9

Basic kernels and kernel types

There are two key properties that are required of a kernel function for an application. Firstly, it should capture the measure of similarity appropriate to the particular task and domain, and secondly, its evaluation should require significantly less computation than would be needed in an explicit evaluation of the corresponding feature mapping ϕ. Both of these issues will be addressed in the next four chapters but the current chapter begins the consideration of the efficiency question.

A number of computational methods can be deployed in order to shortcut the computation: some involve using closed-form analytic expressions, others exploit recursive relations, and others are based on sampling. This chapter aims to show several different methods in action, with the aim of illustrating how to design new kernels for specific applications. It will also pave the way for the final three chapters that carry these techniques into the design of advanced kernels.

We will also return to an important theme already broached in Chapter 3, namely that kernel functions are not restricted to vectorial inputs: kernels can be designed for objects and structures as diverse as strings, graphs, text documents, sets and graph-nodes. Given the different evaluation methods and the diversity of the types of data on which kernels can be defined, together with the methods for composing and manipulating kernels outlined in Chapter 3, it should be clear how versatile this approach to data modelling can be, allowing as it does for refined customisations of the embedding map ϕ to the problem at hand.

9.1 Kernels in closed form

We have already seen polynomial and Gaussian kernels in Chapters 2 and 3. We start by revisiting these important examples, presenting them in a different light in order to illustrate a series of important design principles that will lead us to more sophisticated kernels. The polynomial kernels in particular will serve as a thread linking this section with those that follow.

Polynomial kernels In Chapter 3, Proposition 3.24 showed that the space of valid kernels is closed under the application of polynomials with positive coefficients. We now give a formal definition of the polynomial kernel.

Definition 9.1 [Polynomial kernel] The *derived polynomial kernel* for a kernel κ_1 is defined as

$$\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z})),$$

where $p(\cdot)$ is any polynomial with positive coefficients. Frequently, it also refers to the special case

$$\kappa_d(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + R)^d,$$

defined over a vector space X of dimension n, where R and d are parameters. ■

Expanding the polynomial kernel κ_d using the binomial theorem we have

$$\kappa_d(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} \binom{d}{s} R^{d-s} \langle \mathbf{x}, \mathbf{z} \rangle^s. \tag{9.1}$$

Our discussion in Chapter 3 showed that the features for each component in the sum together form the features of the whole kernel. Hence, we have a reweighting of the features of the polynomial kernels

$$\hat{\kappa}_s(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^s, \text{ for } s = 0, \ldots, d.$$

Recall from Chapter 3 that the feature space corresponding to the kernel $\hat{\kappa}_s(\mathbf{x}, \mathbf{z})$ has dimensions indexed by all monomials of degree s, for which we use the notation

$$\boldsymbol{\phi}_{\mathbf{i}}(\mathbf{x}) = \mathbf{x}^{\mathbf{i}} = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n},$$

where $\mathbf{i} = (i_1, \ldots, i_n) \in \mathbb{N}^n$ satisfies

$$\sum_{j=1}^{n} i_j = s.$$

The features corresponding to the kernel $\kappa_d(\mathbf{x}, \mathbf{z})$ are therefore all functions of the form $\boldsymbol{\phi}_{\mathbf{i}}(\mathbf{x})$ for $\mathbf{i}$ satisfying

$$\sum_{j=1}^{n} i_j \leq d.$$

Proposition 9.2 *The dimension of the feature space for the polynomial kernel* $\kappa_d(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + R)^d$ *is*

$$\binom{n+d}{d}.$$

Proof We will prove the result by induction over n. For $n = 1$, the number is correctly computed as $d + 1$. Now consider the general case and divide the monomials into those that contain at least one factor x_1 and those that have $i_1 = 0$. Using the induction hypothesis there are $\binom{n+d-1}{d-1}$ of the first type of monomial, since there is a 1-1 correspondence between monomials of degree at most d with one factor x_1 and monomials of degree at most $d-1$ involving all base features. The number of monomials of degree at most d satisfying $i_1 = 0$ is on the other hand equal to $\binom{n-1+d}{d}$ since this corresponds to a restriction to one fewer input feature. Hence, the total number of all monomials of degree at most d is equal to

$$\binom{n+d-1}{d-1} + \binom{n-1+d}{d} = \binom{n+d}{d},$$

as required. □

Remark 9.3 [Relative weightings] Note that the parameter R allows some control of the relative weightings of the different degree monomials, since by equation (9.1), we can write

$$\kappa_d(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} a_s \hat{\kappa}_s(\mathbf{x}, \mathbf{z}),$$

where

$$a_s = \binom{d}{s} R^{d-s}.$$

Hence, increasing R decreases the relative weighting of the higher order polynomials. ■

Remark 9.4 [On computational complexity] One of the reasons why it is possible to reduce the evaluation of polynomial kernels to a very simple computation is that we are using *all* of the monomials. This has the effect of reducing the freedom to control the weightings of individual monomials. Paradoxically, it can be much more expensive to use only a subset of them, since, if no pattern exists that can be exploited to speed the overall computation, we are reduced to enumerating each monomial feature in turn. For some special cases, however, recursive procedures can be used as a shortcut as we now illustrate. ■

All-subsets kernel As an example of a different combination of features consider a space with a feature $\boldsymbol{\phi}_A$ for each subset $A \subseteq \{1, 2, \ldots, n\}$ of the input features, including the empty subset. Equivalently we can represent them as we did for the polynomial kernel as features

$$\boldsymbol{\phi}_{\mathbf{i}}(\mathbf{x}) = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n},$$

with the restriction that $\mathbf{i} = (i_1, \ldots, i_n) \in \{0, 1\}^n$. The feature $\boldsymbol{\phi}_A$ is given by multiplying together the input features for all the elements of the subset

$$\boldsymbol{\phi}_A(\mathbf{x}) = \prod_{i \in A} x_i.$$

This generates all monomial features for all combinations of up to n different indices but here, unlike in the polynomial case, each factor in the monomial has degree 1.

Definition 9.5 [All-subsets embedding] The *all-subsets kernel* is defined by the embedding

$$\boldsymbol{\phi} : \mathbf{x} \longmapsto (\boldsymbol{\phi}_A(\mathbf{x}))_{A \subseteq \{1,\ldots,n\}},$$

with the corresponding kernel $\kappa_{\subseteq}(\mathbf{x}, \mathbf{z})$ given by

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle.$$

■

There is a simple computation that evaluates the all-subsets kernel as the following derivation shows

$$\begin{aligned} \kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) &= \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle = \sum_{A \subseteq \{1,\ldots,n\}} \boldsymbol{\phi}_A(\mathbf{x}) \boldsymbol{\phi}_A(\mathbf{z}) = \sum_{A \subseteq \{1,\ldots,n\}} \prod_{i \in A} x_i z_i \\ &= \prod_{i=1}^{n} (1 + x_i z_i), \end{aligned}$$

where the last step follows from an application of the distributive law. We summarise this in the following computation.

Computation 9.6 [All-subsets kernel] The all-subsets kernel is computed by

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{n} (1 + x_i z_i)$$

for n-dimensional input vectors. ■

Note that each subset in this feature space is assigned equal weight unlike the variable weightings characteristic of the polynomial kernel. We will see below that the same kernel can be formulated in a recursive way, giving rise to a class known as the ANOVA kernels. They build on a theme already apparent in the above where we see the kernel expressed as a sum of products, but computed as a product of sums.

Remark 9.7 [Recursive computation] We can clearly compute the polynomial kernel of degree d recursively in terms of lower degree kernels using the recursion

$$\kappa_d(\mathbf{x}, \mathbf{z}) = \kappa_{d-1}(\mathbf{x}, \mathbf{z}) (\langle \mathbf{x}, \mathbf{z} \rangle + R).$$

Interestingly it is also possible to derive a recursion in terms of the input dimensionality n. This recursion follows the spirit of the inductive proof for the dimension of the feature space given for polynomial kernels. We use the notation

$$\kappa_s^m(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}_{1:m}, \mathbf{z}_{1:m} \rangle + R)^s,$$

where $\mathbf{x}_{1:m}$ denotes the restriction of $\mathbf{x}$ to its first m features. Clearly we can compute

$$\kappa_s^0(\mathbf{x}, \mathbf{z}) = R^s \text{ and } \kappa_0^m(\mathbf{x}, \mathbf{z}) = 1.$$

For general m and s, we divide the products in the expansion of

$$(\langle \mathbf{x}_{1:m}, \mathbf{z}_{1:m} \rangle + R)^s$$

into those that contain at least one factor $x_m z_m$ and those that contain no such factor. The sum of the first group is equal to $s\kappa_{s-1}^m(\mathbf{x}, \mathbf{z}) x_m z_m$ since there is a choice of s factors in which the $x_m z_m$ arises, while the remaining factors are drawn in any way from the other $s-1$ components in the overall

product. The sum over the second group equals $\kappa_s^{m-1}(\mathbf{x}, \mathbf{z})$, resulting in the recursion

$$\kappa_s^m(\mathbf{x}, \mathbf{z}) = s\kappa_{s-1}^m(\mathbf{x}, \mathbf{z}) x_m z_m + \kappa_s^{m-1}(\mathbf{x}, \mathbf{z}).$$

Clearly, this is much less efficient than the direct computation of the definition, since we must compute $O(nd)$ intermediate values giving a complexity of $O(nd)$, even if we save all the intermediate values, as compared to $O(n+d)$ for the direct method. The approach does, however, motivate the use of recursion introduced below for the computation of kernels for which a direct route does not exist. ■

Gaussian kernels Gaussian kernels are the most widely used kernels and have been extensively studied in neighbouring fields. Proposition 3.24 of Chapter 3 verified that the following kernel is indeed valid.

