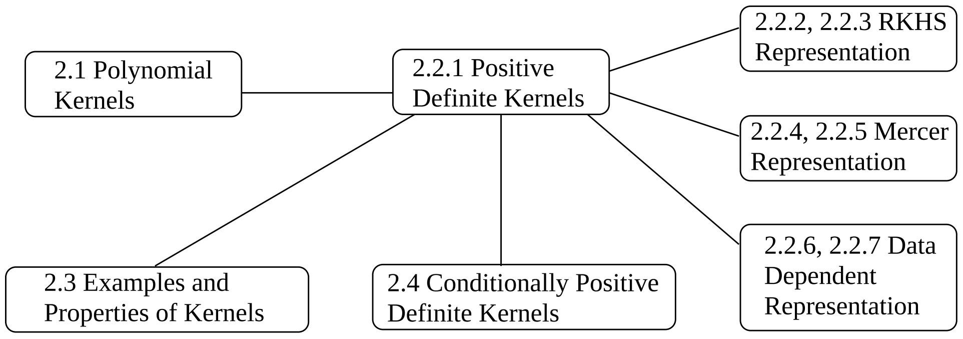
2.1 Polynomial Kernels
2.2.1 Positive Definite Kernels
2.2.2, 2.2.3 RKHS Representation
2.2.4, 2.2.5 Mercer Representation
2.2.6, 2.2.7 Data Dependent Representation
2.3 Examples and Properties of Kernels
2.4 Conditionally Positive Definite Kernels

2.1 Product Features

In this section, we think of $\mathcal{X}$ as a subset of the vector space $\mathbb{R}^N$, ($N \in \mathbb{N}$), endowed with the canonical dot product (1.3).

Monomial Features

Suppose we are given patterns $x \in \mathcal{X}$ where most information is contained in the dth order products (so-called monomials) of entries $[x]_j$ of x,

$$[x]_{j_1} \cdot [x]_{j_2} \cdots [x]_{j_d}, \tag{2.3}$$

where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. Often, these monomials are referred to as *product features*. These features form the basis of many practical algorithms; indeed, there is a whole field of pattern recognition research studying *polynomial classifiers* [484], which is based on first extracting product features and then applying learning algorithms to these features. In other words, the patterns are preprocessed by mapping into the feature space $\mathcal{H}$ of all products of d entries. This has proven quite effective in visual pattern recognition tasks, for instance. To understand the rationale for doing this, note that visual patterns are usually represented as vectors whose entries are the pixel intensities. Taking products of entries of these vectors then corresponds to taking products of pixel intensities, and is thus akin to taking logical "and" operations on the pixels. Roughly speaking, this corresponds to the intuition that, for instance, a handwritten "8" constitutes an eight if there is a top circle *and* a bottom circle. With just one of the two circles, it is not half an "8," but rather a "0." Nonlinearities of this type are crucial for achieving high accuracies in pattern recognition tasks.

Let us take a look at this feature map in the simple example of two-dimensional patterns, for which $\mathcal{X} = \mathbb{R}^2$. In this case, we can collect all monomial feature extractors of degree 2 in the nonlinear map

$$\Phi : \mathbb{R}^2 \to \mathcal{H} = \mathbb{R}^3, \tag{2.4}$$

$$([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2). \tag{2.5}$$

This approach works fine for small toy examples, but it fails for realistically sized problems: for N-dimensional input patterns, there exist

$$N_{\mathcal{H}} = \binom{d+N-1}{d} = \frac{(d+N-1)!}{d!(N-1)!} \tag{2.6}$$

different monomials (2.3) of degree d, comprising a feature space $\mathcal{H}$ of dimension $N_{\mathcal{H}}$. For instance, 16×16 pixel input images and a monomial degree $d = 5$ thus yield a dimension of almost 10^{10}.

In certain cases described below, however, there exists a way of *computing dot products* in these high-dimensional feature spaces without explicitly mapping into the spaces, by means of kernels nonlinear in the input space $\mathbb{R}^N$. Thus, if the subsequent processing can be carried out using dot products exclusively, we are able to deal with the high dimension.

