2

Kernel methods: an overview

In Chapter 1 we gave a general overview to pattern analysis. We identified three properties that we expect of a pattern analysis algorithm: computational efficiency, robustness and statistical stability. Motivated by the observation that recoding the data can increase the ease with which patterns can be identified, we will now outline the kernel methods approach to be adopted in this book. This approach to pattern analysis first embeds the data in a suitable feature space, and then uses algorithms based on linear algebra, geometry and statistics to discover patterns in the embedded data.

The current chapter will elucidate the different components of the approach by working through a simple example task in detail. The aim is to demonstrate all of the key components and hence provide a framework for the material covered in later chapters.

Any kernel methods solution comprises two parts: a module that performs the mapping into the embedding or feature space and a learning algorithm designed to discover linear patterns in that space. There are two main reasons why this approach should work. First of all, detecting linear relations has been the focus of much research in statistics and machine learning for decades, and the resulting algorithms are both well understood and efficient. Secondly, we will see that there is a computational shortcut which makes it possible to represent linear patterns efficiently in high-dimensional spaces to ensure adequate representational power. The shortcut is what we call a kernel function.

2.1 The overall picture

This book will describe an approach to pattern analysis that can deal effectively with the problems described in Chapter 1 one that can detect stable patterns robustly and efficiently from a finite data sample. The strategy adopted is to embed the data into a space where the patterns can be discovered as linear relations. This will be done in a *modular* fashion. Two distinct components will perform the two steps. The initial mapping component is defined implicitly by a so-called *kernel function*. This component will depend on the specific data type and domain knowledge concerning the patterns that are to be expected in the particular data source. The pattern analysis algorithm component is general purpose, and robust. Furthermore, it typically comes with a statistical analysis of its stability. The algorithm is also *efficient*, requiring an amount of computational resources that is polynomial in the size and number of data items even when the dimension of the embedding space grows exponentially.

The strategy suggests *a software engineering approach* to learning systems' design through the breakdown of the task into subcomponents and the reuse of key modules.

In this chapter, through the example of least squares linear regression, we will introduce all of the main ingredients of kernel methods. Though this example means that we will have restricted ourselves to the particular task of supervised regression, four key aspects of the approach will be highlighted.

(i) Data items are embedded into a vector space called the feature space.
(ii) Linear relations are sought among the images of the data items in the feature space.
(iii) The algorithms are implemented in such a way that the coordinates of the embedded points are not needed, only their pairwise inner products.
(iv) The pairwise inner products can be computed efficiently directly from the original data items using a kernel function.

These stages are illustrated in Figure 2.1.

These four observations will imply that, despite restricting ourselves to algorithms that optimise linear functions, our approach will enable the development of a rich toolbox of efficient and well-founded methods for discovering nonlinear relations in data through the use of nonlinear embedding mappings. Before delving into an extended example we give a general definition of a linear pattern.

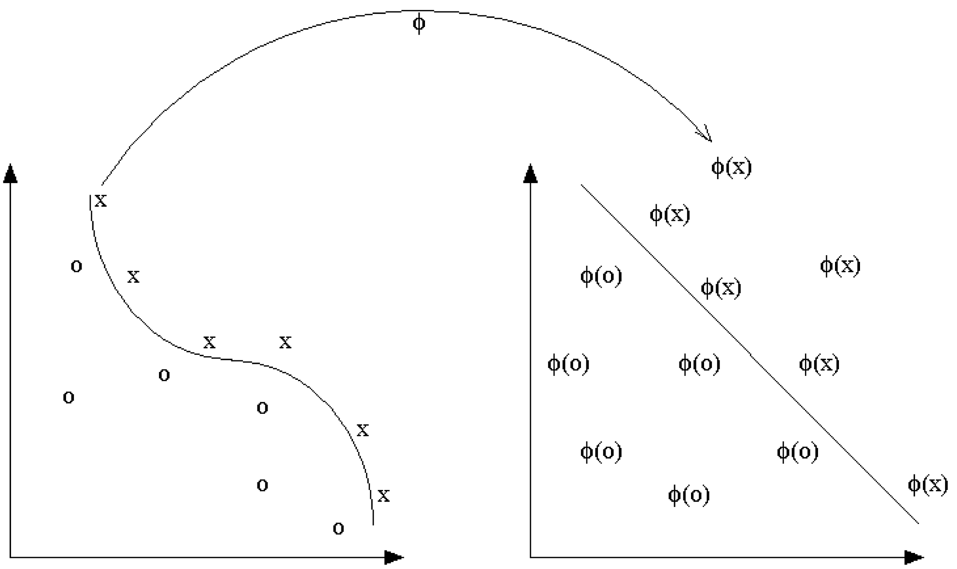

Fig. 2.1. The function ϕ embeds the data into a feature space where the nonlinear pattern now appears linear. The kernel computes inner products in the feature space directly from the inputs.

Definition 2.1 [Linear pattern] A linear pattern is a pattern function drawn from a set of patterns based on a linear function class. ■

2.2 Linear regression in a feature space

2.2.1 Primal linear regression

Consider the problem of finding a homogeneous real-valued linear function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}'\mathbf{x} = \sum_{i=1}^{n} w_i x_i,$$

that best interpolates a given training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ of points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i in $Y \subseteq \mathbb{R}$. Here, we use the notation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ for the n-dimensional input vectors, while $\mathbf{w}'$ denotes the transpose of the vector $\mathbf{w} \in \mathbb{R}^n$. This is naturally one of the simplest relations one might find in the source $X \times Y$, namely a linear function g of the features $\mathbf{x}$ matching the corresponding label y, creating a pattern function that should be approximately equal to zero

$$f((\mathbf{x}, y)) = |y - g(\mathbf{x})| = |y - \langle \mathbf{w}, \mathbf{x} \rangle| \approx 0.$$

This task is also known as *linear interpolation.* Geometrically it corresponds to fitting a hyperplane through the given n-dimensional points. Figure 2.2 shows an example for $n = 1$.