Definition 9.8 [Gaussian kernel] For $\sigma > 0$, the *Gaussian kernel* is defined by

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right).$$

■

For the Gaussian kernel the images of all points have norm 1 in the resulting feature space as $\kappa(\mathbf{x}, \mathbf{x}) = \exp(0) = 1$. The feature space can be chosen so that the images all lie in a single orthant, since all inner products between mapped points are positive.

Note that we are not restricted to using the Euclidean distance in the input space. If for example $\kappa_1(\mathbf{x}, \mathbf{z})$ is a kernel corresponding to a feature mapping $\boldsymbol{\phi}_1$ into a feature space F_1, we can create a Gaussian kernel in F_1 by observing that

$$\|\boldsymbol{\phi}_1(\mathbf{x}) - \boldsymbol{\phi}_1(\mathbf{z})\|^2 = \kappa_1(\mathbf{x}, \mathbf{x}) - 2\kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_1(\mathbf{z}, \mathbf{z}),$$

giving the derived Gaussian kernel as

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\kappa_1(\mathbf{x}, \mathbf{x}) - 2\kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_1(\mathbf{z}, \mathbf{z})}{2\sigma^2}\right).$$

The parameter σ controls the flexibility of the kernel in a similar way to the degree d in the polynomial kernel. Small values of σ correspond to large values of d since, for example, they allow classifiers to fit any labels, hence risking overfitting. In such cases the kernel matrix becomes close to the identity matrix. On the other hand, large values of σ gradually reduce the

kernel to a constant function, making it impossible to learn any non-trivial classifier. The feature space has infinite-dimension for every value of σ but for large values the weight decays very fast on the higher-order features. In other words although the rank of the kernel matrix will be full, for all practical purposes the points lie in a low-dimensional subspace of the feature space.

Remark 9.9 [Visualising the Gaussian feature space] It can be hard to form a picture of the feature space corresponding to the Gaussian kernel. As described in Chapter 3 another way to represent the elements of the feature space is as functions in a Hilbert space

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

with the inner product between functions given by

$$\left\langle \sum_{i=1}^{\ell} \alpha_i \kappa(\mathbf{x}_i, \cdot), \sum_{j=1}^{\ell} \beta_j \kappa(\mathbf{x}_j, \cdot) \right\rangle = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \beta_j \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

To a first approximation we can think of each point as representing a new potentially orthogonal direction, but with the overlap to other directions being bigger the closer the two points are in the input space. ■

9.2 ANOVA kernels

The polynomial kernel and the all-subsets kernel have limited control of what features they use and how they weight them. The polynomial kernel can only use all monomials of degree d or of degree up to d with a weighting scheme depending on just one parameter R. As its name suggests, the all-subsets kernel is restricted to using all the monomials corresponding to possible subsets of the n input space features. We now present a method that allows more freedom in specifying the set of monomials.

The ANOVA kernel κ_d of degree d is like the all-subsets kernel except that it is restricted to subsets of the given cardinality d. We can use the above notation $\mathbf{x}^{\mathbf{i}}$ to denote the expression $x_1^{i_1} x_2^{i_2} \dots x_n^{i_n}$, where $\mathbf{i} = (i_1, \dots, i_n) \in \{0,1\}^n$ with the further restriction that

$$\sum_{j=1}^{n} i_j = d.$$

For the case $d = 0$ there is one feature with constant value 1 corresponding

to the empty set. The difference between the ANOVA and polynomial kernel $\kappa_d(\mathbf{x}, \mathbf{z})$ is the exclusion of repeated coordinates.

ANOVA stands for ANalysis Of VAriance, the first application of Hoeffding's decompositions of functions that led to this kernel (for more about the history of the method see Section 9.10). We now give the formal definition.

Definition 9.10 [ANOVA embedding] The embedding of the ANOVA kernel of degree d is given by

$$\boldsymbol{\phi}_d : \mathbf{x} \longmapsto (\boldsymbol{\phi}_A(\mathbf{x}))_{|A|=d},$$

where for each subset A the feature is given by

$$\boldsymbol{\phi}_A(\mathbf{x}) = \prod_{i \in A} x_i = \mathbf{x}^{\mathbf{i}_A},$$

where $\mathbf{i}_A$ is the indicator function of the set A. ∎

The dimension of the resulting embedding is clearly $\binom{n}{d}$, since this is the number of such subsets, while the resulting inner product is given by

$$\begin{aligned}
\kappa_d(\mathbf{x}, \mathbf{z}) &= \langle \boldsymbol{\phi}_d(\mathbf{x}), \boldsymbol{\phi}_d(\mathbf{z}) \rangle = \sum_{|A|=d} \boldsymbol{\phi}_A(\mathbf{x}) \boldsymbol{\phi}_A(\mathbf{z}) \\
&= \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} (x_{i_1} z_{i_1})(x_{i_2} z_{i_2}) \ldots (x_{i_d} z_{i_d}) \\
&= \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} \prod_{j=1}^{d} x_{i_j} z_{i_j}.
\end{aligned}$$

Note again the sums of products in the definition of the kernel.

Remark 9.11 [Feature space] Notice that this is a proper subspace of the embedding space generated by a polynomial kernel of degree d, since instead of imposing $1 \le i_1 < i_2 < \cdots < i_d \le n$, the polynomial kernel only requires the weaker restriction $1 \le i_1 \le i_2 \le \cdots \le i_d \le n$. The weighting of the subspace of features considered for the ANOVA kernel is also uniform in the polynomial kernel, but the weightings of the features with repeated indices is higher. ∎

We have been keen to stress that it should be possible to evaluate a kernel faster than by an explicit computation of the feature vectors. Here, for an explicit computation, the number of operations grows as $d\binom{n}{d}$ since there are $\binom{n}{d}$ features each of which requires $O(d)$ operations to evaluate. We now consider a recursive method of computation following the spirit of the recursive evaluation of the polynomial kernel given in Remark 9.7.

Naive recursion We again introduce a series of intermediate kernels. Again using the notation $\mathbf{x}_{1:m} = (x_1, \ldots, x_m)$ we introduce, for $m \geq 1$ and $s \geq 0$

$$\kappa_s^m(\mathbf{x}, \mathbf{z}) = \kappa_s(\mathbf{x}_{1:m}, \mathbf{z}_{1:m}),$$

which is the ANOVA kernel of degree s applied to the inputs restricted to the first m coordinates. In order to evaluate $\kappa_s^m(\mathbf{x}, \mathbf{z})$ we now argue inductively that its features can be divided into two groups: those that contain x_m and the remainder. There is a 1-1 correspondence between the first group and subsets of size $d-1$ restricted to $\mathbf{x}_{1:m-1}$, while the second group are subsets of size d restricted to $\mathbf{x}_{1:m-1}$. It follows that

$$\kappa_s^m(\mathbf{x}, \mathbf{z}) = (x_m z_m)\, \kappa_{s-1}^{m-1}(\mathbf{x}, \mathbf{z}) + \kappa_s^{m-1}(\mathbf{x}, \mathbf{z}). \tag{9.2}$$

The base of the recursion occurs when $m < s$ or $s = 0$. Clearly, we have

$$\kappa_s^m(\mathbf{x}, \mathbf{z}) = 0, \text{ if } m < s, \tag{9.3}$$

since no subset of size s can be found, while

$$\kappa_0^m(\mathbf{x}, \mathbf{z}) = 1, \tag{9.4}$$

as the empty set has a feature value of 1. We summarise this in the following computation.

Computation 9.12 [ANOVA recursion] The ANOVA kernel is given by the following recursion

$$\begin{aligned}
\kappa_0^m(\mathbf{x}, \mathbf{z}) &= 1, \text{ if } m \geq 0, \\
\kappa_s^m(\mathbf{x}, \mathbf{z}) &= 0, \text{ if } m < s, \\
\kappa_s^m(\mathbf{x}, \mathbf{z}) &= (x_m z_m)\, \kappa_{s-1}^{m-1}(\mathbf{x}, \mathbf{z}) + \kappa_s^{m-1}(\mathbf{x}, \mathbf{z})
\end{aligned}$$

■

The computational framework presented here forms the basis for several further generalisations that will be discussed in this and subsequent chapters leading to the introduction of string kernels in Chapter 11.