We now describe how dot products in polynomial feature spaces can be computed efficiently, followed by a section in which we discuss more general feature spaces. In order to compute dot products of the form $\langle \Phi(x), \Phi(x') \rangle$, we employ kernel representations of the form

Kernel

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle , \tag{2.7}$$

which allow us to compute the value of the dot product in $\mathcal{H}$ without having to explicitly compute the map Φ.

What does k look like in the case of polynomial features? We start by giving an example for $N = d = 2$, as considered above [561]. For the map

$$\Phi : ([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2, [x]_2[x]_1), \tag{2.8}$$

(note that for now, we have considered $[x]_1[x]_2$ and $[x]_2[x]_1$ as separate features; thus we are looking at *ordered* monomials) dot products in $\mathcal{H}$ take the form

$$\langle \Phi(x), \Phi(x') \rangle = [x]_1^2[x']_1^2 + [x]_2^2[x']_2^2 + 2[x]_1[x]_2[x']_1[x']_2 = \langle x, x' \rangle^2 . \tag{2.9}$$

In other words, the desired kernel k is simply the square of the dot product in input space. The same works for arbitrary $N, d \in \mathbb{N}$ [62]: as a straightforward generalization of a result proved in the context of polynomial approximation [412, Lemma 2.1], we have:

Proposition 2.1 *Define C_d to map $x \in \mathbb{R}^N$ to the vector $C_d(x)$ whose entries are all possible dth degree ordered products of the entries of x. Then the corresponding kernel computing the dot product of vectors mapped by C_d is*

$$k(x, x') = \langle C_d(x), C_d(x') \rangle = \langle x, x' \rangle^d . \tag{2.10}$$

Polynomial Kernel

Proof We directly compute

$$\langle C_d(x), C_d(x') \rangle = \sum_{j_1=1}^{N} \dots \sum_{j_d=1}^{N} [x]_{j_1} \cdot \dots \cdot [x]_{j_d} \cdot [x']_{j_1} \cdot \dots \cdot [x']_{j_d} \tag{2.11}$$

$$= \sum_{j_1=1}^{N} [x]_{j_1} \cdot [x']_{j_1} \dots \sum_{j_d=1}^{N} [x]_{j_d} \cdot [x']_{j_d} = \left(\sum_{j=1}^{N} [x]_j \cdot [x']_j \right)^d = \langle x, x' \rangle^d . \qquad \blacksquare$$

Note that we used the symbol C_d for the feature map. The reason for this is that we would like to reserve Φ_d for the corresponding map computing *unordered* product features. Let us construct such a map Φ_d, yielding the same value of the dot product. To this end, we have to compensate for the multiple occurrence of certain monomials in C_d by scaling the respective entries of Φ_d with the square roots of their numbers of occurrence. Then, by this construction of Φ_d, and (2.10),

$$\langle \Phi_d(x), \Phi_d(x') \rangle = \langle C_d(x), C_d(x') \rangle = \langle x, x' \rangle^d . \tag{2.12}$$

For instance, if n of the j_i in (2.3) are equal, and the remaining ones are different, then the coefficient in the corresponding component of Φ_d is $\sqrt{(d-n+1)!}$. For the general case, see Problem 2.2. For Φ_2, this simply means that [561]

$$\Phi_2(x) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2). \tag{2.13}$$

The above reasoning illustrates an important point pertaining to the construction of feature spaces associated with kernel functions. Although they map into different feature spaces, Φ_d and C_d are both valid instantiations of feature maps for $k(x, x') = \langle x, x' \rangle^d$.

To illustrate how monomial feature kernels can significantly simplify pattern recognition tasks, let us consider a simple toy example.

Toy Example

Example 2.2 (Monomial Features in 2-D Pattern Recognition) *In the example of Figure 2.1, a non-separable problem is reduced to the construction of a separating hyperplane by preprocessing the input data with Φ_2. As we shall see in later chapters, this has advantages both from the* computational *point of view (there exist efficient algorithms for computing the hyperplane) and from the* statistical *point of view (there exist guarantees for how well the hyperplane will generalize to unseen test points).*

In more realistic cases, e.g., if x represents an image with the entries being pixel values, polynomial kernels $\langle x, x' \rangle^d$ enable us to work in the space spanned by products of any d pixel values — provided that we are able to do our work solely in terms of dot products, without any explicit usage of a mapped pattern $\Phi_d(x)$. Using kernels of the form (2.10), we can take higher-order statistics into account, without the combinatorial explosion (2.6) of time and memory complexity which accompanies even moderately high N and d.

