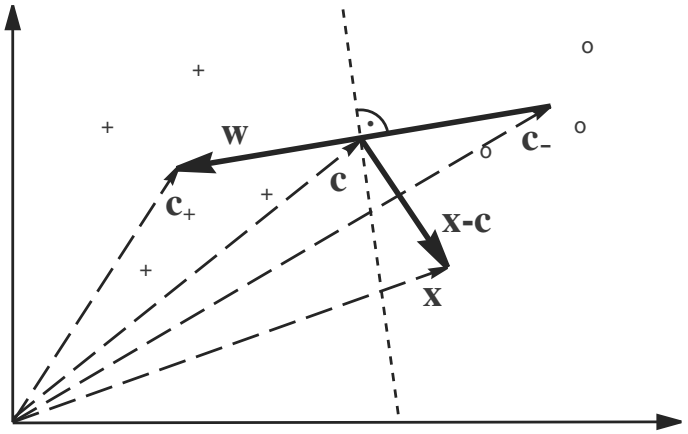

Figure 1.1 A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means c_+, c_- and assign a test input x to the one whose mean is closer. This can be done by looking at the dot product between $x - c$ (where $c = (c_+ + c_-)/2$) and $\mathbf{w} := c_+ - c_-$, which changes sign as the enclosed angle passes through $\pi/2$. Note that the corresponding decision boundary is a hyperplane (the dotted line) orthogonal to $\mathbf{w}$ (from Schölkopf and Smola (2002)).

learning, we want to be able to *generalize* to unseen data points. In the case of binary pattern recognition, given some new input $x \in \mathcal{X}$, we want to predict the corresponding $y \in \{\pm 1\}$. Loosely speaking, we want to choose y such that (x, y) is in some sense *similar* to the training examples. To this end, we need similarity measures in $\mathcal{X}$ and in $\{\pm 1\}$. The latter is easier, as two target values can only be identical or different.[1] For the former, we require a function

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \qquad (x, x') \mapsto k(x, x') \tag{1.2}$$

satisfying, for all $x, x' \in \mathcal{X}$,

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \,, \tag{1.3}$$

where Φ maps into some dot product space $\mathcal{H}$, sometimes called the *feature space*. The similarity measure k is usually called a *kernel*, and Φ is called its *feature map*.

kernels and feature map

The advantage of using such a kernel as a similarity measure is that it allows us to construct algorithms in dot product spaces. For instance, consider the following simple classification algorithm, where $\mathcal{Y} = \{\pm 1\}$. The idea is to compute the means of the two classes in the feature space, $c_+ = \frac{1}{n_+} \sum_{\{i:y_i=+1\}} \Phi(x_i)$, and $c_- = \frac{1}{n_-} \sum_{\{i:y_i=-1\}} \Phi(x_i)$, where n_+ and n_- are the number of examples with

1. When $\mathcal{Y}$ has a more complex structure, things can get complicated — this is the main topic of the present book, but we completely disregard it in this introductory example.

positive and negative target values, respectively. We then assign a new point $\Phi(x)$ to the class whose mean is closer to it. This leads to

$$y = \operatorname{sgn}(\langle \Phi(x), c_+ \rangle - \langle \Phi(x), c_- \rangle + b) \tag{1.4}$$

with $b = \frac{1}{2}\left(\|c_-\|^2 - \|c_+\|^2\right)$. Substituting the expressions for $c_\pm$ yields

$$y = \operatorname{sgn}\left(\frac{1}{n_+}\sum_{\{i:y_i=+1\}} \langle \Phi(x), \Phi(x_i)\rangle - \frac{1}{n_-}\sum_{\{i:y_i=-1\}} \langle \Phi(x), \Phi(x_i)\rangle + b\right). \tag{1.5}$$

Rewritten in terms of k, this reads

$$y = \operatorname{sgn}\left(\frac{1}{n_+}\sum_{\{i:y_i=+1\}} k(x, x_i) - \frac{1}{n_-}\sum_{\{i:y_i=-1\}} k(x, x_i) + b\right), \tag{1.6}$$

where $b = \frac{1}{2}\left(\frac{1}{n_-^2}\sum_{\{(i,j):y_i=y_j=-1\}} k(x_i, x_j) - \frac{1}{n_+^2}\sum_{\{(i,j):y_i=y_j=+1\}} k(x_i, x_j)\right)$. This algorithm is illustrated in figure 1.1 for the case that $\mathcal{X}$ equals $\mathbb{R}^2$ and $\Phi(x) = x$.

Let us consider one well-known special case of this type of classifier. Assume that the class means have the same distance to the origin (hence $b = 0$), and that $k(., x)$ is a density for all $x' \in \mathcal{X}$. If the two classes are equally likely and were generated from two probability distributions that are correctly estimated by the Parzen windows estimators

$$p_+(x) := \frac{1}{n_+}\sum_{\{i:y_i=+1\}} k(x, x_i), \qquad p_-(x) := \frac{1}{n_-}\sum_{\{i:y_i=-1\}} k(x, x_i), \tag{1.7}$$

then (1.6) is the Bayes decision rule.

The classifier (1.6) is quite close to the *support vector machine (SVM)* that we will discuss below. It is linear in the feature space (see (1.4)), while in the input domain, it is represented by a kernel expansion (1.6). In both cases, the decision boundary is a hyperplane in the feature space; however, the normal vectors are usually different.[2]

1.2.2 Positive Definite Kernels

We have above required that a kernel satisfy (1.3), i.e., correspond to a dot product in some dot product space. In the present section, we show that the class of kernels that can be written in the form (1.3) coincides with the class of positive definite kernels. This has far-reaching consequences. There are examples of positive definite

2. For (1.4), the normal vector is $w = c_+ - c_-$. As an aside, note that if we normalize the targets such that $\hat{y}_i = y_i/|\{j : y_j = y_i\}|$, in which case the $\hat{y}_i$ sum to zero, then $\|w\|^2 = \left\langle K, \hat{y}\hat{y}^\top \right\rangle_F$, where $\langle ., . \rangle_F$ is the Frobenius dot product. If the two classes have equal size, then up to a scaling factor involving $\|K\|_2$ and n, this equals the *kernel-target alignment* defined by Cristianini et al. (2002).

kernels which can be evaluated efficiently even though via (1.3) they correspond to dot products in infinite-dimensional dot product spaces. In such cases, substituting $k(x, x')$ for $\langle \Phi(x), \Phi(x') \rangle$, as we have done when going from (1.5) to (1.6), is crucial.