Remark 9.13 [Cost of naive recursion] As we observed above, a direct computation of the kernel by expicit evaluation of the features would involve $O\left(d\binom{n}{d}\right)$ operations. If we were to implement the recursion given in equations (9.2) to (9.4), this would also remain very costly. If we denote with a function $T(m, s)$ the cost of performing the recursive calculation of

$\kappa_s^m(\mathbf{x}, \mathbf{z})$, we can use the recurrence relation to deduce the following estimate for the number of operations:

$$\begin{aligned} T(m,s) &= T(m-1,s) + T(m-1,s-1) + 3 \\ &> T(m-1,s) + T(m-1,s-1), \end{aligned}$$

with $T(m,s) = 1$ for $m < s$ and $T(m,0) = 1$. For these base values we have the inequality

$$T(m,s) \geq \binom{m}{s},$$

while using an inductive assumption gives

$$\begin{aligned} T(m,s) &> T(m-1,s) + T(m-1,s-1) \\ &\geq \binom{m-1}{s} + \binom{m-1}{s-1} = \binom{m}{s}. \end{aligned}$$

Hence the cost of computing the kernel $\kappa_d(\mathbf{x}, \mathbf{z})$ is at least $\binom{n}{d}$ which is still exponential in the degree d. ∎

Dynamic programming evaluation The naive recursion is inefficient because it repeats many of the same computations again and again. The key to a drastic reduction in the overall computational complexity is to save the values of $\kappa_s^m(\mathbf{x}, \mathbf{z})$ in a dynamic programming table indexed by s and m as they are computed:

DP	$m=1$	2	$\cdots$	n
$s=0$	1	1	$\cdots$	1
1	$x_1 z_1$	$x_1 z_1 + x_2 z_2$	$\cdots$	$\sum_{i=1}^n x_i z_i$
2	0	$\kappa_2^2(\mathbf{x}, \mathbf{z})$	$\cdots$	$\kappa_2^n(\mathbf{x}, \mathbf{z})$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
d	0	0	$\cdots$	$\kappa_d^n(\mathbf{x}, \mathbf{z})$

(9.5)

If we begin computation with the first row from left to right and continue down the table taking each row in turn, the evaluation of a particular entry depends on the entry diagonally above to its left and the entry immediately to its left. Hence, both values will already be available in the table. The required kernel evaluation is the bottom rightmost entry in the table. We summarise in the following algorithm.

Algorithm 9.14 [ANOVA kernel] The ANOVA kernel κ_d is computed in Code Fragment 9.1. ∎

Input	vectors $\mathbf{x}$ and $\mathbf{z}$ of length n, degree d
Process	for $j = 0 : n$
2	$\quad \mathrm{DP}(0, j) = 1;$
3	end
6	for $k = 1 : d$
7	$\quad \mathrm{DP}(k, k-1) = 0;$
9	$\quad$ for $j = k : n$
10	$\quad\quad \mathrm{DP}(k, j) = \mathrm{DP}(k, j-1) + x(j)\, z(j)\, \mathrm{DP}(k-1, j-1);$
11	$\quad$ end
12	end
Output	kernel evaluation $\kappa_d(\mathbf{x}, \mathbf{z}) = \mathrm{DP}(d, n)$

Code Fragment 9.1. Pseudocode for ANOVA kernel.

Cost of the computation The computation involves

$$3\left(nd - \frac{d(d-1)}{2}\right)$$

numerical operations. If we take into account that entries $\kappa_s^m(\mathbf{x}, \mathbf{z})$ with $m > n - d + s$ cannot affect the result, this can be reduced still further.

Remark 9.15 [Dynamic programming notation] Such a staged recursive implementation is often referred to as a *dynamic programming*, though dynamic programs can involve more complex evaluations at each stage and can be run over more general structures such as graphs rather than simple tables. We will use a uniform notation for showing dynamic programming tables following the example of (9.5), with DP in the upper left corner indicating this type of table. See Appendix B for details of the different table types. ■

Remark 9.16 [On using different degrees] Observe that the final column in the table contains the ANOVA kernel evaluations for all degrees up to and including d. It is therefore possible, at very little extra cost, to evaluate the kernel

$$\kappa_{\leq d}(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} \kappa_s(\mathbf{x}, \mathbf{z}),$$

or indeed any appropriate reweighting of the component kernels. If we take this to the extreme value of $d = n$, we recover the all-subsets kernel

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \kappa_{\leq n}(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{n} \kappa_s(\mathbf{x}, \mathbf{z}),$$

though the method of computation described here is a lot less efficient than that given in Remark 9.1

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{n} (1 + x_i z_i),$$

which provides a simpler recursion of complexity only $O(3n)$. Viewed recursively we can see

$$\kappa_{\subseteq}(\mathbf{x}_m, \mathbf{z}_m) = \kappa_{\subseteq}(\mathbf{x}_{m-1}, \mathbf{z}_{m-1})(1 + x_m z_m),$$

corresponding to dividing all subsets into those that contain m and those that do not. ■

Remark 9.17 [Uneven term weighting] In both the ANOVA kernel and the all-subsets kernel, we have the freedom to downplay some features and emphasise others by simply introducing a weighting factor $a_i \geq 0$ whenever we evaluate

$$a_i x_i z_i,$$

so that, for example, the all-subsets kernel becomes

$$\kappa_{\subseteq}(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{n} (1 + a_i x_i z_i).$$

In the case of the ANOVA evaluation it is also possible to vary weightings of the features depending on the structure of the monomial. For example, monomials with non-consecutive indices can have their weights reduced. This will form a key ingredient in the string kernels. ■

Computation 9.18 [Alternative recursion for the ANOVA kernel] Other recursive computations are possible for the ANOVA kernel. When originally introduced, the recursion given was as follows

$$\begin{aligned} \kappa_0(\mathbf{x}, \mathbf{z}) &= 1 \\ \kappa_d(\mathbf{x}, \mathbf{z}) &= \frac{1}{d} \sum_{s=1}^{d} (-1)^{s+1} \kappa_{d-s}(\mathbf{x}, \mathbf{z}) \bar{\kappa}_s(\mathbf{x}, \mathbf{z}) \end{aligned}$$

where $\bar{\kappa}_s(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} (x_i z_i)^s$. ■

Remark 9.19 [Correctness of alternative recursion] The correctness of this alternative can be verified formally. Intuitively, since $\bar{\kappa}_s(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$, the first term in the sum is

$$\frac{1}{d} \kappa_{d-1}(\mathbf{x}, \mathbf{z}) \langle \mathbf{x}, \mathbf{z} \rangle,$$

which includes the d subsets as features with weighting 1, but also includes features $\boldsymbol{\phi}_{\mathbf{i}}(\mathbf{x})$ for which some components have repeated entries. The second term removes these but introduces others which are in turn removed by the following term and so on. The complexity of this method using tables is $O\left(nd + d^2\right)$, which is comparable to Algorithm 9.14. However the version discussed above is more useful for extensions to the case of strings and is numerically more stable. ■

Remark 9.20 [Extensions to general kernels on structures] We can view the individual components $x_i z_i$ in the ANOVA kernel as a base kernel

$$\kappa_i(\mathbf{x}, \mathbf{z}) = x_i z_i.$$

Indeed the reweighting scheme discussed in Remark 9.17 can be seen as changing this base kernel to

$$\kappa_i(\mathbf{x}, \mathbf{z}) = a_i x_i z_i.$$

Taking this view suggests that we could compare data types more general than real numbers in each coordinate, be they discrete objects such as characters, or multi-coordinate vectors. More generally we can simply replace the n coordinates by n general base kernels that provide different methods of comparing the two objects. We do not require that the ith kernel depends only on the ith coordinate. It might depend on a set of coordinates that could overlap with the coordinates affecting other features. In general all of the base kernels could depend on all of the input coordinates. Given a sequence $\kappa_1(\mathbf{x}, \mathbf{z}), \ldots, \kappa_n(\mathbf{x}, \mathbf{z})$ of base kernels, this more general version of the ANOVA kernel therefore has the following form

$$\kappa_d^A(\mathbf{x}, \mathbf{z}) = \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} \prod_{j=1}^{d} \kappa_{i_j}(\mathbf{x}, \mathbf{z}).$$

There are many different applications where this might prove useful. For example it allows us to consider the case when the inputs are discrete structures with d components, with $\kappa_i(\mathbf{x}, \mathbf{z})$ providing a comparison of the ith

subcomponent. Both recursions can still be used. For example the original recursion becomes

$$\kappa_s^m(\mathbf{x},\mathbf{z}) = \kappa_m(\mathbf{x},\mathbf{z})\,\kappa_{s-1}^{m-1}(\mathbf{x},\mathbf{z}) + \kappa_s^{m-1}(\mathbf{x},\mathbf{z}),$$

while for the second recursion we must replace the evaluation of $\bar{\kappa}_s(\mathbf{x},\mathbf{z})$ by

$$\bar{\kappa}_s(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{n} \kappa_i(\mathbf{x},\mathbf{z})^s.$$

■

9.3 Kernels from graphs

The ANOVA kernel is a method that allows us to be slightly more selective about which monomials are included. Clearly it is, however, very far from general. Ideally we would like to extend the range of selective options available while still retaining the efficient computation it affords. We will achieve this by first showing how the all-subsets kernel computation can be represented using a graph. This will suggest a more general family of methods that subsumes all three examples considered so far.