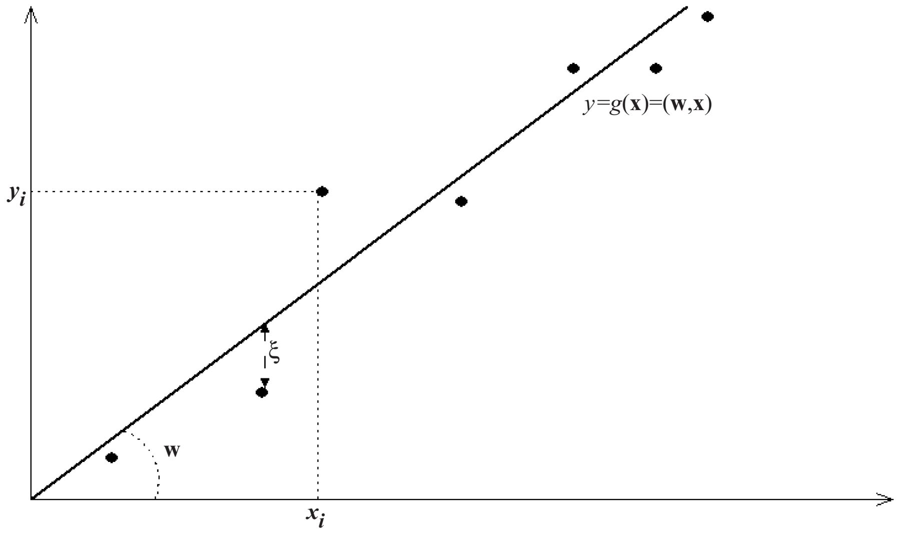

Fig. 2.2. A one-dimensional linear regression problem.

In the exact case, when the data has been generated in the form $(\mathbf{x}, g(\mathbf{x}))$, where $g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ and there are exactly $\ell = n$ linearly independent points, it is possible to find the parameters $\mathbf{w}$ by solving the system of linear equations

$$\mathbf{X}\mathbf{w} = \mathbf{y},$$

where we have used $\mathbf{X}$ to denote the matrix whose rows are the row vectors $\mathbf{x}_1', \ldots, \mathbf{x}_\ell'$ and $\mathbf{y}$ to denote the vector $(y_1, \ldots, y_\ell)'$.

Remark 2.2 [Row versus column vectors] Note that our inputs are column vectors but they are stored in the matrix $\mathbf{X}$ as row vectors. We adopt this convention to be consistent with the typical representation of data in an input file and in our Matlab code, while preserving the standard vector representation. ■

If there are less points than dimensions, there are many possible $\mathbf{w}$ that describe the data exactly, and a criterion is needed to choose between them. In this situation we will favour the vector $\mathbf{w}$ with minimum norm. If there are more points than dimensions and there is noise in the generation process, then we should not expect there to be an exact pattern, so that an approximation criterion is needed. In this situation we will select the pattern with smallest error. In general, if we deal with noisy small datasets, a mix of the two strategies is needed: find a vector $\mathbf{w}$ that has both small norm and small error.

The distance shown as ξ in the figure is the error of the linear function on the particular training example, $\xi = (y - g(\mathbf{x}))$. This value is the output of the putative pattern function

$$f((\mathbf{x}, y)) = |y - g(\mathbf{x})| = |\xi|.$$

We would like to find a function for which all of these training errors are small. The sum of the squares of these errors is the most commonly chosen measure of the collective discrepancy between the training data and a particular function

$$\mathcal{L}(g, S) = \mathcal{L}(\mathbf{w}, S) = \sum_{i=1}^{\ell}(y_i - g(\mathbf{x}_i))^2 = \sum_{i=1}^{\ell}\xi_i^2 = \sum_{i=1}^{\ell}\mathcal{L}((\mathbf{x}_i, y_i), g),$$

where we have used the same notation $\mathcal{L}((\mathbf{x}_i, y_i), g) = \xi_i^2$ to denote the squared error or *loss* of g on example $(\mathbf{x}_i, y_i)$ and $\mathcal{L}(f, S)$ to denote the collective loss of a function f on the training set S. The learning problem now becomes that of choosing the vector $\mathbf{w} \in W$ that minimises the collective loss. This is a well-studied problem that is applied in virtually every discipline. It was introduced by Gauss and is known as *least squares approximation.*

Using the notation above, the vector of output discrepancies can be written as

$$\boldsymbol{\xi} = \mathbf{y} - \mathbf{X}\mathbf{w}.$$

Hence, the loss function can be written as

$$\mathcal{L}(\mathbf{w}, S) = \|\boldsymbol{\xi}\|_2^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})'(\mathbf{y} - \mathbf{X}\mathbf{w}). \tag{2.1}$$

Note that we again use $\mathbf{X}'$ to denote the transpose of $\mathbf{X}$. We can seek the optimal $\mathbf{w}$ by taking the derivatives of the loss with respect to the parameters $\mathbf{w}$ and setting them equal to the zero vector

$$\frac{\partial\mathcal{L}(\mathbf{w}, S)}{\partial\mathbf{w}} = -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\mathbf{w} = \mathbf{0},$$

hence obtaining the so-called ‘normal equations’

$$\mathbf{X}'\mathbf{X}\mathbf{w} = \mathbf{X}'\mathbf{y}. \tag{2.2}$$

If the inverse of $\mathbf{X}'\mathbf{X}$ exists, the solution of the least squares problem can be expressed as

$$\mathbf{w} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}.$$

Hence, to minimise the squared loss of a linear interpolant, one needs to maintain as many parameters as dimensions, while solving an $n \times n$ system of linear equations is an operation that has cubic cost in n.