1.2.2.1 Prerequisites

Definition 1 (Gram Matrix) *Given a kernel k and inputs $x_1, \ldots, x_n \in \mathcal{X}$, the $n \times n$ matrix*

$$K := (k(x_i, x_j))_{ij} \tag{1.8}$$

is called the Gram matrix *(or* kernel matrix*) of k with respect to $x_1, \ldots, x_n$.*

Definition 2 (Positive Definite Matrix) *A real $n \times n$ symmetric matrix K_{ij} satisfying*

$$\sum_{i,j} c_i c_j K_{ij} \geq 0 \tag{1.9}$$

for all $c_i \in \mathbb{R}$ is called positive definite*. If for equality in (1.9) only occurs for $c_1 = \cdots = c_n = 0$, then we shall call the matrix* strictly positive definite*.*

Definition 3 (Positive Definite Kernel) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which for all $n \in \mathbb{N}, x_i \in \mathcal{X}$ gives rise to a positive definite Gram matrix is called a* positive definite kernel*. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which for all $n \in \mathbb{N}$ and distinct $x_i \in \mathcal{X}$ gives rise to a strictly positive definite Gram matrix is called a* strictly positive definite kernel*.*

Occasionally, we shall refer to positive definite kernels simply as a *kernels*. Note that for simplicity we have restricted ourselves to the case of real-valued kernels. However, with small changes, the below will also hold for the complex-valued case.

Since $\sum_{i,j} c_i c_j \langle \Phi(x_i), \Phi(x_j) \rangle = \left\langle \sum_i c_i \Phi(x_i), \sum_j c_j \Phi(x_j) \right\rangle \geq 0$, kernels of the form (1.3) are positive definite for any choice of Φ. In particular, if $\mathcal{X}$ is already a dot product space, we may choose Φ to be the identity. Kernels can thus be regarded as generalized dot products. While they are not generally bilinear, they share important properties with dot products, such as the Cauchy-Schwartz inequality:

Proposition 4 *If k is a positive definite kernel, and $x_1, x_2 \in \mathcal{X}$, then*

$$k(x_1, x_2)^2 \leq k(x_1, x_1) \cdot k(x_2, x_2). \tag{1.10}$$

Proof The 2×2 Gram matrix with entries $K_{ij} = k(x_i, x_j)$ is positive definite. Hence both its eigenvalues are nonnegative, and so is their product, K's determinant, i.e.,

$$0 \leq K_{11} K_{22} - K_{12} K_{21} = K_{11} K_{22} - K_{12}^2. \tag{1.11}$$

Substituting $k(x_i, x_j)$ for K_{ij}, we get the desired inequality. ∎

1.2.2.2 Construction of the Reproducing Kernel Hilbert Space

We now define a map from $\mathcal{X}$ into the space of functions mapping $\mathcal{X}$ into $\mathbb{R}$, denoted as $\mathbb{R}^{\mathcal{X}}$, via

$$\begin{aligned} \Phi : \mathcal{X} &\to \mathbb{R}^{\mathcal{X}} \\ x &\mapsto k(., x). \end{aligned} \tag{1.12}$$

Here, $\Phi(x) = k(., x)$ denotes the function that assigns the value $k(x', x)$ to $x' \in \mathcal{X}$.

We next construct a dot product space containing the images of the inputs under Φ. To this end, we first turn it into a vector space by forming linear combinations

$$f(.) = \sum_{i=1}^{n} \alpha_i k(., x_i). \tag{1.13}$$

Here, $n \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$ and $x_i \in \mathcal{X}$ are arbitrary.

Next, we define a dot product between f and another function $g(.) = \sum_{j=1}^{n'} \beta_j k(., x'_j)$ (with $n' \in \mathbb{N}$, $\beta_j \in \mathbb{R}$ and $x'_j \in \mathcal{X}$) as

$$\langle f, g \rangle := \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \beta_j k(x_i, x'_j). \tag{1.14}$$

To see that this is well-defined although it contains the expansion coefficients, note that $\langle f, g \rangle = \sum_{j=1}^{n'} \beta_j f(x'_j)$. The latter, however, does not depend on the particular expansion of f. Similarly, for g, note that $\langle f, g \rangle = \sum_{i=1}^{n} \alpha_i g(x_i)$. This also shows that $\langle \cdot, \cdot \rangle$ is bilinear. It is symmetric, as $\langle f, g \rangle = \langle g, f \rangle$. Moreover, it is positive definite, since positive definiteness of k implies that for any function f, written as (1.13), we have

$$\langle f, f \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geq 0. \tag{1.15}$$

Next, note that given functions $f_1, \ldots, f_p$, and coefficients $\gamma_1, \ldots, \gamma_p \in \mathbb{R}$, we have

$$\sum_{i,j=1}^{p} \gamma_i \gamma_j \langle f_i, f_j \rangle = \left\langle \sum_{i=1}^{p} \gamma_i f_i, \sum_{j=1}^{p} \gamma_j f_j \right\rangle \geq 0. \tag{1.16}$$

Here, the left-hand equality follows from the bilinearity of $\langle \cdot, \cdot \rangle$, and the right-hand inequality from (1.15).

By (1.16), $\langle \cdot, \cdot \rangle$ is a positive definite kernel, defined on our vector space of functions. For the last step in proving that it even is a dot product, we note that by (1.14), for all functions (1.13),

$$\langle k(., x), f \rangle = f(x), \tag{1.17}$$

and in particular

$$\langle k(.,x), k(.,x')\rangle = k(x,x'). \tag{1.18}$$

reproducing kernel

By virtue of these properties, k is called a *reproducing kernel* (Aronszajn, 1950) .

Due to (1.17) and proposition 4, we have

$$|f(x)|^2 = |\langle k(.,x), f\rangle|^2 \leq k(x,x)\cdot\langle f,f\rangle. \tag{1.19}$$

By this inequality, $\langle f,f\rangle = 0$ implies $f = 0$, which is the last property that was left to prove in order to establish that $\langle .,.\rangle$ is a dot product.

Skipping some details, we add that one can complete the space of functions (1.13) in the norm corresponding to the dot product, and thus get a Hilbert space H, called a *reproducing kernel Hilbert space (RKHS)*.

reproducing kernel Hilbert space(RKHS)

One can define an RKHS as a Hilbert space $\mathcal{H}$ of functions on a set $\mathcal{X}$ with the property that for all $x \in \mathcal{X}$ and $f \in \mathcal{H}$, the point evaluations $f \mapsto f(x)$ are continuous linear functionals (in particular, all point values $f(x)$ are well-defined, which already distinguishes RKHSs from many L_2 Hilbert spaces). From the point evaluation functional, one can then construct the reproducing kernel using the Riesz representation theorem. The Moore-Aronszajn theorem (Aronszajn, 1950) states that for every positive definite kernel on $\mathcal{X}\times\mathcal{X}$, there exists a unique RKHS and vice versa.

There is an analogue of the kernel trick for distances rather than dot products, i.e., dissimilarities rather than similarities. This leads to the larger class of *conditionally positive definite kernels*. Those kernels are defined just like positive definite ones, with the one difference being that their Gram matrices need to satisfy (1.9) only subject to

$$\sum_{i=1}^{n} c_i = 0. \tag{1.20}$$

Interestingly, it turns out that many kernel algorithms, including SVMs and kernel principal component analysis (PCA) (see section 1.3.2), can be applied also with this larger class of kernels, due to their being translation invariant in feature space (Schölkopf and Smola, 2002; Hein et al., 2005).