As a first example consider the all-subsets kernel. We can illustrate the computation

$$\kappa_{\subseteq}(\mathbf{x},\mathbf{z}) = \prod_{i=1}^{n}(1 + x_i z_i)$$

with the graph shown in Figure 9.1. Here the computation flows from left

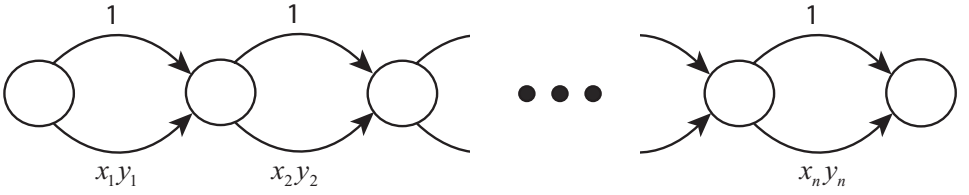

Fig. 9.1. Kernel graph for the all-subsets kernel.

to right with the value at a node given by summing over the edges reaching that node, the value at the previous node times the value on the edge. The double edges enable us to code the fact that the previous value is multiplied by the factor

$$(1 + x_i z_i).$$

We can view the graph as an illustration of the computational strategy, but it also encodes the features of the kernel as the set of directed paths from the leftmost to the rightmost vertex. Each such path must go through each vertex using either the upper or lower edge. Using the lower edge corresponds to including the feature while using the upper edge means that it is left out. It follows that there are 2^n paths in 1-1 correspondence with the subsets of $\{1, \ldots, n\}$, that is with the coordinates of the kernel feature mapping.

The perspective afforded by illustrating the kernel computation as a graph suggests defining a more general set of kernels in terms of a general graph structure.

Consider a directed (edges have an associated direction) graph $G = (V, E)$ where V is a set of vertices with two labelled vertices: s the source vertex and t the sink vertex. The set E of directed edges without loops is labelled with the base kernels. For the all-subsets kernel these were either a constant or $x_i z_i$ for some i, but in general we could consider the base kernels $\kappa_i(\mathbf{x}, \mathbf{z})$ in the spirit of Remark 9.20. If we treat a constant function as a kernel we can simply view the edges $e \in E$ as indexing a kernel $\kappa_e(\mathbf{x}, \mathbf{z})$. Note that we will denote path p by the sequence of vertices it contains, for example

$$p = (u_0 u_1 \ldots u_d),$$

where it is understood that $(u_i \to u_{i+1}) \in E$ for $i = 1, \ldots, d-1$. If $(u_d \to v) \in E$ then pv denotes the path

$$(u_0 u_1 \ldots u_d v).$$

We summarise in the following definition.

Definition 9.21 A *kernel graph* is a directed graph $G = (V, E)$ with a source vertex s of in-degree 0 and sink vertex t of out-degree 0. Furthermore, each edge e is labelled with a kernel κ_e. Let P_{st} be the set of directed paths from vertex s to vertex t and for a path $p = (u_0 u_1 \cdots u_d)$ let

$$\kappa_p(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{d} \kappa_{(u_{i-1} \to u_i)}(\mathbf{x}, \mathbf{z}),$$

that is, the product of the kernels associated with the edges of p. We now introduce the *graph kernel* specified by the graph G by

$$\kappa_G(\mathbf{x}, \mathbf{z}) = \sum_{p \in P_{st}} \kappa_p(\mathbf{x}, \mathbf{z}) = \sum_{p \in P_{st}} \prod_{i=1}^{d} \kappa_{(u_{i-1} \to u_i)}(\mathbf{x}, \mathbf{z}).$$

For simplicity we assume that no vertex has a directed edge to itself. If the graph has no directed loops we call the graph *simple*, while in general we allow directed loops involving more than one edge. ■

A graph kernel can be seen to be a kernel since sums and products of kernels are kernels though for graphs with loops there will be an issue of convergence of the infinite sums involved. Notice that if the base kernels are the simple products $x_i z_i$ or the constant 1, then we have each path corresponding to a monomial feature formed as the product of the factors on that path.

Remark 9.22 [Vertex labelled versus edge labelled graphs] We have introduced kernel graphs by labelling the edges with base kernels. It is also possible to label the vertices of the graph and again define the kernel as the sum over all paths with the features for a path being the product of the features associated with nodes on the path. We omit these details as they do not add any additional insights. ■

Remark 9.23 [Flexibility of graph kernels] Clearly the range of options available using different graphs G is very large. By adjusting the structure of the graph we can control which monomial features are included. We have even allowed the possibility of graphs with loops that would give rise to infinite feature spaces including monomials of arbitrary length. We have been able to significantly broaden the definition of the kernels, but can we retain the efficient computational methods characteristic of the all-subsets kernel? In particular if the graph is directed does the computation dictated by the directed graph really compute the kernel? ■

Computation of graph kernels Given a kernel graph $G = (V, E)$, consider introducing a kernel for each vertex $u \in V$ defined by

$$\kappa_u(\mathbf{x}, \mathbf{z}) = \sum_{p \in P_{su}} \kappa_p(\mathbf{x}, \mathbf{z}).$$

Hence, we have $\kappa_G(\mathbf{x}, \mathbf{z}) = \kappa_t(\mathbf{x}, \mathbf{z})$ and we take $\kappa_s(\mathbf{x}, \mathbf{z}) = 1$. Consider a vertex u. All paths arriving at u from s must reach u by a directed edge into u. Similarly, any path p from s to a vertex v that has a directed edge e to u can be extended to a path pu from s to u. We can therefore decompose the sum

$$\kappa_u(\mathbf{x}, \mathbf{z}) = \sum_{p \in P_{su}} \kappa_p(\mathbf{x}, \mathbf{z}) = \sum_{v : v \to u} \sum_{p \in P_{sv}} \kappa_{pu}(\mathbf{x}, \mathbf{z})$$

$$= \sum_{v:v\to u} \kappa_{(v\to u)}(\mathbf{x},\mathbf{z}) \sum_{p\in P_{sv}} \kappa_p(\mathbf{x},\mathbf{z})$$
$$= \sum_{v:v\to u} \kappa_{(v\to u)}(\mathbf{x},\mathbf{z})\,\kappa_v(\mathbf{x},\mathbf{z}). \tag{9.6}$$

This is a set of linear equations for the vector of unknown values

$$\boldsymbol{\kappa} = (\kappa_u(\mathbf{x},\mathbf{z}))_{u\in V\setminus s}.$$

If we define the square matrix $\mathbf{A}$ indexed by $V \setminus s$ to have entries

$$\mathbf{A}_{uv} = \begin{cases} \kappa_{(v\to u)}(\mathbf{x},\mathbf{z}) & \text{if } (v\to u)\in E \\ -1 & \text{if } u = v, \\ 0 & \text{otherwise,} \end{cases} \tag{9.7}$$

and the vector $\mathbf{b}$ to have entries

$$b_u = \begin{cases} -\kappa_{(a\to u)}(\mathbf{x},\mathbf{z}) & \text{if } (a\to u)\in E \\ 0 & \text{otherwise,} \end{cases} \tag{9.8}$$

then we can rewrite equation (9.6) to give the computation:

Computation 9.24 [General graph kernels] Using $\mathbf{A}$ given by (9.7) and $\mathbf{b}$ by (9.8), the evaluation of the general graph kernel is obtained by solving

$$\mathbf{A}\boldsymbol{\kappa} = \mathbf{b},$$

and returning

$$\kappa(\mathbf{x},\mathbf{z}) = \kappa_t(\mathbf{x},\mathbf{z}).$$

■

Hence, provided the matrix $\mathbf{A}$ is invertible, we can in general solve the system in $O\left(|V|^3\right)$ steps to evaluate the kernel. The question of invertibility is related to the fact that once the graph has loops we cannot guarantee in general that the kernel evaluation will be bounded. Hence, care must be taken in defining graphs with loops. There are several types of graph structures other than the simple graph kernels with no loops for which more efficient computations exist.

If the graph is simple we can order the computation so that the values needed to compute $\kappa_u(\mathbf{x},\mathbf{z})$ have already been evaluated. We must perform a so-called topological sort of the vertices to obtain an order that is compatible with the partial order determined by the directed edges. This results in the computation described by the graph structure, multiplying the values stored at the previous nodes with the factor on the edge and summing, giving a complexity of $O(|E|)$.

Algorithm 9.25 [Simple graph kernels] Computation of simple graph kernels is given in Code Fragment 9.2. ■

Input	vectors $\mathbf{x}$ and $\mathbf{z}$ of length n, simple graph kernel $G = (V, E)$
Process	Find an ordering u_i of the vertices of G
2	such that $u_i \to u_j$ implies $i < j$.
	$\mathrm{DP}(1) = 1;$
6	for $i = 2 : \lvert V \rvert$
7	$\mathrm{DP}(i) = 0;$
9	for $j : u_j \to u_i$
10	$\mathrm{DP}(i) = \mathrm{DP}(i) + \kappa_{(u_j \to u_i)}(\mathbf{x}, \mathbf{z})\,\mathrm{DP}(j)\,;$
11	end
12	end
Output	kernel evaluation $\kappa_G(\mathbf{x}, \mathbf{z}) = \mathrm{DP}(\lvert V \rvert)$

Code Fragment 9.2. Pseudocode for simple graph kernels.