This cost refers to the number of operations and is generally expressed as a complexity of $O\left(n^3\right)$, meaning that the number of operations $t\left(n\right)$ required for the computation can be bounded by

$$t\left(n\right) \leq Cn^3$$

for some constant C.

The predicted output on a new data point can now be computed using the prediction function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle.$$

Remark 2.3 [Dual representation] Notice that if the inverse of $\mathbf{X}'\mathbf{X}$ exists we can express $\mathbf{w}$ in the following way

$$\mathbf{w} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = \mathbf{X}'\mathbf{X}(\mathbf{X}'\mathbf{X})^{-2}\mathbf{X}'\mathbf{y} = \mathbf{X}'\boldsymbol{\alpha},$$

making it a linear combination of the training points, $\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i$. ■

Remark 2.4 [Pseudo-inverse] If $\mathbf{X}'\mathbf{X}$ is singular, the pseudo-inverse can be used. This finds the $\mathbf{w}$ that satisfies the equation (2.2) with minimal norm. Alternatively we can trade off the size of the norm against the loss. This is the approach known as ridge regression that we will describe below. ■

As mentioned Remark 2.4 there are situations where fitting the data exactly may not be possible. Either there is not enough data to ensure that the matrix $\mathbf{X}'\mathbf{X}$ is invertible, or there may be noise in the data making it unwise to try to match the target output exactly. We described this situation in Chapter 1 as seeking an approximate pattern with algorithms that are robust. Problems that suffer from this difficulty are known as *ill-conditioned*, since there is not enough information in the data to precisely specify the solution. In these situations an approach that is frequently adopted is to restrict the choice of functions in some way. Such a restriction or bias is referred to as *regularisation*. Perhaps the simplest regulariser is to favour functions that have small norms. For the case of least squares regression, this gives the well-known optimisation criterion of ridge regression.

Computation 2.5 [Ridge regression] Ridge regression corresponds to solving the optimisation

$$\min_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w},S) = \min_{\mathbf{w}} \lambda \left\|\mathbf{w}\right\|^2 + \sum_{i=1}^{\ell}(y_i - g(\mathbf{x}_i))^2, \tag{2.3}$$

where λ is a positive number that defines the relative trade-off between norm and loss and hence controls the degree of regularisation. The learning problem is reduced to solving an optimisation problem over $\mathbb{R}^n$. ■

2.2.2 Ridge regression: primal and dual

Again taking the derivative of the cost function with respect to the parameters we obtain the equations

$$\mathbf{X}'\mathbf{X}\mathbf{w}+\lambda\mathbf{w} = \left(\mathbf{X}'\mathbf{X}+\lambda\mathbf{I}_n\right)\mathbf{w} = \mathbf{X}'\mathbf{y}, \tag{2.4}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix. In this case the matrix $(\mathbf{X}'\mathbf{X}+\lambda\mathbf{I}_n)$ is always invertible if $\lambda > 0$, so that the solution is given by

$$\mathbf{w} = \left(\mathbf{X}'\mathbf{X}+\lambda\mathbf{I}_n\right)^{-1}\mathbf{X}'\mathbf{y}. \tag{2.5}$$

Solving this equation for $\mathbf{w}$ involves solving a system of linear equations with n unknowns and n equations. The complexity of this task is $O(n^3)$. The resulting prediction function is given by

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{y}'\mathbf{X}\left(\mathbf{X}'\mathbf{X}+\lambda\mathbf{I}_n\right)^{-1}\mathbf{x}.$$

Alternatively, we can rewrite equation (2.4) in terms of $\mathbf{w}$ (similarly to Remark 2.3) to obtain

$$\mathbf{w} = \lambda^{-1}\mathbf{X}'\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) = \mathbf{X}'\boldsymbol{\alpha},$$

showing that again $\mathbf{w}$ can be written as a linear combination of the training points, $\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i$ with $\boldsymbol{\alpha} = \lambda^{-1}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)$. Hence, we have

$$\begin{aligned} \boldsymbol{\alpha} &= \lambda^{-1}\left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) \\ &\Rightarrow \lambda\boldsymbol{\alpha} = \left(\mathbf{y} - \mathbf{X}\mathbf{X}'\boldsymbol{\alpha}\right) \\ &\Rightarrow \left(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_\ell\right)\boldsymbol{\alpha} = \mathbf{y} \\ &\Rightarrow \boldsymbol{\alpha} = \left(\mathbf{G} + \boldsymbol{\lambda}\mathbf{I}_\ell\right)^{-1}\mathbf{y}, \end{aligned} \tag{2.6}$$

where $\mathbf{G} = \mathbf{X}\mathbf{X}'$ or, component-wise, $\mathbf{G}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Solving for $\boldsymbol{\alpha}$ involves solving ℓ linear equations with ℓ unknowns, a task of complexity $O(\ell^3)$. The resulting prediction function is given by

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \left\langle \sum_{i=1}^{\ell} \alpha_i \mathbf{x}_i, \mathbf{x} \right\rangle = \sum_{i=1}^{\ell} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle = \mathbf{y}' (\mathbf{G} + \lambda \mathbf{I}_\ell)^{-1} \mathbf{k},$$

where $k_i = \langle \mathbf{x}_i, \mathbf{x} \rangle$. We have thus found two distinct methods for solving the ridge regression optimisation of equation (2.3). The first given in equation (2.5) computes the weight vector explicitly and is known as the *primal solution*, while equation (2.6) gives the solution as a linear combination of the training examples and is known as the *dual solution*. The parameters $\boldsymbol{\alpha}$ are known as the *dual variables*.

The crucial observation about the dual solution of equation (2.6) is that the information from the training examples is given by the inner products between pairs of training points in the matrix $\mathbf{G} = \mathbf{X}\mathbf{X}'$. Similarly, the information about a novel example $\mathbf{x}$ required by the predictive function is just the inner products between the training points and the new example $\mathbf{x}$.