We conclude this section with a note on terminology. In the early years of kernel machine learning research, it was not the notion of positive definite kernels that was being used. Instead, researchers considered kernels satisfying the conditions of Mercer's theorem (Mercer, 1909); see e.g. Vapnik (1998) and Cristianini and Shawe-Taylor (2000). However, while all such kernels do satisfy (1.3), the converse is not true. Since (1.3) is what we are interested in, positive definite kernels are thus the right class of kernels to consider.

1.2.3 Constructing Kernels

constructing new kernels

In the following we demonstrate how to assemble new kernel functions from existing ones using elementary operations preserving positive definiteness. The following proposition will serve us as the main working horse:

Proposition 5 *Below, $k_1, k_2, \ldots$ are arbitrary positive definite kernels on $\mathcal{X} \times \mathcal{X}$, where $\mathcal{X}$ is a nonempty set.*
(i) The set of positive definite kernels is a closed convex cone, i.e., (a) if $\alpha_1, \alpha_2 \geq 0$, then $\alpha_1 k_1 + \alpha_2 k_2$ is positive definite.
(ii) The pointwise product $k_1 k_2$ is positive definite.
(iii) Assume that for $i = 1, 2$, k_i is a positive definite kernel on $\mathcal{X}_i \times \mathcal{X}_i$, where $\mathcal{X}_i$ is a nonempty set. Then the tensor product $k_1 \otimes k_2$ and the direct sum $k_1 \oplus k_2$ are positive definite kernels on $(\mathcal{X}_1 \times \mathcal{X}_2) \times (\mathcal{X}_1 \times \mathcal{X}_2)$.
(iv) If $k(x, x') := \lim_{n\to\infty} k_n(x, x')$ exists for all x, x', then k is positive definite.
(v) The function $k(x, x') := f(x)f(x')$ is a valid positive definite kernel for any function f.

Let us use this proposition now to construct new kernel functions.

1.2.3.1 Polynomial Kernels

From proposition 5 it is clear that homogeneous polynomial kernels $k(x, x') = \langle x, x' \rangle^p$ are positive definite for $p \in \mathbb{N}$ and $x, x' \in \mathbb{R}^d$. By direct calculation we can derive the corresponding feature map (Poggio, 1975):

$$\langle x, x' \rangle^p = \left\langle \sum_{j=1}^{d} [x]_j, [x']_j \right\rangle^p = \sum_{j \in [d]^p} [x]_{j_1} \cdot \cdots \cdot [x]_{j_p} \cdot [x']_{j_1} \cdot \cdots \cdot [x']_{j_p} = \langle C_p(x), C_p(x') \rangle \,, \tag{1.21}$$

where C_p maps $x \in \mathbb{R}^d$ to the vector $C_p(x)$ whose entries are all possible pth-degree ordered products of the entries of x. The polynomial kernel of degree p thus computes a dot product in the space spanned by all monomials of degree p in the input coordinates. Other useful kernels include the inhomogeneous polynomial,

$$k(x, x') = (\langle x, x' \rangle + c)^p \text{ where } p \in \mathbb{N} \text{ and } c \geq 0, \tag{1.22}$$

which computes all monomials up to degree p.

1.2.3.2 Gaussian Kernel

Using the infinite Taylor expansion of the exponential function $e^z = \sum_{i=1}^{\infty} \frac{1}{i!} z^i$, it follows from propostion 5(iv) that

$$e^{\gamma \langle x, x' \rangle}$$

is a kernel function for any $x, x' \in \mathcal{X}$ and $\gamma \in \mathbb{R}$. Therefore, it follows immediately that the widely used Gaussian function $e^{-\gamma||x-x'||^2}$ with $\gamma > 0$ is a valid kernel function. This can be seen as rewriting the Gaussian function as

$$e^{-\gamma||x-x'||^2} = e^{-\gamma\langle x,x\rangle} e^{2\gamma\langle x,x'\rangle} e^{-\gamma\langle x',x'\rangle},$$

and using proposition 5(ii).

We see that the Gaussian kernel corresponds to a mapping into $\mathcal{C}^\infty$, i.e. the space of continuous functions. However, the feature map is *normalized*, i.e. $||\Phi(x)||^2 = k(x,x) = 1$ for any $x \in \mathcal{X}$. Moreover, as $k(x,x') > 0$ for all $x, x' \in \mathcal{X}$, all mapped points lie inside the same orthant in feature space.

1.2.3.3 Spline Kernels

It is possible to obtain spline functions as a result of kernel expansions (Smola, 1996; Vapnik et al., 1997) simply by noting that convolution of an even number of indicator functions yields a positive kernel function. Denote by I_X the indicator (or characteristic) function on the set X, and denote by $\otimes$ the convolution operation, $(f \otimes g)(x) := \int_{\mathbb{R}^d} f(x')g(x'-x)dx'$. Then the B-spline kernels are given by

$$k(x,x') = B_{2p+1}(x-x') \text{ where } p \in \mathbb{N} \text{ with } B_{i+1} := B_i \otimes B_0. \tag{1.23}$$

Here B_0 is the characteristic function on the unit ball[3] in $\mathbb{R}^d$. From the definition of (1.23) it is obvious that for odd m we may write B_m as the inner product between functions $B_{m/2}$. Moreover, note that for even m, B_m is not a kernel.

1.2.4 The Representer Theorem

From kernels, we now move to functions that can be expressed in terms of kernel expansions. The representer theorem (Kimeldorf and Wahba, 1971; Cox and O'Sullivan, 1990) shows that solutions of a large class of optimization problems can be expressed as kernel expansions over the sample points. We present a slightly more general version of the theorem with a simple proof (Schölkopf et al., 2001). As above, $\mathcal{H}$ is the RKHS associated with the kernel k.

Theorem 6 (Representer Theorem) *Denote by $\Omega : [0,\infty) \to \mathbb{R}$ a strictly monotonic increasing function, by $\mathcal{X}$ a set, and by $c : (\mathcal{X} \times \mathbb{R}^2)^n \to \mathbb{R} \cup \{\infty\}$ an arbitrary loss function. Then each minimizer $f \in \mathcal{H}$ of the regularized risk functional*

$$c\left((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))\right) + \Omega\left(\|f\|_{\mathcal{H}}^2\right) \tag{1.24}$$

3. Note that in $\mathbb{R}$ one typically uses $\xi_{\left[-\frac{1}{2},\frac{1}{2}\right]}$.

admits a representation of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x). \tag{1.25}$$

Proof We decompose any $f \in \mathcal{H}$ into a part contained in the span of the kernel functions $k(x_1, \cdot), \cdots, k(x_n, \cdot)$, and one in the orthogonal complement:

$$f(x) = f_{\|}(x) + f_{\perp}(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) + f_{\perp}(x). \tag{1.26}$$