Remark 9.26 [Message passing] The evaluation of the array entry $\mathrm{DP}(i)$ at the vertex u_i can be viewed as the result of a set of messages received from vertices u_j, for which there is an edge $u_j \to u_i$. The message passed by the vertex u_j is its computed value $\mathrm{DP}(j)$. Algorithms of this type are referred to as *message passing algorithms* and play an important role in Bayesian belief networks. ■

Example 9.27 [Polynomial kernel] We can also represent the polynomial kernel as a simple graph kernel. The graph corresponding to its computation is shown in Figure 9.2. Clearly, paths from the leftmost to rightmost vertex

Fig. 9.2. Kernel graph for the polynomial kernel.

again correspond to its features, but notice how many different paths give rise to the same feature. It is this that gives rise to the different weightings that the various monomials receive.

Example 9.28 [ANOVA kernel] Recall the recursive computation of the ANOVA kernel illustrated in Table (9.5). Now consider creating a graph whose vertices are the entries in the table with directed edges to a vertex from the entries used to compute the value stored at that vertex. The edges are labelled with the factor that is applied to the value in the computation. Hence, for example, the vertex $(1,2)$ corresponding to the entry for $s=1$ and $m=2$ has a directed edge labelled x_2y_2 from vertex $(0,1)$ and an edge from $(1,1)$ labelled 1. This corresponds to the recursive computation

$$\kappa_1^2(\mathbf{x},\mathbf{z}) = (x_2z_2)\,\kappa_{1-1}^{2-1}(\mathbf{x},\mathbf{z}) + \kappa_1^{2-1}(\mathbf{x},\mathbf{z}).$$

We have included one extra vertex $(0,0)$ corresponding to $s=0$ and $m=0$ with edges labelled 1 from vertex $(0,i)$ to $(0,i+1)$ for $i=0,\ldots,n-1$. Finally, vertices (s,m) with $s>m$ together with associated edges have been deleted since they correspond to entries that are always 0. The graph is shown in Figure 9.3. We can view the graph as an illustration of the computational strategy, but it also encodes the features of the kernel as the set of directed paths from the vertex $(0,0)$ to the vertex (d,n). Each such path corresponds to the monomial formed by the product of all the factors on the path. Hence, for example, the path $(0,0)(1,1)\cdots(d,d)(d,d+1)\cdots(d,n)$ corresponds to the feature $x_1x_2\cdots x_d$. Clearly, to reach the vertex (d,n) we must move down d rows; each time we do so the feature indexing that column is included. Hence, each path corresponds to one set of d features with different paths selecting a different subset.

Remark 9.29 [Regular language kernels] There is a well-known equivalence between languages specified by regular expressions, also known as *regular languages*, and languages recognised by non-deterministic or deterministic finite state automata (FSA). We can use directed labelled graphs to specify non-deterministic FSAs. The nodes of the graph correspond to the states of the FSA, with the source being the initial state, and the sink the accepting state. The directed edges are labelled with symbols from the alphabet of the language or a symbol ε for so-called ε-transitions that do not generate a symbol. Hence, we can view a kernel graph as a non-deterministic FSA over the alphabet of base kernels, where ε-transitions correspond to edges labelled by 1. The languages of the non-deterministic FSA is the set of finite strings that are accepted by the FSA. This is exactly the set of monomial features specified by the kernel graph with the only difference that for the kernel graph a reordering of the base kernels specifies the same feature, while for the FSA, permuting the symbols in a string creates a distinct word in the language. Modulo this difference, we can use regular expressions as a

Fig. 9.3. Kernel graph for the ANOVA kernel.

shorthand for specifying the monomial features that we wish to include in our feature space, with the guarantee that the evaluation of the resulting kernel will be polynomial in the number of states of an equivalent non-deterministic FSA. ■

9.4 Diffusion kernels on graph nodes

One of the appeals of using kernel methods for pattern analysis is that they can be defined on non-vectorial data, making it possible to extend the application of the techniques to many different types of objects. In later chapters we will define specialist kernels for strings and trees to further illustrate this point, but here we introduce a method that can take any basic

comparison rule and convert it into a valid kernel. All that is required is some measure of similarity that captures at the simplest level association between two objects, perhaps only non-zero for objects that are closely related. The construction will use this to construct a refined similarity that relates all objects and is guaranteed to be a kernel.

The method again uses paths in graphs to create more complex relations between objects. In the kernel graphs described in the previous section the labelling of the graph was adjusted for each kernel evaluation $\kappa_G(\mathbf{x}, \mathbf{z})$. In this section we describe how similar ideas can be used to define a new kernel over a set of data, where each data item is associated with a vertex of the graph.

Definition 9.30 [Base similarity graph] A *base similarity graph* for a dataset S is a graph $G = (S, E)$, whose vertices are the data items in S, while the (undirected) links between them are labelled with some base similarity measure $\tau(\mathbf{x}, \mathbf{z})$. The *(base) similarity matrix* $\mathbf{B} = \mathbf{B}_1$ of G is indexed by the elements of S with entries

$$\mathbf{B}_{\mathbf{xz}} = \tau(\mathbf{x}, \mathbf{z}) = \tau_1(\mathbf{x}, \mathbf{z}).$$

■

In general, the base similarity could be a simple method of comparing the objects with the only restriction that it must be symmetric. Furthermore, no assumption is made that it corresponds to a kernel, though if it is a kernel the method will deliver derived kernels. This would therefore allow us to create more complex kernels in a principled way.

Remark 9.31 [Finite domains] In general a base similarity over a finite domain could be specified by the similarity matrix $\mathbf{B}$. This is the most general way to specify it. Both the kernels defined in this section are introduced for finite domains and deal directly with the similarity matrix. However, for large (or infinite) domains we require a compact function that exploits some features of the data, to create a similarity between objects. We will see that the kernels introduced here can be extended in this way when we treat a specific application area in Chapter 10. We want to emphasise here that the kernels can be defined between elements of non-metric spaces as long as some base comparisons are available. ■

Consider now the similarity $\tau_2(\mathbf{x}, \mathbf{z})$ obtained by summing the products

of the weights on all paths of length 2 between two vertices $\mathbf{x}$ and $\mathbf{z}$:

$$\tau_2(\mathbf{x},\mathbf{z}) = \sum_{(\mathbf{x}_0\mathbf{x}_1\mathbf{x}_2)\in P^2_{\mathbf{xz}}} \prod_{i=1}^{2} \tau_1(\mathbf{x}_{i-1},\mathbf{x}_i),$$

where we use $P^k_{\mathbf{xz}}$ to denote the set of paths of length k starting at $\mathbf{x}$ and finishing at $\mathbf{z}$.

Remark 9.32 [Paths as features] Notice that in this case we cannot view the paths as features, since the set of paths used in the sum depends on the arguments of the kernel function. This is in contrast to the fixed set of paths used for the graph kernels. We cannot therefore conclude that this is a kernel simply by considering sums and products of kernels. ■

Now consider how to compute the corresponding similarity matrix $\mathbf{B}_2$

$$\begin{aligned}\tau_2(\mathbf{x},\mathbf{z}) &= (\mathbf{B}_2)_{\mathbf{xz}} = \sum_{(\mathbf{x}_0\mathbf{x}_1\mathbf{x}_2)\in P^2_{\mathbf{xz}}} \prod_{i=1}^{2} \tau_1(\mathbf{x}_{i-1},\mathbf{x}_i)\\ &= \sum_{(\mathbf{x},\mathbf{x}_1)\in E} \tau_1(\mathbf{x},\mathbf{x}_1)\,\tau_1(\mathbf{x}_1,\mathbf{z})\\ &= \mathbf{B}^2_{\mathbf{xz}}.\end{aligned}$$

Hence, the new similarity matrix is simply the square of the base similarity matrix. It now is immediately apparent that the resulting function τ_2 is indeed a kernel, since the square of a symmetric matrix is always positive semi-definite, as its eigenvectors remain unchanged but the corresponding eigenvalues are each squared.

If we want to determine the features of this kernel we rewrite the sum as

$$\begin{aligned}(\mathbf{B}_2)_{\mathbf{xz}} &= \sum_{(\mathbf{x},\mathbf{x}_1)\in E} \tau_1(\mathbf{x},\mathbf{x}_1)\,\tau_1(\mathbf{x}_1,\mathbf{z}) = \sum_{i=1}^{\ell} \tau_1(\mathbf{x},\mathbf{x}_i)\,\tau_1(\mathbf{x}_i,\mathbf{z})\\ &= \left\langle (\tau_1(\mathbf{x},\mathbf{x}_i))_{i=1}^{\ell}, (\tau_1(\mathbf{z},\mathbf{x}_i))_{i=1}^{\ell} \right\rangle,\end{aligned}$$

where we have assumed that the datapoints are given as

$$S = \{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_\ell\}.$$

Hence, the kernel corresponds to the projection into the space with features given by the evaluation of the base similarity with each of the examples

$$\boldsymbol{\phi} : \mathbf{x} \longmapsto (\tau_1(\mathbf{x},\mathbf{x}_i))_{i=1}^{\ell}.$$

The intuition behind this construction is that we can enhance the base similarity by linking objects that share a number of 'common friends'. Hence we might consider taking a combination of the two matrices

$$\mathbf{B}_1 + \mathbf{B}_2.$$

We can similarly extend the consideration to paths of length k

$$\tau_k(\mathbf{x}, \mathbf{z}) = \sum_{(\mathbf{x}_0 \mathbf{x}_1 \ldots \mathbf{x}_k) \in P^k_{\mathbf{xz}}} \prod_{i=1}^{k} \tau_1(\mathbf{x}_{i-1}, \mathbf{x}_i).$$

A simple inductive argument shows that the corresponding similarity matrix $\mathbf{B}_k$ satisfies

$$\mathbf{B}_k = \mathbf{B}^k.$$

We would expect the relevance of longer paths to decay since linking objects through long paths might create erroneous measures of similarity. We can control the contribution of longer paths by introducing a decay factor λ into a general sum. For example if we choose

$$\mathbf{K} = \sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k \mathbf{B}^k,$$

we can write

$$\mathbf{K} = \exp(\lambda \mathbf{B}),$$

leading to the following definition.