The matrix $\mathbf{G}$ is referred to as the *Gram matrix*. The Gram matrix and the matrix $(\mathbf{G} + \lambda \mathbf{I}_\ell)$ have dimensions $\ell \times \ell$. If the dimension n of the feature space is larger than the number ℓ of training examples, it becomes more efficient to solve equation (2.6) rather than the primal equation (2.5) involving the matrix $(\mathbf{X}'\mathbf{X}+\lambda\mathbf{I}_n)$ of dimension $n \times n$. Evaluation of the predictive function in this setting is, however, always more costly since the primal involves $O(n)$ operations, while the complexity of the dual is $O(n\ell)$. Despite this we will later see that the dual solution can offer enormous advantages.

Hence one of the key findings of this section is that the ridge regression algorithm can be solved in a form that only requires inner products between data points.

Remark 2.6 [Primal-dual] The primal-dual dynamic described above recurs throughout the book. It also plays an important role in optimisation, text analysis, and so on. ■

Remark 2.7 [Statistical stability] Though we have addressed the question of efficiency of the ridge regression algorithm, we have not attempted to analyse explicitly its robustness or stability. These issues will be considered in later chapters. ■

2.2.3 Kernel-defined nonlinear feature mappings

The ridge regression method presented in the previous subsection addresses the problem of identifying linear relations between one selected variable and the remaining features, where the relation is assumed to be functional. The resulting predictive function can be used to estimate the value of the selected variable given the values of the other features. Often, however, the relations that are sought are nonlinear, that is the selected variable can only be accurately estimated as a nonlinear function of the remaining features. Following our overall strategy we will map the remaining features of the data into a new feature space in such a way that the sought relations can be represented in a linear form and hence the ridge regression algorithm described above will be able to detect them.

We will consider an embedding map

$$\boldsymbol{\phi} : \mathbf{x} \in \mathbb{R}^n \longmapsto \boldsymbol{\phi}(\mathbf{x}) \in F \subseteq \mathbb{R}^N.$$

The choice of the map $\boldsymbol{\phi}$ aims to convert the nonlinear relations into linear ones. Hence, the map reflects our expectations about the relation $y = g(\mathbf{x})$ to be learned. The effect of $\boldsymbol{\phi}$ is to recode our dataset S as $\widehat{S} = \{(\boldsymbol{\phi}(\mathbf{x}_1), y_1), ..., (\boldsymbol{\phi}(\mathbf{x}_\ell), y_\ell)\}$. We can now proceed as above looking for a relation of the form

$$f((\mathbf{x}, y)) = |y - g(\mathbf{x})| = |y - \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle| = |\xi|.$$

Although the primal method could be used, problems will arise if N is very large making the solution of the $N \times N$ system of equation (2.5) very expensive. If, on the other hand, we consider the dual solution, we have shown that all the information the algorithm needs is the inner products between data points $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle$ in the feature space F. In particular the predictive function $g(\mathbf{x}) = \mathbf{y}' (\mathbf{G} + \lambda \mathbf{I}_\ell)^{-1} \mathbf{k}$ involves the Gram matrix $\mathbf{G} = \mathbf{X}\mathbf{X}'$ with entries

$$\mathbf{G}_{ij} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle, \tag{2.7}$$

where the rows of $\mathbf{X}$ are now the feature vectors $\boldsymbol{\phi}(\mathbf{x}_1)', \ldots, \boldsymbol{\phi}(\mathbf{x}_\ell)'$, and the vector $\mathbf{k}$ contains the values

$$k_i = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}) \rangle. \tag{2.8}$$

When the value of N is very large, it is worth taking advantage of the dual solution to avoid solving the large $N \times N$ system. Making the optimistic assumption that the complexity of evaluating ϕ is $O(N)$, the complexity of evaluating the inner products of equations (2.7) and (2.8) is still $O(N)$

making the overall complexity of computing the vector $\boldsymbol{\alpha}$ equal to

$$O(\ell^3 + \ell^2 N), \tag{2.9}$$

while that of evaluating g on a new example is

$$O(\ell N). \tag{2.10}$$

We have seen that in the dual solution we make use of inner products in the feature space. In the above analysis we assumed that the complexity of evaluating each inner product was proportional to the dimension of the feature space. The inner products can, however, sometimes be computed more efficiently as a direct function of the input features, without explicitly computing the mapping $\boldsymbol{\phi}$. In other words the feature-vector representation step can be by-passed. A function that performs this direct computation is known as a *kernel function*.

Definition 2.8 [Kernel function] A *kernel* is a function κ that for all $\mathbf{x}, \mathbf{z} \in X$ satisfies

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

where ϕ is a mapping from X to an (inner product) feature space F

$$\boldsymbol{\phi} : \mathbf{x} \longmapsto \boldsymbol{\phi}(\mathbf{x}) \in F.$$

■

Kernel functions will be an important theme throughout this book. We will examine their properties, the algorithms that can take advantage of them, and their use in general pattern analysis applications. We will see that they make possible the use of feature spaces with an exponential or even infinite number of dimensions, something that would seem impossible if we wish to satisfy the efficiency requirements given in Chapter 1. Our aim in this chapter is to give examples to illustrate the key ideas underlying the proposed approach. We therefore now give an example of a kernel function whose complexity is less than the dimension of its corresponding feature space F, hence demonstrating that the complexity of applying ridge regression using the kernel improves on the estimates given in expressions (2.9) and (2.10) involving the dimension N of F.