Here $\alpha_i \in \mathbb{R}$ and $f_{\perp} \in \mathcal{H}$ with $\langle f_{\perp}, k(x_i, \cdot)\rangle_{\mathcal{H}} = 0$ for all $i \in [n] := \{1, \ldots, n\}$. By (1.17) we may write $f(x_j)$ (for all $j \in [n]$) as

$$f(x_j) = \langle f(\cdot), k(x_j, .)\rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x_j) + \langle f_{\perp}(\cdot), k(x_j, .)\rangle_{\mathcal{H}} = \sum_{i=1}^{n} \alpha_i k(x_i, x_j). \tag{1.27}$$

Second, for all $f_{\perp}$,

$$\Omega(\|f\|_{\mathcal{H}}^2) = \Omega\left(\left\|\sum_{i}^{n} \alpha_i k(x_i, \cdot)\right\|_{\mathcal{H}}^2 + \|f_{\perp}\|_{\mathcal{H}}^2\right) \geq \Omega\left(\left\|\sum_{i}^{n} \alpha_i k(x_i, \cdot)\right\|_{\mathcal{H}}^2\right). \tag{1.28}$$

Thus for any fixed $\alpha_i \in \mathbb{R}$ the risk functional (1.24) is minimized for $f_{\perp} = 0$. Since this also has to hold for the solution, the theorem holds. ∎

Monotonicity of Ω does not prevent the regularized risk functional (1.24) from having multiple local minima. To ensure a global minimum, we would need to require convexity. If we discard the strictness of the monotonicity, then it no longer follows that each minimizer of the regularized risk admits an expansion (1.25); it still follows, however, that there is always another solution that is as good, and that *does* admit the expansion.

The significance of the representer theorem is that although we might be trying to solve an optimization problem in an infinite-dimensional space $\mathcal{H}$, containing linear combinations of kernels centered on *arbitrary* points of $\mathcal{X}$, it states that the solution lies in the span of n particular kernels — those centered on the training points. We will encounter (1.25) again further below, where it is called the *support vector expansion*. For suitable choices of loss functions, many of the α_i often equal zero.

1.3 Operating in Reproducing Kernel Hilbert Spaces

We have seen that kernels correspond to an inner product in some possibly high-dimensional feature space. Since direct computation in these spaces is computationally infeasible one might argue that sometimes the application of kernels is rather limited. However, in this section we demonstrate for some cases that direct opera-

tion in feature space is possible. Subsequently we introduce kernel PCA which can extract features corresponding to principal components in this high-dimensional feature space.

1.3.1 Direct Operations in RKHS

1.3.1.1 Translation

Consider the modified feature map $\tilde{\Phi}(x) = \Phi(x) + \Gamma$, with $\Gamma \in \mathcal{H}$. This feature map corresponds to a translation in feature space. The dot product $\left\langle \tilde{\Phi}(x), \tilde{\Phi}(x') \right\rangle$ yields for this case the terms

$$\langle \Phi(x), \Phi(x') \rangle + \langle \Phi(x), \Gamma \rangle + \langle \Gamma, \Phi(x') \rangle + \langle \Gamma, \Gamma \rangle ,$$

which cannot always be evaluated. However, let us restrict the translation Γ to be in the span of the functions $\Phi(x_1), \cdots, \Phi(x_n) \in \mathcal{H}$ with $\{x_1, \ldots, x_n\} \in \mathcal{X}^n$. Thus if $\Gamma = \sum_{i=1}^n \alpha_i \Phi(x_i), \alpha_i \in \mathbb{R}$, then the dot product between translated feature maps can be evaluated in terms of the kernel functions solely. Thus we obtain for our modified feature map

$$\left\langle \tilde{\Phi}(x), \tilde{\Phi}(x') \right\rangle = k(x, x') + \sum_{i=1}^n \alpha_i k(x_i, x) + \sum_{i=1}^n \alpha_i k(x_i, x') + \sum_{i,j=1}^n \alpha_i \alpha_j k(x_i, x_j). \tag{1.29}$$

1.3.1.2 Centering

As a concrete application for a translation operation consider the case that we would like to *center* a set of points in the RKHS. Thus we would like to have a feature map $\tilde{\Phi}$ such that $\frac{1}{n} \sum_{i=1}^n \tilde{\Phi}(x_i) = 0$. Using $\tilde{\Phi}(x) = \Phi(x) + \Gamma$ with $\Gamma = -\sum_{i=1}^n \frac{1}{n} \Phi(x_i)$ this can be obtained immediately utilizing (1.29). The kernel matrix $\tilde{K}$ of the centered feature map $\tilde{\Phi}$ can then be expressed directly in terms of matrix operations by

$$\tilde{K}_{ij} = (K - 1_m K - K 1_m + 1_m K 1_m)_{ij} ,$$

where $1_m \in \mathbb{R}^{m \times m}$ is the constant matrix with all entries equal to $1/m$, and K is the kernel matrix evaluated using Φ.

1.3.1.3 Computing Distances

An essential tool for structured prediction is the problem of computing distances between two objects. For example, to assess the quality of a prediction we would like to measure the distance between predicted object and true object. Since kernel functions can be interpreted as dot products (see (1.3)) they provide an elegant way to measure distances between arbitrary objects. Consider two objects $x_1, x_2 \in \mathcal{X}$,

such as two-word sequences or two automata. Assume we have a kernel function k on such objects; we can use their distance in the RKHS, i.e.,

$$d(x_1, x_2) = ||\Phi(x_1) - \Phi(x_2)||_{\mathcal{H}} = \sqrt{k(x_1, x_1) + k(x_2, x_2) - 2k(x_1, x_2)}.$$

Here, we have utilized the fact that the dot product in $\mathcal{H}$ can be evaluated by kernel functions and thus define the distance between the objects to be the distance between the images of the feature map Φ.

1.3.1.4 Subspace Projections

Another elementary operation which can be performed in a Hilbert space is the one-dimensional orthogonal *projection*. Given two points Ψ, Γ in the RKHS $\mathcal{H}$ we project the point Ψ to the subspace spanned by the point Γ, obtaining

$$\Psi' = \frac{\langle \Gamma, \Psi \rangle}{||\Gamma||^2} \Gamma. \tag{1.30}$$

Considering the case that Ψ and Γ are given by kernel expansions, we see immediately that any dot product with the projected point Ψ' can be expressed with kernel functions only. Using such a projection operation in RKHS, it is straightforward to define a *deflation* procedure:

$$\Psi' = \Psi - \frac{\langle \Gamma, \Psi \rangle}{||\Gamma||^2} \Gamma. \tag{1.31}$$

Using projection and deflation operations, one can perform e.g. the Gram-Schmidt orthogonalization procedure for the construction of orthogonal bases. This was used for example in information retrieval (Cristianini et al., 2001) and computer vision (Wolf and Shashua, 2003). An alternative application of deflation and subspace projection in RKHS was introduced by Rosipal and Trejo (2002) in the context of *subspace regression*.