Definition 9.33 [Exponential diffusion kernel] We will refer to

$$\mathbf{K} = \exp(\lambda \mathbf{B})$$

as the *exponential diffusion kernel* of the base similarity measure $\mathbf{B}$. ■

Definition 9.34 [von Neumann diffusion kernel] An alternative combination gives rise to the so-called *von Neumann diffusion kernel*

$$\mathbf{K} = \sum_{k=0}^{\infty} \lambda^k \mathbf{B}^k = (\mathbf{I} - \lambda \mathbf{B})^{-1},$$

which will only exist provided

$$\lambda < \|\mathbf{B}\|_2^{-1},$$

where $\|\mathbf{B}\|_2$ is the spectral radius of $\mathbf{B}$. ■

Note that the justification that the diffusion kernels are indeed valid kernels is contained in Remark 9.36 below.

Computation 9.35 [Evaluating diffusion kernels] We can evaluate these kernels by performing an eigen-decomposition of $\mathbf{B}$, apply the corresponding function to the eigenvalues and remultiply the components back together. This follows from the fact that for any polynomial $p(x)$

$$p\left(\mathbf{V}'\mathbf{\Lambda}\mathbf{V}\right) = \mathbf{V}'p\left(\mathbf{\Lambda}\right)\mathbf{V},$$

for any orthonormal matrix $\mathbf{V}$ and diagonal matrix $\mathbf{\Lambda}$. This strategy works since $\mathbf{B}$ is symmetric and so can be expressed in the form

$$\mathbf{B} = \mathbf{V}'\mathbf{\Lambda}\mathbf{V},$$

where $\mathbf{\Lambda}$ contains its real eigenvalues. ■

Remark 9.36 [Eigenvalues and relation to kernel PCA] The computation also re-emphasises the point that all of the kernels have the same eigenvectors as $\mathbf{B}$: they simply apply a reweighting to the eigenvalues. Note that using the diffusion construction when the base similarity measure is a kernel can therefore be viewed as a soft version of kernel PCA. PCA applies a threshold function to the eigenvalues, setting those smaller than λ_k to zero. In contrast the diffusion kernels apply a function to each eigenvalue: $(1-\mu\lambda)^{-1}$ for the von Neumann kernel and $\exp(\mu\lambda)$ for the exponential kernel, that increases the relative weight of the larger eigenvalues when compared to the smaller ones. The fact that the functions $(1-\mu\lambda)^{-1}$ and $\exp(\mu\lambda)$ have positive values for λ in the indicated range ensures that the kernel matrices are indeed positive semi-definite. ■

We will further explore these kernels in applications to textual data in Chapter 10.

9.5 Kernels on sets

As another example of a non-vectorial space consider the set of subsets $\mathcal{P}(D)$ of a fixed domain D, a set sometimes referred to as the *power set* of D. The elements of the input space are therefore sets. They could include all the subsets of D or some subselection of the whole power set

$$X \subseteq \mathcal{P}(D).$$

Consider now two elements A_1 and A_2 of X, that is two subsets of the domain D. In Chapter 2 we gave one example of a kernel $\kappa(A_1, A_2)$ on such

subsets of D defined as

$$\kappa(A_1, A_2) = 2^{|A_1 \cap A_2|}.$$

In this section we will extend the range of available kernels.

Suppose that D is equipped with a measure or probability density function μ. If D is finite this is simply a mapping from D to the positive reals. For infinite D there are technical details involving a σ-algebra of subsets known as the measurable sets. We must assume that the sets included in X are all measurable. In either case μ defines an integration over the set D

$$\mu(f) = \int_D f(a)\, d\mu(a),$$

for so-called measurable functions. In particular the indicator function I_A of a measurable set A defines the measure (or probability) of the set by

$$\mu(A) = \mu(I_A) \in \mathbb{R}^+.$$

Definition 9.37 [Intersection kernel] We now define the *intersection kernel* over the subsets of D by

$$\kappa_\cap(A_1, A_2) = \mu(A_1 \cap A_2),$$

that is, the measure of the intersection between the two sets. ■

This can be seen to be a valid kernel by considering the feature space of all measurable functions with the inner product defined by

$$\langle f_1, f_2 \rangle = \int_D f_1(a)\, f_2(a)\, d\mu(a).$$

The feature mapping is now given by

$$\boldsymbol{\phi} : A \longmapsto I_A,$$

implying that

$$\begin{aligned}
\kappa_\cap(A_1, A_2) &= \mu(A_1 \cap A_2) = \int_D I_{A_1 \cap A_2}(a)\, d\mu(a) \\
&= \int_D I_{A_1}(a)\, I_{A_2}(a)\, d\mu(a) = \langle I_{A_1}, I_{A_2} \rangle \\
&= \langle \boldsymbol{\phi}(A_1), \boldsymbol{\phi}(A_2) \rangle,
\end{aligned}$$

as required. This construction also makes clear that we could have mapped to the indicator function of the complementary set provided the measure of this set is finite.

Definition 9.38 [Union complement kernel] Assuming that $\mu(D) = 1$, we have the *union complement kernel*

$$\tilde{\kappa}(A_1, A_2) = \mu((D \setminus A_1) \cap (D \setminus A_2)) = 1 - \mu(A_1 \cup A_2).$$

■

There are many cases in which we may want to represent data items as sets. There is also the potential for an interesting duality, since we represent an object by the items it contains, while the items themselves can also be described by the set of objects they are contained in. This duality will be exploited in the case of kernels for text briefly discussed here but covered in detail in Chapter 10.

Example 9.39 Consider a document as a set of words from a dictionary D. The similarity between two documents d_1 and d_2 will then be the measure of the intersection between those two sets. The importance of a word can also be encoded in a distribution over D weighting the calculation of the measure of the intersection. As mentioned above we can also compare words by defining a function over documents indicating in which documents they appear. This induces a duality between term-based and document-based representations that can be exploited in many ways. This duality corresponds to the duality implied by kernel-based representations, and hence when using this kind of data there is an interesting interpretation of dual features that will be explored further in Chapter 10.

Remark 9.40 [Agreement kernel] We can sum the two kernels $\kappa_\cap$ and $\tilde{\kappa}$ described above to obtain the kernel

$$\begin{aligned}\kappa(A_1, A_2) &= \tilde{\kappa}(A_1, A_2) + \kappa_\cap(A_1, A_2) \\ &= 1 - \mu(A_1 \cup A_2) + \mu(A_1 \cap A_2) \\ &= 1 - \mu(A_1 \setminus A_2) - \mu(A_2 \setminus A_1),\end{aligned}$$

that is, the measure of the points on which the two indicator functions agree. If we used indicator functions I_A^- which mapped elements not in the set to -1, then, after introducing a factor of $1/2$, this would give rise to the kernel

$$\begin{aligned}\hat{\kappa}(A_1, A_2) &= \frac{1}{2}\left\langle I_{A_1}^-, I_{A_2}^-\right\rangle \\ &= \frac{1}{2}(\kappa(A_1, A_2) - \mu(A_1 \setminus A_2) - \mu(A_2 \setminus A_1)) \\ &= \frac{1}{2} - \mu(A_1 \setminus A_2) - \mu(A_2 \setminus A_1).\end{aligned}$$

The second kernel is simply the first minus 1/2. This will be a better starting point since it leaves more options open – we can always add 1/2 as this is equivalent to adding an extra feature with constant value $\sqrt{1/2}$, but in general we cannot subtract a constant from the kernel matrix. ■