Example 2.9 Consider a two-dimensional input space $X \subseteq \mathbb{R}^2$ together with the feature map

$$\boldsymbol{\phi} :\ \mathbf{x} = (x_1, x_2) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \in F = \mathbb{R}^3.$$

The hypothesis space of linear functions in F would then be

$$g(\mathbf{x}) = w_{11}x_1^2 + w_{22}x_2^2 + w_{12}\sqrt{2}x_1x_2$$

The feature map takes the data from a two-dimensional to a three-dimensional space in a way that linear relations in the feature space correspond to quadratic relations in the input space. The composition of the feature map with the inner product in the feature space can be evaluated as follows

$$\begin{aligned}\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z})\rangle &= \left\langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2)\right\rangle \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 = \langle \mathbf{x}, \mathbf{z}\rangle^2 .\end{aligned}$$

Hence, the function

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z}\rangle^2$$

is a kernel function with F its corresponding feature space. This means that we can compute the inner product between the projections of two points into the feature space without explicitly evaluating their coordinates. Note that the same kernel computes the inner product corresponding to the four-dimensional feature map

$$\boldsymbol{\phi} : \mathbf{x} = (x_1, x_2) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \in F = \mathbb{R}^4,$$

showing that the feature space is not uniquely determined by the kernel function.

Example 2.10 The previous example can readily be generalised to higher dimensional input spaces. Consider an n-dimensional space $X \subseteq \mathbb{R}^n$; then the function

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z}\rangle^2$$

is a kernel function corresponding to the feature map

$$\boldsymbol{\phi} : \mathbf{x} \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (x_ix_j)_{i,j=1}^n \in F = \mathbb{R}^{n^2},$$

since we have that

$$\begin{aligned}\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z})\rangle &= \left\langle (x_ix_j)_{i,j=1}^n, (z_iz_j)_{i,j=1}^n\right\rangle \\ &= \sum_{i,j=1}^n x_ix_jz_iz_j = \sum_{i=1}^n x_iz_i \sum_{j=1}^n x_jz_j \\ &= \langle \mathbf{x}, \mathbf{z}\rangle^2 .\end{aligned}$$

If we now use this kernel in the dual form of the ridge regression algorithm, the complexity of the computation of the vector $\boldsymbol{\alpha}$ is $O(n\ell^2 + \ell^3)$ as opposed to a complexity of $O(n^2\ell^2 + \ell^3)$ predicted in the expressions (2.9) and (2.10). If we were analysing 1000 images each with 256 pixels this would roughly correspond to a 50-fold reduction in the computation time. Similarly, the time to evaluate the predictive function would be reduced by a factor of 256.

The example illustrates our second key finding that kernel functions can improve the computational complexity of computing inner products in a feature space, hence rendering algorithms efficient in very high-dimensional feature spaces.

The example of dual ridge regression and the polynomial kernel of degree 2 have demonstrated how a linear pattern analysis algorithm can be efficiently applied in a high-dimensional feature space by using an appropriate kernel function together with the dual form of the algorithm. In the next remark we emphasise an observation arising from this example as it provides the basis for the approach adopted in this book.

Remark 2.11 [Modularity] There was no need to change the underlying algorithm to accommodate the particular choice of kernel function. Clearly, we could use any suitable kernel for the data being considered. Similarly, if we wish to undertake a different type of pattern analysis we could substitute a different algorithm while retaining the chosen kernel. This illustrates the modularity of the approach that makes it possible to consider the algorithmic design and analysis separately from that of the kernel functions. This modularity will also become apparent in the structure of the book. ■

Hence, some chapters of the book are devoted to the theory and practice of designing kernels for data analysis. Other chapters will be devoted to the development of algorithms for some of the specific data analysis tasks described in Chapter 1.

2.3 Other examples

The previous section illustrated how the kernel methods approach can implement nonlinear regression through the use of a kernel-defined feature space. The aim was to show how the key components of the kernel methods approach fit together in one particular example. In this section we will briefly describe how kernel methods can be used to solve many of the tasks outlined in Chapter 1, before going on to give an overview of the different kernels we will be considering. This will lead naturally to a road map for the rest of the book.

2.3.1 Algorithms

Part II of the book will be concerned with algorithms. Our aim now is to indicate the range of tasks that can be addressed.

Classification Consider now the supervised classification task. Given a set

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)\}$$

of points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with labels y_i from $Y = \{-1, +1\}$, find a prediction function $g(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle - b)$ such that

$$\mathbb{E}[0.5\,|g(\mathbf{x}) - y|]$$

is small, where we will use the convention that $\operatorname{sign}(0) = 1$. Note that the 0.5 is included to make the loss the discrete loss and the value of the expectation the probability that a randomly drawn example $\mathbf{x}$ is misclassified by g.

Since g is a thresholded linear function, this can be regarded as learning a hyperplane defined by the equation $\langle \mathbf{w}, \mathbf{x} \rangle = b$ separating the data according to their labels, see Figure 2.3. Recall that a hyperplaneis an affine subspace of dimension $n - 1$ which divides the space into two half spaces corresponding to the inputs of the two distinct classes. For example in Figure 2.3 the hyperplane is the dark line, with the positive region above and the negative region below. The vector $\mathbf{w}$ defines a direction perpendicular to the hyperplane, while varying the value of b moves the hyperplane parallel to itself. A representation involving $n + 1$ free parameters therefore can describe all possible hyperplanes in $\mathbb{R}^n$.

Both statisticians and neural network researchers have frequently used this simple kind of classifier, calling them respectively *linear discriminants* and *perceptrons*. The theory of linear discriminants was developed by Fisher in 1936, while neural network researchers studied perceptrons in the early 1960s, mainly due to the work of Rosenblatt. We will refer to the quantity $\mathbf{w}$ as the *weight vector*, a term borrowed from the neural networks literature.

There are many different algorithms for selecting the weight vector $\mathbf{w}$, many of which can be implemented in dual form. We will describe the perceptron algorithm and support vector machine algorithms in Chapter 7.