1.3.2 Kernel Principal Component Analysis

A standard method for feature extraction is the method of principal component analysis (PCA), which aims to identify principal axes in the input. The principal axes are recovered as the eigenvectors of the empirical estimate of the covariance matrix $C_{emp} = \mathbb{E}_{emp}\left[(\mathbf{x} - \mathbb{E}_{emp}[\mathbf{x}])\,(\mathbf{x} - \mathbb{E}_{emp}[\mathbf{x}])^\top\right]$. In contrast to PCA, kernel PCA introduced by Schölkopf et al. (1998) tries to identify principal components of variables which are nonlinearly related to input variables, i.e. principal axis in some feature space $\mathcal{H}$. To this end, given some training set $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of size n, one considers the eigenvectors $\mathbf{v} \in \mathcal{H}$ of the empirical covariance operator in feature space:

covariance in feature space

$$\mathbf{C}_{emp} = \mathbb{E}_{emp}\left[(\Phi(\mathbf{x}) - \mathbb{E}_{emp}[\Phi(\mathbf{x})])\,(\Phi(\mathbf{x}) - \mathbb{E}_{emp}[\Phi(\mathbf{x})])^\top\right].$$

Although this operator and thus its eigenvectors $\mathbf{v}$ cannot be calculated directly, they can be retrieved in terms of kernel evaluations only. To see this, note that even in the case of a high-dimensional feature space $\mathcal{H}$, a finite training set $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of size n when mapped to this feature space spans a subspace $E \subset \mathcal{H}$ whose dimension is at most n. Thus, there are at most n principal axes $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in E^n$ with nonzero eigenvalues. It can be shown that these principal axes can be expressed as linear combinations of the training points $\mathbf{v}_j = \sum_{i=1}^{N} \alpha_i^j \Phi(\mathbf{x}_i), 1 \leq j \leq n$, where the coefficients $\alpha^j \in \mathbb{R}^n$ are obtained as eigenvectors of the kernel matrix evaluated on the training set. If one retains all principal components, kernel PCA can be considered as a basis transform in E, leaving the dot product of training points invariant. To see this, let $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in E^n$ be the principal axes of $\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\}$. The kernel PCA map $\phi_n : \mathcal{X} \to \mathbb{R}^n$ is defined coordinatewise as

$$[\phi_n]_p(\mathbf{x}) = \Phi(\mathbf{x}) \cdot \mathbf{v}_p, \quad 1 \leq p \leq n.$$

Note that by definition, for all i and j, $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$ lie in E and thus

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = \phi_n(\mathbf{x}_i) \cdot \phi_n(\mathbf{x}_j). \tag{1.32}$$

The kernel PCA map is especially useful if one has structured data and one wants to use an algorithm which is not readily expressed in dot products.

1.4 Kernels for Structured Data

We have seen several instances of positive definite kernels, and now intend to describe some kernel functions which are particularly well suited to operate on data domains other than real vector spaces. We start with the simplest data domain: sets.

1.4.1 Set Kernels

Assume that we have given a finite alphabet Σ, i.e. a collection of symbols which we call characters. Furthermore let us denote by $\mathcal{P}(\Sigma)$ the power set of Σ. Then, we define a *set kernel* to be any valid kernel function k which takes two sets $A \in \mathcal{P}(\Sigma)$ and $B \in \mathcal{P}(\Sigma)$ as arguments. As a concrete example, consider the following kernel:

$$k(A, B) = \sum_{x \in A,\ y \in B} 1_{x=y},$$

kernels for text

where $1_{x=y}$ denotes a comparison. This kernel measures the size of the intersection of two sets and is widely used e.g. in text classification where it is referred to as the *sparse vector kernel*. Considering a text document as a set of words, the sparse vector kernel measures the similarity of text document via the number of common

words. Such a kernel was used e.g. in Joachims (1998) for text categorization using SVMs.

The feature map corresponding to the set kernel can be interpreted as a *representation by its parts*. Each singleton $x_i \in \Sigma, 1 \leq i \leq |\Sigma|$, i.e. all sets of cardinality 1, is mapped to the vertex e_i of the unit simplex in $\mathbb{R}^{|\Sigma|}$. Each set A with $|A| > 1$ is then the average of the vertex coordinates, i.e.,

$$\Phi(A) = \sum_{x \in A} \Phi(x) = \sum_{x_i \in \Sigma, x \in A} 1_{x=x_i} e_i.$$

Set kernels are in general very efficient to evaluate as long as the alphabet is finite since the feature map yields a sparse vector in $\mathbb{R}^{|\Sigma|}$. For example, in text classification each dimension corresponds to a specific word, and a component is set to a constant whenever the related word occurs in the text. This is also known as the *bag-of-words* representation. Using an efficient sparse representation, the dot product between two such vectors can be computed quickly.

1.4.2 Rational Kernels

One of the shortcomings of set kernels in applications such as natural language applications is that any relation among the set elements such as, e.g., word order in a document, is completely ignored. However, in many applications one considers data with a more sequential nature such as word sequences in text classification, temporal utterance order in speech recognition, or chains of amino acids in protein analysis. In these cases the data are of sequential nature and can consist of variable-length sequences over some basic alphabet Σ. In the following we review kernels which were introduced to deal with such data types and which belong to the general class of *rational* kernels.

Rational kernels are in principle similarity measures over *sets* of sequences. Since sets of sequences can be compactly represented by automata, rational kernels can be considered as kernels for weighted automata. For a discussion on automata theory see e.g. Hopcroft et al. (2000). In particular, since sequences can be considered as very simple automata, rational kernels automatically implement kernels for sequences. At the heart of a rational kernel is the concept of *weighted transducers* which can be considered as a representation of a binary relation between sequences; see e.g. Mohri et al. (2002) and Cortes et al. (2004).

kernels for automata

Definition 7 (Weighted Transducer) *Given a semiring $K = (\mathbb{K}, \oplus, \otimes)$, a weighted finite-state transducer (WFST) T over $\mathbb{K}$ is given by an* input *alphabet Σ, an output alphabet Ω, a finite set of* states *S, a finite set of* transitions *$E \subseteq S \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times S$, a set of* initial *states $S_0 \in S$, a set of* final *states $S_\infty \subseteq S$, and a weight function $w : S \to \mathbb{K}$.*

In our further discussion we restrict the output alphabet Ω to be equal to the input alphabet, i.e. $\Omega = \Sigma$. We call a sequence of transitions $h = e_1, \ldots, e_n \subset E$ a *path*, where the ith transition is denoted by $\pi_i(h)$. By $\pi_0(h)$ and $\pi_\infty(h)$ we denote starting

and termination states of a path h respectively. Given two sequences $x, y \in \Sigma^*$, we call a path h *successful* if it starts at an initial state, i.e. $\pi_0(h) \in S_0$, terminates in a final state, i.e. $\pi_\infty(h) \in S_\infty$, and concatenating the input and output symbols associated with the traversed transitions equals the sequences x and y. There might be more than a single successful path and we will denote the set of all successful paths depending on the pair (x, y) by $\Pi(x, y)$. Furthermore, for each transition $\pi_i[h] \in E$ we denote by $w(\pi_i[h]) \in \mathbb{K}$ the weight associated with the particular transition $\pi_i[h]$. A transducer is called *regulated* if the weight of any sequence input-output pair $(x, y) \in \Sigma^* \times \Sigma^*$ calculated by

$$[\![T]\!](x, y) := \bigoplus_{h \in \Pi(x,y)} w(\pi_0[h]) \otimes \bigotimes_{i=1}^{|h|} w(\pi_i[h]) \otimes w(\pi_\infty[h]) \tag{1.33}$$

is well-defined and in $\mathbb{K}$.