9.6 Kernels on real numbers

There is an interesting application of the intersection kernel that defines further kernels for real numbers. This can in turn be combined with the ANOVA kernel construction to obtain new kernels between vectors. Let us first consider real numbers $x, z \in \mathbb{R}$. The simplest and most obvious kernel is of course

$$\kappa(x, z) = xz,$$

though we can equally use the polynomial or Gaussian kernel construction in one-dimension to obtain for example

$$\kappa(x, z) = \exp\left(-\frac{(x-z)^2}{2\sigma^2}\right).$$

Now consider representing $x \in \mathbb{R}^+$ by the interval $[0, x]$, that is the set $\{y : 0 \leq y \leq x\}$. Applying an intersection kernel with the standard integration measure we obtain

$$\kappa(x, z) = \min(x, z). \tag{9.9}$$

Hence, normalising we obtain that

$$\kappa(x, z) = \frac{\min(x, z)}{\sqrt{\min(x, x)\min(z, z)}} = \frac{\min(x, z)}{\sqrt{xz}}$$

is a kernel. Similarly representing $x \in \mathbb{R}^+$ by the interval $[0, x^{-p}]$ for some $p > 0$ and applying the intersection construction, we obtain that

$$\kappa(x, z) = \frac{1}{\max(x, z)^p}$$

is a kernel, which when normalised gives the kernel

$$\kappa(x, z) = \frac{(xz)^{p/2}}{\max(x, z)^p}.$$

For $p = 1$ we have

$$\kappa(x, z) = \frac{\sqrt{xz}}{\max(x, z)}, \tag{9.10}$$

while for $p = 2$ we get an interesting variant of the standard inner product

$$\kappa(x, z) = \frac{xz}{\max(x, z)^2}.$$

If there is an upper bound C on the value of x we can map to the interval $[x, C]$ to obtain the kernel

$$\kappa(x, z) = C - \max(x, z).$$

Furthermore we are not restricted to using 1-dimensional sets. For example mapping $x \in \mathbb{R}^+$ to the rectangle

$$[0, x] \times \left[0, x^{-1}\right],$$

results in an intersection kernel

$$\kappa(x, z) = \frac{\min(x, z)}{\max(x, z)}.$$

Of course the 2-dimensional construction is more natural for two-dimensional vectors and gives kernels such as

$$\kappa((x_1, x_2), (z_1, z_2)) = \min(x_1, z_1) \min(x_2, z_2)$$

and

$$\kappa((x_1, x_2), (z_1, z_2)) = \frac{\sqrt{x_1 x_2 z_1 z_2}}{\max(x_1, z_1) \max(x_2, z_2)}.$$

Actually these last two kernels could have been obtained by applying the generalised ANOVA construction of degree 2 (see Remark 9.20) using the base kernels given in equations (9.9) and (9.10) respectively.

Finally consider mapping $z \in \mathbb{R}$ to the function $g(|\cdot - z|) \in L_2(\mu)$ for some fixed g and defining the inner product via the integration

$$\kappa(z_1, z_2) = \int g(|x - z_1|) g(|x - z_2|) \, d\mu(x).$$

This is clearly a kernel since the image of z is a point in the corresponding Hilbert space. Taking μ to be the standard measure, the choice of the function g could be a Gaussian to give a soft comparison of two numbers or for example a threshold function

$$g(x) = \begin{cases} 1 & \text{if } x \leq C; \\ 0 & \text{otherwise.} \end{cases}$$

This results in the kernel

$$\kappa(z_1, z_2) = \max(0, 2C - |z_1 - z_2|). \tag{9.11}$$

Remark 9.41 [Distance induced by min kernel] The distance induced by the 'min' kernel of equation (9.9) is

$$\begin{aligned} \|\boldsymbol{\phi}(x) - \boldsymbol{\phi}(z)\|^2 &= x + z - 2\min(x, z) \\ &= \max(x, z) - \min(x, z) = |x - z|, \end{aligned}$$

as opposed to the normal Euclidean distance induced by the standard kernel xz. ■

Remark 9.42 [Spline kernels] Notice that we can put together the ideas of the generalised ANOVA construction of Remark 9.20 with the the 'min' kernel

$$\kappa(x, z) = \min(x, z)$$

to obtain the so-called spline kernels for multi-dimensional data. ■

Derived subsets kernel Our final example of defining kernels over sets considers the case where we already have a kernel κ_0 defined on the elements of a universal set U.

Definition 9.43 [Derived subsets kernel] Given a base kernel κ_0 on a set U, we define the *subset kernel* derived from κ_0 between finite subsets A and B of U by

$$\kappa(A, B) = \sum_{a \in A} \sum_{b \in B} \kappa_0(a, b).$$

■

The following proposition verifies that a derived subsets kernel is indeed a kernel.

Proposition 9.44 *Given a kernel κ_0 on a set U the derived subsets kernel κ is an inner product in an appropriately defined feature space and hence is a kernel.*

Proof Let $\boldsymbol{\phi}_0$ be an embedding function into a feature space F_0 for the kernel κ_0 so that

$$\kappa_0(a, b) = \langle \boldsymbol{\phi}_0(a), \boldsymbol{\phi}_0(b) \rangle.$$

Consider the embedding function for a finite subset $A \subseteq U$ defined by

$$\boldsymbol{\phi}(A) = \sum_{a \in A} \boldsymbol{\phi}_0(a) \in F_0.$$

We have

$$\begin{aligned}\kappa(a,b) &= \sum_{a\in A}\sum_{b\in B}\kappa_0(a,b) = \sum_{a\in A}\sum_{b\in B}\langle\boldsymbol{\phi}_0(a),\boldsymbol{\phi}_0(b)\rangle \\ &= \left\langle\sum_{a\in A}\boldsymbol{\phi}_0(a),\sum_{b\in B}\boldsymbol{\phi}_0(b)\right\rangle = \langle\boldsymbol{\phi}(A),\boldsymbol{\phi}(B)\rangle,\end{aligned}$$

as required. □

Example 9.45 Another application of the derived subsets kernel idea is in allowing deformed matches in normal kernels. This can be useful when we want to introduce specific invariances into the kernel design. Suppose we are computing the kernel between the images $\mathbf{x},\mathbf{z}\in X$ of two handwritten characters. We know that the two arguments should be considered equivalent if slight transformations of the images are applied, such as small rotations, translations, thickenings and even the addition of some noise. Suppose we devise a function

$$\boldsymbol{\phi}: X\to\mathcal{P}(X)$$

that expands the argument to a set of arguments that are all equivalent to it, hence introducing a 'jittering' effect. The similarity of two images $\mathbf{x}$ and $\mathbf{z}$ can now be measured in $\mathcal{P}(X)$ between the sets of images $\boldsymbol{\phi}(\mathbf{x})$ and $\boldsymbol{\phi}(\mathbf{z})$ using the derived subsets kernel for a base kernel for comparing images.

9.7 Randomised kernels

There are cases of kernels where the feature space is explicitly constructed by a vector

$$\boldsymbol{\phi}(\mathbf{x}) = (\phi_i(\mathbf{x}))_{i\in I},$$

and each of the individual $\phi_i(\mathbf{x})$ are simple to compute, but the size of the set I makes complete evaluation of the feature vector prohibitively expensive. If the feature space inner product is given by

$$\kappa(\mathbf{x},\mathbf{z}) = \left\langle(\phi_i(\mathbf{x}))_{i\in I},(\phi_i(\mathbf{z}))_{i\in I}\right\rangle = \sum_{i\in I}\mu_i\phi_i(\mathbf{x})\,\phi_i(\mathbf{z}),$$

with $\sum_i\mu_i = 1$ then we can estimate the value of the inner product by sampling indices according to $\boldsymbol{\mu}$ and forming an empirical estimate of the inner product through an average of N randomly drawn features

$$\hat{\kappa}(\mathbf{x},\mathbf{z}) = \frac{1}{N}\sum_{i\sim\boldsymbol{\mu}}\phi_i(\mathbf{x})\,\phi_i(\mathbf{z}).$$

We can bound the maximal deviation that can be caused by changing a single feature by $2c/N$ where $[-c, c]$ is a bound on the range of the functions ϕ_i. Applying McDiarmid's Theorem 4.5 we obtain

$$P\left\{\hat{\kappa}\left(\mathbf{x}, \mathbf{z}\right) - \mathbb{E}\hat{\kappa}\left(\mathbf{x}, \mathbf{z}\right) \geq \epsilon\right\} \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{N} 4c^2/N^2}\right) = \exp\left(\frac{-N\epsilon^2}{2c^2}\right).$$

Observing that $\mathbb{E}\hat{\kappa}\left(\mathbf{x}, \mathbf{z}\right) = \kappa\left(\mathbf{x}, \mathbf{z}\right)$ and that the same argument can be applied to $-\hat{\kappa}\left(\mathbf{x}, \mathbf{z}\right)$ gives

$$P\left\{\left|\hat{\kappa}\left(\mathbf{x}, \mathbf{z}\right) - \kappa\left(\mathbf{x}, \mathbf{z}\right)\right| \geq \epsilon\right\} \leq 2\exp\left(\frac{-N\epsilon^2}{2c^2}\right).$$

Hence, with high probability we obtain a good estimate of the true kernel evaluation even using only a modest number of features. If for example we require that with probability at least $1-\delta$ all of the entries in an $\ell \times \ell$ kernel matrix are within ϵ of their true values, we should choose N so that

$$2\exp\left(\frac{-N\epsilon^2}{2c^2}\right) \leq \frac{2\delta}{\ell\left(\ell+1\right)},$$

implying that

$$N \geq \frac{2c^2}{\epsilon^2} \ln \frac{\ell\left(\ell+1\right)}{\delta}.$$

This leads to the following computation.