Fig. 2.3. A linear function for classification creates a separating hyperplane.

Principal components analysis Detecting regularities in an unlabelled set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ of points from $X \subseteq \mathbb{R}^n$ is referred to as unsupervised learning. As mentioned in Chapter 1, one such task is finding a low-dimensional representation of the data such that the expected residual is as small as possible. Relations between features are important because they reduce the effective dimensionality of the data, causing it to lie on a lower dimensional surface. This may make it possible to recode the data in a more efficient way using fewer coordinates. The aim is to find a smaller set of variables defined by functions of the original features in such a way that the data can be approximately reconstructed from the new coordinates.

Despite the difficulties encountered if more general functions are considered, a good understanding exists of the special case when the relations are assumed to be linear. This subcase is attractive because it leads to analytical solutions and simple computations. For linear functions the problem is equivalent to projecting the data onto a lower-dimensional linear subspace in such a way that the distance between a vector and its projection is not too large. The problem of minimising the average squared distance between vectors and their projections is equivalent to projecting the data onto the space spanned by the first k eigenvectors of the matrix $\mathbf{X}'\mathbf{X}$

$$\mathbf{X}'\mathbf{X}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

and hence the coordinates of a new vector $\mathbf{x}$ in the new space can be obtained by considering its projection onto the eigenvectors $\langle \mathbf{x}, \mathbf{v}_i \rangle$, $i = 1, \ldots, k$. This technique is known as principal components analysis (PCA).

The algorithm can be rendered nonlinear by first embedding the data into a feature space and then consider projections in that space. Once again we will see that kernels can be used to define the feature space, since the algorithm can be rewritten in a form that only requires inner products between inputs. Hence, we can detect nonlinear relations between variables in the data by embedding the data into a kernel-induced feature space, where linear relations can be found by means of PCA in that space. This approach is known as *kernel PCA* and will be described in detail in Chapter 6.

Remark 2.12 [Low-rank approximation] Of course some information about linear relations in the data is already implicit in the rank of the data matrix. The rank corresponds to the number of non-zero eigenvalues of the covariance matrix and is the dimensionality of the subspace in which the data lie. The rank can also be computed using only inner products, since the eigenvalues of the inner product matrix are equal to those of the covariance matrix. We can think of PCA as finding a low-rank approximation, where the quality of the approximation depends on how close the data is to lying in a subspace of the given dimensionality. ■

Clustering Finally, we mention finding clusters in a training set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ of points from $X \subseteq \mathbb{R}^n$. One method of defining clusters is to identify a fixed number of centres or prototypes and assign points to the cluster defined by the closest centre. Identifying clusters by a set of prototypes divides the space into what is known as a Voronoi partitioning.

The aim is to minimise the expected squared distance of a point from its cluster centre. If we fix the number of centres to be k, a classic procedure is known as k-means and is a widely used heuristic for clustering data. The k-means procedure must have some method for measuring the distance between two points. Once again this distance can always be computed using only inner product information through the equality

$$\|\mathbf{x} - \mathbf{z}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{z}, \mathbf{z} \rangle - 2\langle \mathbf{x}, \mathbf{z} \rangle.$$

This distance, together with a dual representation of the mean of a given set of points, implies the k-means procedure can be implemented in a kernel-defined feature space. This procedure is not, however, a typical example of a kernel method since it fails to meet our requirement of efficiency. This is because the optimisation criterion is not convex and hence we cannot guarantee that the procedure will converge to the optimal arrangement. A number of clustering methods will be described in Chapter 8.

2.3.2 Kernels

Part III of the book will be devoted to the design of a whole range of kernel functions. The approach we have outlined in this chapter shows how a number of useful tasks can be accomplished in high-dimensional feature spaces defined implicitly by a kernel function. So far we have only seen how to construct very simple polynomial kernels. Clearly, for the approach to be useful, we would like to have a range of potential kernels together with machinery to tailor their construction to the specifics of a given data domain. If the inputs are elements of a vector space such as $\mathbb{R}^n$ there is a natural inner product that is referred to as the *linear kernel* by analogy with the polynomial construction. Using this kernel corresponds to running the original algorithm in the input space. As we have seen above, at the cost of a few extra operations, the polynomial construction can convert the linear kernel into an inner product in a vastly expanded feature space. This example illustrates a general principle we will develop by showing how more complex kernels can be created from simpler ones in a number of different ways. Kernels can even be constructed that correspond to infinite-dimensional feature spaces at the cost of only a few extra operations in the kernel evaluations.

An example of creating a new kernel from an existing one is provided by normalising a kernel. Given a kernel $\kappa(\mathbf{x}, \mathbf{z})$ that corresponds to the feature mapping $\boldsymbol{\phi}$, the *normalised kernel* $\kappa(\mathbf{x}, \mathbf{z})$ corresponds to the feature map

$$\mathbf{x} \longmapsto \boldsymbol{\phi}(\mathbf{x}) \longmapsto \frac{\boldsymbol{\phi}(\mathbf{x})}{\|\boldsymbol{\phi}(\mathbf{x})\|}.$$

Hence, we will show in Chapter 5 that we can express the kernel $\hat{\kappa}$ in terms of κ as follows

$$\hat{\kappa}(\mathbf{x}, \mathbf{z}) = \left\langle \frac{\boldsymbol{\phi}(\mathbf{x})}{\|\boldsymbol{\phi}(\mathbf{x})\|}, \frac{\boldsymbol{\phi}(\mathbf{z})}{\|\boldsymbol{\phi}(\mathbf{z})\|} \right\rangle = \frac{\kappa(\mathbf{x}, \mathbf{z})}{\sqrt{\kappa(\mathbf{x}, \mathbf{x})\kappa(\mathbf{z}, \mathbf{z})}}.$$