The interpretation of the weights $w(h)$ and in particular $[\![T]\!](x, y)$ depends on how they are manipulated algebraically and on the underlying semiring $\mathbb{K}$. As a concrete example for the representation of binary relations, let us consider the positive semiring $(\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1}) = (\mathbb{R}_+, +, \times, 0, 1)$ which is also called the probability or real semiring. A binary relation between two sequences $x, y \in \Sigma^*$ is e.g. the conditional probability $[\![T]\!](x, y) = P(y|x)$. Let x_i denote the ith element of the sequence x. We can calculate the conditional probability as

$$P(y|x) = \sum_{h \in \Pi(x,y)} \prod_{i=0} P(y_i|\pi_i[h], x_i) \times P(y_\infty|\pi_\infty(h), x_\infty),$$

where the sum is over all successful paths h and $w(\pi_i[h])) := P(y_i|\pi_i(h), x_i)$ denotes the probability of performing the transition $\pi_i(h)$ and observing (x_i, y_i) as input and outpout symbols. However, reconsidering the example with the *tropical* semiring $(\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1}) = (R \cup \{\infty, -\infty\}, \min, +, +\infty, 0)$ we obtain

$$[\![T]\!](x, y) = \max_{h \in \Pi(x,y)} \sum_{i=0} w(\pi_i[h]) + w(\pi_\infty[h]),$$

which is also known as the Viterbi approximation if the weights are negative log-probabilities, i.e. $w(\pi_\infty[h]) = -\log P(y_i|\pi_i[h], x_i)$. It is also possible to perform algebraic operations on transducers directly. Let T_1, T_2 be two weighted transducers, then a fundamental operation is *composition*.

Definition 8 (Composition) *Given two transducers* $T_1 = \{\Sigma, \Omega, S^1, E^1, S_0^1, S_\infty^1, w^1\}$ *and* $T_2 = \{\Omega, \Delta, S^2, E^2, S_0^2, S_\infty^2, w^2\}$*, the* composition $T_1 \circ T_2$ *is defined as transducer* $R = \{\Sigma, \Delta, S, E, S_0, S_\infty, w\}$ *such that*

$$S = S^1 \times S^2, \quad S_0 = S_0^1 \times S_0^2, \quad S_\infty = S_\infty^1 \times S_\infty^2$$

and each transition $e \in E$ *satisfies*

$$\forall e : (p, p') \overset{a:c/w}{\rightarrow} (q, q') \quad \Rightarrow \quad \exists \{p \overset{a:b/w_1}{\rightarrow} q,\ p' \overset{b:c/w_2}{\rightarrow} q'\},$$

with $w = w_1 \otimes w_2$.

For example, if the transducer T_1 models the conditional probabilities of a label given a feature observation $P(y|\phi(x))$ and another T_2 transducer models the conditional probabilities of a feature given an actual input $P(\phi(x)|x)$, then the transducer obtained by a composition $R = T_1 \circ T_2$ represents $P(y|x)$. In this sense, a composition can be interpreted as a matrix operation for transducers which is apparent if one considers the weights of the composed transducer:

$$[\![T_1 \circ T_2]\!](x, y) = \sum_{z \in \Omega} [\![T_1]\!](x, z)[\![T_2]\!](z, y).$$

Finally, let us introduce the *inverse* transducer T^{-1} that is obtained by swapping all input and output symbols on every transition of a transducer T. We are now ready to introduce the concept of rational kernels.

Definition 9 (Rational Kernel) *A kernel k over the alphabet Σ^* is called* rational *if it can be expressed as weight computation over a transducer T, i.e. $k(x, x') = \Psi([\![T]\!](x, x'))$ for some function $\Psi : \mathbb{K} \to \mathbb{R}$. The kernel is said to be defined by the pair (T, Ψ).*

kernel evaluation by transducers

Unfortunately, not any transducer gives rise to a positive definite kernel. However, from proposition 5(v) and from the definition it follows directly that any transducer $S := T \circ T^{-1}$ is a valid kernel since

$$k(x, y) = \sum_z [\![T]\!](x, z)[\![T]\!](x', z) = [\![S]\!](x, x').$$

The strength of rational kernels is their compact representation by means of transducers. This allows an easy and modular design of novel application-specific similarity measures for sequences. Let us give an example for a rational kernel.

1.4.2.1 n-gram Kernels

An n-gram is a block of n adjacent characters from an alphabet Σ. Hence, the number of distinct n-grams in a text is less than or equal to $|\Sigma|^n$. This shows that the space of all possible n-grams can be very high even for moderate values of n. The basic idea behind the *n-gram kernel* is to compare sequences by means of the subsequences they contain:

$$k(x, x') = \sum_{s \in \Sigma^n} \#(s \in x)\#(s \in x'), \tag{1.34}$$

where $\#(s \in x)$ denotes the number of occurrences of s in x. In this sense, the more subsequences two sequences share, the more similar they are. Vishwanathan and Smola (2004) proved that this class of kernels can be computed in $O(|x| + |x'|)$ time and memory by means of a special suited data structure allowing one to find a compact representation of all subsequences of x in only $O(|x|)$ time and space.

Furthermore, the authors show that the function $f(x) = \langle w, \Phi(x) \rangle$ can be computed in $O(|x|)$ time if preprocessing linear in the size of the expansion w is carried out. Cortes et al. (2004) showed that this kernel can be implemented by a transducer kernel by explicitly constructing a transducer that counts the number of occurrences of n symbol blocks; see e.g figure 1.2. One then can rewrite (1.34) as

$$k(x, x') = [\![T \circ T^{-1}]\!](x, x'). \tag{1.35}$$

In the same manner, one can design transducers that can compute similarities incorporating various costs as, for example, for gaps and mismatches; see Cortes et al. (2004).

1.4.3 Convolution Kernels

One of the first instances of kernel functions on structured data was *convolutional kernels* introduced by Haussler (1999). The key idea is that one may take a structured object and split it up into parts. Suppose that the object $x \in \mathcal{X}$ consists of substructures $x_p \in \mathcal{X}_p$ where $1 \leq p \leq r$ and r denotes the number of overall substructures. Given then the set $\mathcal{P}(\mathcal{X})$ of all possible substructures $\bigotimes_{i=1}^{r} \mathcal{X}_i$, one can define a *relation* R between a subset of $\mathcal{P}$ and the composite object x. As an example consider the relation "part-of" between subsequences and sequences. If there are only a finite number of subsets, the relation R is called finite. Given a finite relation R, let $R^{-1}(x)$ define the set of all possible decompositions of x into its substructures: $R^{-1}(x) = \{z \in \mathcal{P}(\mathcal{X}) : R(z, x)\}$. In this case, Haussler (1999) showed that the so-called R-convolution given as

representation by parts

$$k(x, y) = \sum_{x' \in R^{-1}(x)} \sum_{y' \in R^{-1}(y)} \prod_{i=1}^{r} k_i(x'_i, y'_i) \tag{1.36}$$

is a valid kernel with k_i being a positive definite kernel on $\mathcal{X}_i$.The idea of decomposing a structured object into parts can be applied recursively so that one only requires to construct kernels k_i over the "atomic" parts $\mathcal{X}_i$.