Computation 9.46 [Evaluating randomised kernels] With probability at least $1-\delta$ we can estimate all the entries of the kernel matrix of a randomised kernel to accuracy ϵ by sampling

$$N \geq \frac{2c^2}{\epsilon^2} \ln \frac{\ell\left(\ell+1\right)}{\delta}$$

features $\boldsymbol{\phi}_k$ and setting

$$\hat{\kappa}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{1}{N}\sum_{k=1}^{N} \phi_k\left(\mathbf{x}_i\right)\phi_k\left(\mathbf{x}_j\right),$$

for $i, j = 1, \ldots, \ell$, where the range of the features is bounded in $[-c, c]$. ■

Remark 9.47 [Sampling intersection kernels] One example where this could arise is in kernels defined in terms of intersection of sets

$$\kappa_{\cap}\left(A_1, A_2\right) = \mu\left(A_1 \cap A_2\right).$$

The features here are the elements of the domain D. If the structure of the sets makes it difficult to integrate the measure over their intersection, we may be able to use the randomisation approach. This is equivalent to estimating the integral

$$\mu(A_1 \cap A_2) = \int_D I_{A_1 \cap A_2}(a)\, d\mu(a)$$

via a Monte Carlo random sampling. Hence, if we can efficiently generate random examples according to μ, we can apply the randomisation approach. An example where this might be relevant is for a kernel between boolean functions. Estimating when a boolean function has any satisfying assignment becomes NP-hard even if, for example, we restrict the structure of the function to a disjunctive normal form with 3 literals per clause. Hence, an exact evaluation of the integral would be computationally infeasible. We can, however, sample binary vectors and check if they satisfy both functions. Note that such a kernel would be needed in an application where the examples were boolean functions and we were looking for patterns among them. ■

9.8 Other kernel types

This section concludes this chapter's overview of methods for designing kernels by outlining the three kernel types to be presented in the last three chapters of the book. Although they will be discussed in detail later, it is worth placing them in the context of the current survey to furnish a fuller picture of the diversity of data and techniques available.

9.8.1 Kernels from successive embeddings

We saw in Chapter 3 that the class of kernel functions satisfies a series of closure properties. Furthermore, we saw that as long as we can show the existence of an embedding function $\boldsymbol{\phi}$ such that

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle$$

then we know that κ is a kernel, even if we are not able either computationally or explicitly to construct $\boldsymbol{\phi}$. These two facts open up the possibility of defining kernels by successive adjustments, either performing successive embeddings or manipulating the given kernel function (or matrix). In this way we can sculpt the feature space for the particular application.

We must exploit knowledge of the domain by designing a series of simple

embeddings each of which could for example introduce some invariance into the embedding by ensuring that all transformations of an example under a particular invariance are mapped to the same image. Any classifier using the resulting kernel will be invariant to that transformation. Note also that further embeddings cannot undo this invariance. Hence, the final kernel will exhibit invariance to all of the transformations considered.

The overall kernel resulting from a sequence of such embeddings is by definition a valid kernel, although we might not be able to explicitly construct its features. The proof of validity follows the stages of its construction by composing the sequence of successive embeddings to create the embedding corresponding to the overall kernel

$$\boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\psi}_1(\mathbf{x}) \circ \boldsymbol{\psi}_2(\mathbf{x}) \circ \cdots \circ \boldsymbol{\psi}_n(\mathbf{x}).$$

Figure 9.4 illustrates this point with a sequence of two embeddings progressively increasing the separation of the datapoints associated with the two classes of a classification problem.

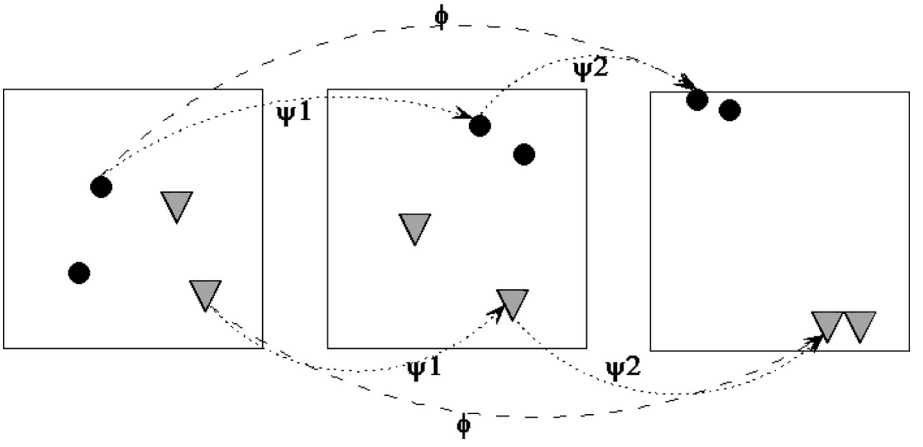

Fig. 9.4. A sequence of two embeddings can be composed to create an overall embedding with desirable properties.

Example 9.48 In Chapter 10 we will see this approach applied in the case of text documents, where we can successively embed a document to gradually take care of synonymy, semantics, stop words, removal of document length information, and so on. Each of these properties is realised by composing with an additional embedding function and can be seen as implementing an invariance. A similar approach may well be applicable in other domains such as, for example, machine vision.

9.8.2 Kernels over general structures

In Remark 9.20 we saw how the ANOVA construction could be used to define a kernel in a recursive fashion, provided a set of base kernels was available. This situation is not uncommon in recursively-defined data structures, such as strings and trees. For example, comparing two symbols in the sequence provides a base kernel for the case of strings. We will show how this can be used to recursively build a kernel between sequences of symbols, hence enabling pattern analysis of documents, DNA sequences, etc.

This approach makes it possible to build general kernels defined over structured data objects, provided we have a way to compare the elementary components of such structures. Typically the method of combination corresponds to a recursive computation. The approach is not restricted to strings or mathematical structures such as trees or graphs, but can be applied to complex objects that arise in particular application domains such as web pages, chemometrics, and so on.

9.8.3 Kernels from generative information

When designing a kernel for a particular application it is vital that it should incorporate as much domain knowledge as possible. At the simplest level the similarity measure defined by the kernel should reflect our understanding of the relative distances between data items. Indeed one advantage of the kernel approach is that this question is often easier to answer than the direct definition of pattern functions that would otherwise be needed.

There are, however, other more sophisticated methods of incorporating such domain knowledge. For example we can construct a model of the way that the data are generated, in what are known as *generative models*. In this approach, the generative models could be deterministic or probabilistic, and could be either simple functions or complex graphical structures such as finite state automata or hidden Markov models (HMMs). In Chapter 12 we will explore methods for converting these models of the data into kernel functions, hence developing several frameworks for incorporating prior knowledge in a principled way including the well-known Fisher kernels.

9.9 Summary

- Many kernels can be computed in closed form including the polynomial and Gaussian kernels.
- Many kernels can be defined using recursive relations including the polynomial kernel, the all-subsets kernel and the ANOVA kernel.

- Graph kernels generalise the family of recursively-defined kernels allowing more detailed sculpting of which features are to be included.
- Kernels over graph nodes can be defined in terms of diffusion processes starting from a general base similarity measure.
- Simple kernels can be defined based on fundamental set operations.
- Sampling can enable accurate estimation of kernels whose feature set prohibits efficient exact evaluation.

9.10 Further reading and advanced topics

In the early years of research on kernel methods only functions in closed form were considered. The introduction of ANOVA kernels gave the first example of kernels defined by means of a recursive relation, which could be efficiently evaluated using dynamic programming. That in turn triggered the observation that other kernels could be defined in terms of recursive relations, or even in terms of very general computational procedures. At the same time, the idea that kernels need to be defined on vectorial inputs was been recognised as unnecessarily restrictive. From 1999 the first kernels between strings defined by means of recursions started to appear [155], [154], [52]. After those first contributions, a flood of different approaches have been proposed, the main ones are summarised in this chapter with more detailed presentations of many of them deferred to Chapter 11. More recently it has been recognised that one does not even need to have a kernel function at all, just a kernel matrix for the available data. This has led to optimisation procedures for directly inferring the kernel matrix, which are not discussed here.

Gaussian and polynomial kernels were already discussed in the first paper on support vector machines [16]. Gaussian kernels in particular have been investigated for a long time within the literature on reproducing kernel Hilbert spaces [153]. ANOVA kernels were first suggested by Burges and Vapnik (under the name of Gabor kernels) in 1995 [21] in the form described in Computation 9.18. The recursion described in Algorithm 9.14 was proposed by Chris Watkins in 1999 [154]. The paper of Takimoto and Warmuth 2002 [130] introduces the 'all subsets kernel' as well as the idea of kernels based on paths in a graph.

These regular language kernels are related to the rational kernels proposed in [26].

Diffusion kernels were proposed by Imre Kondor in [80], while von Neumann kernels were introduced in [69]. The bag-of-words kernels were pro-

posed by Thorsten Joachims [65] and can be considered as an example of kernels between sets.

The discussion of kernels over sets or between numbers is largely a collection of folk results, but strongly influenced by discussions with Chris Watkins.

Kernels over general structures were simultaneouly elaborated by Watkins and by Haussler [155], [154], [52] and will be discussed in further detail in Chapter 11.

Kernels based on data compression ideas from information theory have been proposed [36] and form yet another approach for defining similarity measures satisfying the finitely positive semi-definite property. Kernels defined on sets of points have also been recently explored in [63], while an interesting approach for graph kernels has been proposed by [73].

For constantly updated pointers to online literature and free software see the book's companion website: www.kernel-methods.net.