These constructions will not, however, in themselves extend the range of data types that can be processed. We will therefore also develop kernels that correspond to mapping inputs that are not vectors into an appropriate feature space. As an example, consider the input space consisting of all subsets of a fixed set D. Consider the kernel function of two subsets A_1 and A_2 of D defined by

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

that is the number of common subsets A_1 and A_2 share. This kernel corresponds to a feature map $\boldsymbol{\phi}$ to the vector space of dimension $2^{|D|}$ indexed by all subsets of D, where the image of a set A is the vector with

$$\boldsymbol{\phi}\left(A\right)_U = \begin{cases} 1; & \text{if } U \subseteq A, \\ 0; & \text{otherwise.} \end{cases}$$

This example is defined over a general set and yet we have seen that it fulfills the conditions for being a valid kernel, namely that it corresponds to an inner product in a feature space. Developing this approach, we will show how kernels can be constructed from different types of input spaces in a way that reflects their structure even though they are not in themselves vector spaces. These kernels will be needed for many important applications such as text analysis and bioinformatics. In fact, the range of valid kernels is very large: some are given in closed form; others can only be computed by means of a recursion or other algorithm; in some cases the actual feature mapping corresponding to a given kernel function is not known, only a guarantee that the data can be embedded in some feature space that gives rise to the chosen kernel. In short, provided the function can be evaluated efficiently and it corresponds to computing the inner product of suitable images of its two arguments, it constitutes a potentially useful kernel.

Selecting the best kernel from among this extensive range of possibilities becomes the most critical stage in applying kernel-based algorithms in practice. The selection of the kernel can be shown to correspond in a very tight sense to the encoding of our prior knowledge about the data and the types of patterns we can expect to identify. This relationship will be explored by examining how kernels can be derived from probabilistic models of the process generating the data.

In Chapter 3 the techniques for creating and adapting kernels will be presented, hence laying the foundations for the later examples of practical kernel based applications. It is possible to construct complex kernel functions from simpler kernels, from explicit features, from similarity measures or from other types of prior knowledge. In short, we will see how it will be possible to treat the kernel part of the algorithm in a modular fashion, constructing it from simple components and then modifying it by means of a set of well-defined operations.

2.4 The modularity of kernel methods

The procedures outlined in the previous sections will be generalised and analysed in subsequent chapters, but a consistent trend will emerge. An algorithmic procedure is adapted to use only inner products between inputs. The method can then be combined with a kernel function that calculates the inner product between the images of two inputs in a feature space, hence making it possible to implement the algorithm in a high-dimensional space.

The modularity of kernel methods shows itself in the reusability of the learning algorithm. The same algorithm can work with any kernel and hence for any data domain. The kernel component is data specific, but can be combined with different algorithms to solve the full range of tasks that we will consider. All this leads to a very natural and elegant approach to learning systems design, where modules are combined together to obtain complex learning systems. Figure 2.4 shows the stages involved in the implementation of kernel pattern analysis. The data is processed using a kernel to create a kernel matrix, which in turn is processed by a pattern analysis algorithm to producce a pattern function. This function is used to process unseen examples. This book will follow a corresponding modular structure

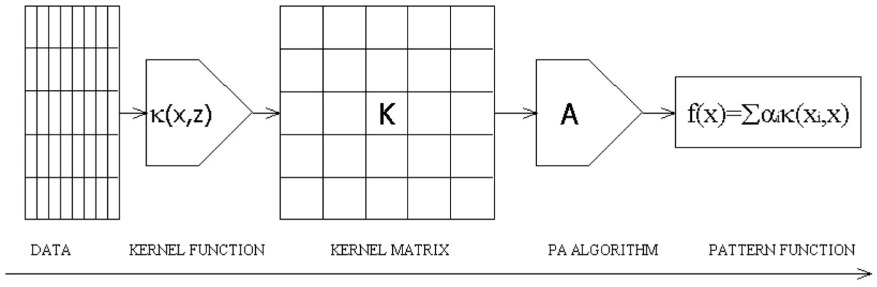

Fig. 2.4. The stages involved in the application of kernel methods.

developing each of the aspects of the approach independently.

From a computational point of view kernel methods have two important properties. First of all, they enable access to very high-dimensional and correspondingly flexible feature spaces at low computational cost both in space and time, and yet secondly, despite the complexity of the resulting function classes, virtually all of the algorithms presented in this book solve convex optimisation problems and hence do not suffer from local minima. In Chapter 7 we will see that optimisation theory also confers other advantages on the resulting algorithms. In particular duality will become a central theme throughout this book, arising within optimisation, text representation, and algorithm design.

Finally, the algorithms presented in this book have a firm statistical foundation that ensures they remain resistant to overfitting. Chapter 4 will give a unified analysis that makes it possible to view the algorithms as special cases of a single framework for analysing generalisation.

2.5 Roadmap of the book

The first two chapters of the book have provided the motivation for pattern analysis tasks and an overview of the kernel methods approach to learning systems design. We have described how at the top level they involve a two-stage process: the data is implicitly embedded into a feature space through the use of a kernel function, and subsequently linear patterns are sought in the feature space using algorithms expressed in a dual form. The resulting systems are modular: any kernel can be combined with any algorithm and vice versa. The structure of the book reflects that modularity, addressing in three main parts general design principles, specific algorithms and specific kernels.

Part I covers **foundations** and presents the general principles and properties of kernel functions and kernel-based algorithms. Chapter 3 presents the theory of kernel functions including their characterisations and properties. It covers methods for combining kernels and for adapting them in order to modify the geometry of the feature space. The chapter lays the groundwork necessary for the introduction of specific examples of kernels in Part III. Chapter 4 develops the framework for understanding how their statistical stability can be controlled. Again it sets the scene for Part II, where specific algorithms for dimension reduction, novelty-detection, classification, ranking, clustering, and regression are examined.