Convolution kernels are very general and were successfully applied in the context of natural language processing (Collins and Duffy, 2002; Lodhi et al., 2000). However, in general the definition of R and in particular R^{-1} for a specific problem is quite difficult.

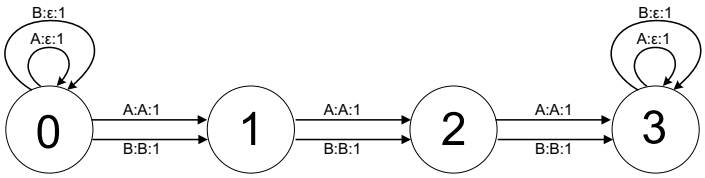

Figure 1.2 A transducer that can be used for calculation of 3-grams for a binary alphabet.

1.4.4 Kernels Based on Local Information

Sometimes it is easier to describe the local neighborhood than to construct a kernel for the overall data structure. Such a neighborhood of a data item might be defined by any item that differs only by the presence or absence of a single property. For example, when considering English words, neighbors of a word can be defined as any other word that would be obtained by misspelling. Given a set of data items, all information about neighbor relations can be represented by e.g. a *neighbor graph*. A vertex in such a neighbor graph would correspond to a data item and two vertices are connected whenever they satisfy some neighbor rule. For example, in the case of English words, a neighbor rule could be that two words are neighbors whenever their edit distance is smaller than some apriori defined threshold. Kondor and Lafferty (2002) utilize such neighbor graphs to construct global similarity measures by using a *diffusion process* analogy. To this end, the authors define a diffusion process by using the so-called *graph Laplacian*, L being a square matrix and where each entry encodes information on how to propagate the information from vertex to vertex. In particular, if A denotes the binary adjacency matrix of the neighbor graph, the graph Laplacian is given by $L = A - D$, where D is a diagonal matrix and each diagonal D_{ii} is the vertex degree of the ith data item. The resulting kernel matrix K is then obtained as the matrix exponential of βL with $\beta < 1$ being a propagation parameter:

similarities due to a diffusion process

$$K = e^{-\beta L} := \lim_{n\to\infty} \left(\mathbf{1} - \frac{\beta}{n} L\right)^n .$$

Such diffusion kernels were successfully applied to such diverse applications as text-categorization, as e.g. in Kandola et al. (2002); gene-function prediction by Vert and Kanehisa (2002); and semisupervised learning, as e.g. in Zhou et al. (2004).

Even if it is possible to define a kernel function for the whole instance space, sometimes it might be advantageous to take into account information from local structure of the data. Recall the Gaussian kernel and polynomial kernels. When applied to an image, it makes no difference whether one uses as x the image or a version of x where all locations of the pixels have been permuted. This indicates that the function space on $\mathcal{X}$ induced by k does not take advantage of the *locality* properties of the data. By taking advantage of the local structure, estimates can be improved. On biological sequences one may assign more weight to the entries of the sequence close to the location where estimates should occur, as was performed e.g. by Zien et al. (2000). In other words, one replaces $\langle x, x' \rangle$ by $x^\top \Omega x$, where $\Omega \succeq 0$ is a diagonal matrix with largest terms at the location which needs to be classified.

In contrast, for images, local interactions between image patches need to be considered. One way is to use the *pyramidal* kernel introduced in Schölkopf (1997) and DeCoste and Schölkopf (2002), which was inspired by the pyramidal cells of the brain: It takes inner products between corresponding image patches, then raises the latter to some power p_1, and finally raises their sum to another power p_2. This

means that mainly short-range interactions are considered and that the long-range interactions are taken with respect to short-range groups.

1.4.5 Tree and Graph Kernels

We now discuss similarity measures on more structured objects such as trees and graphs.

1.4.5.1 Kernels on Trees

For trees Collins and Duffy (2002) propose a decomposition method which maps a tree x into its set of subtrees. The kernel between two trees x, x' is then computed by taking a weighted sum of all terms between both trees and is based on the convolutional kernel (see section 1.4.3). In particular, Collins and Duffy (2002) show an $O(|x| \cdot |x'|)$ algorithm to compute this expression, where $|x|$ is the number of nodes of the tree. When restricting the sum to all proper rooted subtrees it is possible to reduce the time of computation to $O(|x| + |x'|)$ time by means of a tree to sequence conversion (Vishwanathan and Smola, 2004).

1.4.5.2 Kernels on Graphs

A *labeled graph* G is described by a finite set of vertices V, a finite set of edges E, two sets of symbols which we denote by Σ and Ω, and two functions $\mathsf{v} : V \to \Sigma$ and $\mathsf{e} : E \to \Omega$ which assign each vertex and edge a label from the sets Σ, Ω respectively. For directed graphs, the set of edges is a subset of the Cartesian product of the ordered set of vertices with itself, i.e. $E \subseteq V \times V$ such that $(v_i, v_j) \in E$ if and only if vertex v_i is connected to vertex v_j. One might hope that a kernel for a labeled graph can be similarly constructed using some decomposition approach similar to the case of trees. Unfortunately, due to the existence of cycles, graphs cannot be as easily serialized, which prohibits, for example, the use of transducer kernels for graph comparison. A workaround is to artificially construct *walks*, i.e. eventually repetitive sequences of vertex and edge labels. Let us denote by $W(G)$ the set of all possible walks in a graph G of arbitrary length. Then, using an appropriate sequence kernel k_h, a valid kernel for two graphs G_1, G_2 would take the form

graph kernels based on paths

$$k_G(G_1, G_2) = \sum_{h \in W(G_1)} \sum_{h' \in W(G_2)} k_h(h, h'). \tag{1.37}$$

Unfortunately, this kernel can only be evaluated if the graph is acyclic since otherwise the sets $P(G_1), P(G_2)$ are not finite. However, one can restrict the set of all walks $W(G)$ to the set of all *paths* $P(G) \subset W(G)$, i.e. nonrepetitive sequences of vertex and edge labels. Borgwardt and Kriegel (2005) show that computation of this so-called *all-path kernel* is NP-complete. As an alternative, for graphs where each edge is assigned to a cost instead of a general label they propose to further restrict the set of paths. They propose to choose the subset of paths which appear in

path kernels are intractable

shortest-path graph kernel

an all-pairs shortest-path transformed version of the original graph. Thus for each graph G_i which has to be compared, the authors build a new completely connected graph $\hat{G}_i$ of the same size. In contrast to the original graph each edge in $\hat{G}_i$ between nodes v_i and v_j corresponds to the length of the shortest path from v_i to v_j in the original graph G_i. The new kernel function between the transformed graphs is then calculated by comparing all walks of length 1, i.e.,

$$k_{\hat{G}}(G_1, G_2) = \sum_{\substack{h \in W(\hat{G}_1) \\ |h|=1}} \sum_{\substack{h' \in W(\hat{G}_2) \\ |h'|=1}} k_h(h, h'). \tag{1.38}$$