To conclude this section, note that it is possible to modify (2.10) such that it maps into the space of all monomials *up to* degree d, by defining $k(x, x') = (\langle x, x' \rangle + 1)^d$ (Problem 2.17). Moreover, in practice, it is often useful to multiply the kernel by a scaling factor c to ensure that its numeric range is within some bounded interval, say $[-1, 1]$. The value of c will depend on the dimension and range of the data.

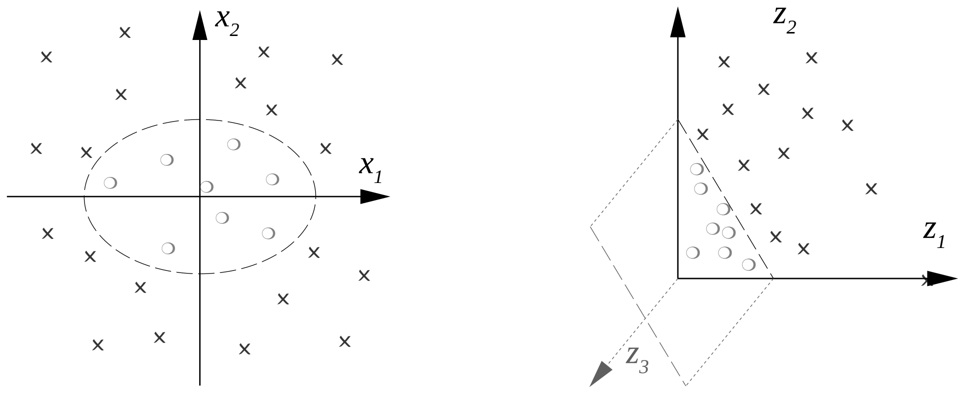

Figure 2.1 Toy example of a binary classification problem mapped into feature space. We assume that the true decision boundary is an ellipse in input space (left panel). The task of the learning process is to estimate this boundary based on empirical data consisting of training points in both classes (crosses and circles, respectively). When mapped into feature space via the nonlinear map $\Phi_2(x) = (z_1, z_2, z_3) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2)$ (right panel), the ellipse becomes a hyperplane (in the present simple case, it is parallel to the z_3 axis, hence all points are plotted in the (z_1, z_2) plane). This is due to the fact that ellipses can be written as linear equations in the entries of (z_1, z_2, z_3). Therefore, in feature space, the problem reduces to that of estimating a hyperplane from the mapped data points. Note that via the polynomial kernel (see (2.12) and (2.13)), the dot product in the three-dimensional space can be computed without computing Φ_2. Later in the book, we shall describe algorithms for constructing hyperplanes which are based on dot products (Chapter 7).

2.2 The Representation of Similarities in Linear Spaces

In what follows, we will look at things the other way round, and start with the kernel rather than with the feature map. Given some kernel, can we construct a feature space such that the kernel computes the dot product in that feature space; that is, such that (2.2) holds? This question has been brought to the attention of the machine learning community in a variety of contexts, especially during recent years [4, 152, 62, 561, 480]. In functional analysis, the same problem has been studied under the heading of *Hilbert space representations* of kernels. A good monograph on the theory of kernels is the book of Berg, Christensen, and Ressel [42]; indeed, a large part of the material in the present chapter is based on this work. We do not aim to be fully rigorous; instead, we try to provide insight into the basic ideas. As a rule, all the results that we state without proof can be found in [42]. Other standard references include [16, 455].

There is one more aspect in which this section differs from the previous one: the latter dealt with vectorial data, and the domain $\mathcal{X}$ was assumed to be a subset of $\mathbb{R}^N$. By contrast, the results in the current section hold for data drawn from domains which need no structure, other than their being nonempty sets. This generalizes kernel learning algorithms to a large number of situations where a vectorial representation is not readily available, and where one directly works