Part II develops specific **algorithms**. Chapter 5 starts to develop the tools for analysing data in a kernel-defined feature space. After covering a number of basic techniques, it shows how they can be used to create a simple novelty-detection algorithm. Further analysis of the structure of the data in the feature space including implementation of Gram–Schmidt orthonormalisation, leads eventually to a dual version of the Fisher discriminant. Chapter 6 is devoted to discovering patterns using eigenanalysis. The techniques developed include principal components analysis, maximal covariance, and canonical correlation analysis. The application of the patterns in classification leads to an alternative formulation of the Fisher discriminant, while their use in regression gives rise to the partial least squares algorithm. Chapter 7 considers algorithms resulting from optimisation problems and includes sophisticated novelty detectors, the support vector machine, ridge regression, and support vector regression. On-line algorithms for classification and regression are also introduced. Finally, Chapter 8 considers ranking and shows how both batch and on-line kernel based algorithms can be created to solve this task. It then considers clustering in kernel-defined feature spaces showing how the classical k-means algorithm can be implemented in such feature spaces as well as spectral clustering methods. Finally, the problem of data visualisation is formalised and solved also using spectral methods. Appendix C contains an index of the pattern analysis methods covered in Part II.

Part III is concerned with **kernels.** Chapter 9 develops a number of techniques for creating kernels leading to the introduction of ANOVA kernels, kernels defined over graphs, kernels on sets and randomised kernels. Chapter 10 considers kernels based on the vector space model of text, with emphasis on the refinements aimed at taking account of the semantics. Chapter 11 treats kernels for strings of symbols, trees, and general structured data. Finally Chapter 12 examines how kernels can be created from generative models of data either using the probability of co-occurrence or through the Fisher kernel construction. Appendix D contains an index of the kernels described in Part III.

We conclude this roadmap with a specific mention of some of the questions that will be addressed as the themes are developed through the chapters (referenced in brackets):

- Which functions are valid kernels and what are their properties? (Chapter 3)
- How can we guarantee the statistical stability of patterns? (Chapter. 4)
- What algorithms can be kernelised? (Chapter 5, 6, 7 and 8)
- Which problems can be tackled effectively using kernel methods? (Chapters 9 and 10)
- How can we develop kernels attuned to particular applications? (Chapters 10, 11 and 12)

2.6 Summary

- Linear patterns can often be detected efficiently by well-known techniques such as least squares regression.

- Mapping the data via a nonlinear function into a suitable feature space enables the use of the same tools for discovering nonlinear patterns.
- Kernels can make it feasible to use high-dimensional feature spaces by avoiding the explicit computation of the feature mapping.
- The proposed approach is modular in the sense that any kernel will work with any kernel-based algorithm.
- Although linear functions require vector inputs, the use of kernels enables the approach to be applied to other types of data.

2.7 Further reading and advanced topics

The method of least squares for linear regression was (re)invented and made famous by Carl F. Gauss (1777–1855) in the late eighteenth century, by using it to predict the position of an asteroid that had been observed by the astronomer Giuseppe Piazzi for several days and then 'lost'. Before Gauss (who published it in *Theoria motus corporum coelestium*, 1809), it had been independently discovered by Legendre (but published only in 1812, in *Nouvelle Methods pour la determination des orbites des cometes*. It is now a cornerstone of function approximation in all disciplines.

The Widrow–Hoff algorithm is described in [157]. The ridge regression algorithm was published by Hoerl and Kennard [56], and subsequently discovered to be a special case of the regularisation theory of [136] for the solution of ill-posed problems. The dual form of ridge regression was studied by Saunders *et al.*, [113], which gives a formulation similar to that presented here. An equivalent heuristic was widely used in the neural networks literature under the name of weight decay. The combination of ridge regression and kernels has also been explored in the literature of Gaussian Processes [158] and in the literature on regularization networks [105] and RKHSs: [152], see also [129].

The linear Fisher discriminant dates back to 1936 [44], and its use with kernels to the works in [11] and [98], see also [121]. The perceptron algorithm dates back to 1957 by Rosenblatt [109], and its kernelization is a well-known folk algorithm, closely related to the work in [1].

The theory of linear discriminants dates back to the 1930s, when Fisher [44] proposed a procedure for classification of multivariate data by means of a hyperplane. In the field of artificial intelligence, attention was drawn to this problem by the work of Frank Rosenblatt [109], who starting from 1956 introduced the perceptron learning rule. Minsky and Papert's famous book Perceptrons [99] analysed the computational limitations of linear learning machines. The classical book by Duda and Hart (recently reprinted in a new edition [40]) provides a survey of the state-of-the-art in the field. Also useful is [14] which includes a description of a class of generalised learning machines.

The idea of using kernel functions as inner products in a feature space was introduced into machine learning in 1964 by the work of Aizermann, Bravermann and Rozoener [1] on the method of potential functions and this work is mentioned in a footnote of the very popular first edition of Duda and Hart's book on pattern classification [39]. Through this route it came to the attention of the authors of [16], who combined it with large margin hyperplanes, leading to support vector machines and the (re)introduction of the notion of a kernel into the mainstream of the machine learning literature.

The use of kernels for function approximation however dates back to Aronszain [6], as does the development of much of their theory [152].

An early survey of the modern usage of kernel methods in pattern analysis can be found in [20], and more accounts in the books by [32] and [118]. The book [139] describes SVMs, albeit with not much emphasis on kernels. Other books in the area include: [129], [66], [53].

A further realization of the possibilities opened up by the concept of the kernel function is represented by the development of kernel PCA by [119] that will be discussed in Chapter 6. That work made the point that much more complex relations than just linear classifications can be inferred using kernel functions.

Clustering will be discussed in more detail in Chapter 8, so pointers to the relevant literature can be found in Section 8.5.

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