Since algorithms for determining all-pairs shortest paths as, for example, Floyd-Warshall, are of cubic order and comparing all walks of length 1 is of fourth order, the all-pairs shortest-path kernel in (1.38) can be evaluated in $O(|V|^4)$ complexity.

comparing random walks

An alternative approach proposed by Kashima et al. (2003) is to compare two graphs by measuring the similarity of the *probability distributions* of random walks on the two graphs. The authors propose to consider a walk h as a hidden variable and the kernel as a *marginalized kernel* where marginalization is over h, i.e.,

$$k_{RG}(G_1, G_2) = \mathbb{E}[k_G(G_1, G_2)] = \sum_{h \in W(G_1)} \sum_{h' \in W(G_2)} k_h(h, h') p(h|G_1) p(h|G_2), \tag{1.39}$$

where the conditional distributions $p(h|G_1), p(h'|G_2)$ in (1.39) for the random walk h, h' are defined as start, transition, and termination probability distribution over the vertices in V. Note that this marginalized graph kernel can be interpreted as a *randomized version* of (1.37).

By using the dot product of the two probability distributions as kernel, the induced feature space $\mathcal{H}$ is infinite-dimensional, with one dimension for every possible label sequence. Nevertheless, the authors developed an algorithm for how to calculate (1.39) explicitly with $O(|V|^6)$ complexity.

1.4.6 Kernels from Generative Models

In their quest to make density estimates directly accessible to kernel methods Jaakkola and Haussler (1999a,b) designed kernels which work directly on probability density estimates $p(x|\theta)$. Denote by

$$U_\theta(x) := \partial_\theta - \log p(x|\theta) \tag{1.40}$$

$$I := \mathbf{E}_x \left[U_\theta(x) U_\theta^\top(x) \right] \tag{1.41}$$

Fisher information

the Fisher scores and the Fisher information matrix respectively. Note that for maximum likelihood estimators $\mathbf{E}_x [U_\theta(x)] = 0$ and therefore I is the covariance of $U_\theta(x)$. The Fisher kernel is defined as

$$k(x, x') := U_\theta^\top(x) I^{-1} U_\theta(x') \text{ or } k(x, x') := U_\theta^\top(x) U_\theta(x') \tag{1.42}$$

depending on whether we study the normalized or the unnormalized kernel respectively. It is a versatile tool to reengineer existing density estimators for the purpose of discriminative estimation.

In addition to that, it has several attractive theoretical properties: Oliver et al. (2000) show that estimation using the normalized Fisher kernel corresponds to an estimation subject to a regularization on the $L_2(p(\cdot|\theta))$ norm.

Moreover, in the context of exponential families (see section 3.6 for a more detailed discussion) where $p(x|\theta) = \exp(\langle \phi(x), \theta \rangle - g(\theta))$, we have

$$k(x, x') = [\phi(x) - \partial_\theta g(\theta)] [\phi(x') - \partial_\theta g(\theta)] \tag{1.43}$$

for the unnormalized Fisher kernel. This means that up to centering by $\partial_\theta g(\theta)$ the Fisher kernel is identical to the kernel arising from the inner product of the sufficient statistics $\phi(x)$. This is not a coincidence and is often encountered when working with nonparametric exponential families. A short description of exponential families is given further below in section 3.6. Moreover, note that the centering is immaterial, as can be seen in lemma 13.

1.5 An Example of a Structured Prediction Algorithm Using Kernels

In this section we introduce concepts for structured prediction based on kernel functions. The basic idea is based on the property that kernel methods embed any data type into a linear space and thus can be used to transform the targets to a new representation more amenable to prediction using existing technqiues. However, since one is interested in predictions of the original type one has to solve an additional *reconstruction* problem that is independent of the learning problem and therefore might be solved more easily. The first algorithm following this recipe was kernel dependency estimation (KDE) introduced by Weston et al. (2002) and which we discuss next.

kernel dependency estimation

Given n pairs of data items $D_n = \{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$ one is interested in learning a mapping $t_Z : \mathcal{X} \to \mathcal{Y}$. As a first step in KDE one constructs a linear embedding of the targets only. For example, Weston et al. (2002) propose kernel PCA using a kernel function on $\mathcal{Y}$, i.e. $k_y(y_1, y_2) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Note that this kernel function gives rise to a feature map ϕ_y into a RKHS $\mathcal{H}_y$ and allows application of the kernel PCA map (see section 1.3.2). The new vectorial representation of the outputs can then be used to learn a map $T_{\mathcal{H}}$ from the input space $\mathcal{X}$ to the vectorial representation of the outputs, i.e. $\mathbb{R}^n$. This new learning problem using the transformed output is a standard multivariate regression problem and was solved for example in Weston et al. (2002) with kernel ridge regression using a kernel for $\mathcal{X}$.

kernel for the outputs

Finally, for a given new input point x^* and its predicted representation $T_{\mathcal{H}}(x^*)$, one has to *reconstruct* the output element $y^* \in \mathcal{Y}$ that matches the predicted representation best, i.e.

$$y^* = \arg\min_{y \in \mathcal{Y}} \; ||\phi_y(y) - T_{\mathcal{H}}(x^*)||^2_{\mathcal{H}_y}. \tag{1.44}$$

pre-image/decoding problem

The problem (1.44) is known as the *pre-image* problem or alternatively as the *decoding* problem and has wide applications in kernel methods. , We summarize all feature maps used in KDE in figure 1.3 where we denote by $\Gamma : \mathcal{H}_y \to \mathcal{Y}$ the pre-image map which is given by (1.44). In chapter 8, we see an application of KDE to the task of string prediction where the authors design a pre-image map based on n-gram kernels.

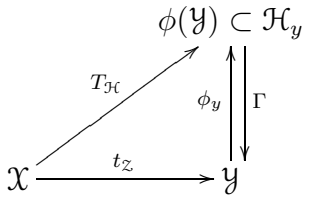

Figure 1.3 Mappings between original sets $\mathcal{X}, \mathcal{Y}$ and corresponding feature spaces $\mathcal{H}_y$ in kernel dependency estimation.

1.6 Conclusion

Kernels can be used for decorrelation of nontrivial structures between points in Euclidean space. Furthermore, they can be used to embed complex data types into linear spaces leading straightforward to distance and similarity measures among instances of arbitrary type. Finally, kernel functions *encapsulate* the data from the algorithm and thus allow use of the same algorithm on different data types without changing the implementation. Thus, whenever a learning algorithm can be expressed in kernels it can be utilized for arbitrary data types by exchanging the kernel function. This reduces the effort of using existing inference algorithms for novel application fields to introducing a novel specifically designed kernel function.