Chapter 5
Support Vector and Kernel Methods

Nello Cristianini* and John Shawe-Taylor**
*University of California at Davis, USA
**Royal Holloway, University of London, UK

Kernel Methods (KM) are a relatively new family of algorithms that presents a series of useful features for pattern analysis in datasets. In recent years, their simplicity, versatility and efficiency have made them a standard tool for practitioners, and a fundamental topic in many data analysis courses. We will outline some of their important features in this Chapter, referring the interested reader to more detailed articles and books for a deeper discussion (see for example [135] and references therein).

KMs combine the simplicity and computational efficiency of linear algorithms, such as the perceptron algorithm or ridge regression, with the flexibility of non-linear systems, such as for example neural networks, and the rigour of statistical approaches such as regularization methods in multivariate statistics. As a result of the special way they represent functions, these algorithms typically reduce the learning step to a convex optimization problem, that can always be solved in polynomial time, avoiding the problem of local minima typical of neural networks, decision trees and other non-linear approaches.

Their foundation in the principles of Statistical Learning Theory make them remarkably resistant to overfitting especially in regimes where other methods are affected by the 'curse of dimensionality'. It is for this reason that they have become popular in bioinformatics and text analysis. Another important feature for applications is that they can naturally accept input data that are not in the form of vectors, such as for example strings, trees and images.

Their characteristically modular design makes them amenable to theoretical analysis but also well suited to a software engineering approach: a general purpose learning module is combined with a data specific 'kernel function' that provides the interface with the data and incorporates domain knowledge. Many

learning modules can be used depending on whether the task is one of classification, regression, clustering, novelty detection, ranking, etc. At the same time many kernel functions have been designed: for protein sequences, for text and hypertext documents, for images, time series, etc. The result is that this method can be used for dealing with rather exotic tasks, such as ranking strings, or clustering graphs, in addition to such classical tasks as classifying vectors. We will delay the definition of a kernel till the next section even though kernels form the core of this contribution.

In this Chapter, we will introduce the main concepts behind this approach to data analysis by discussing some simple examples. We will start with the simplest algorithm and the simplest kernel function, so as to illustrate the basic concepts. Then we will discuss the issue of overfitting, the role of generalization bounds and how they suggest more effective strategies, leading to the Support Vector Machine (SVM) algorithm. In the conclusion, we will briefly discuss other pattern recognition algorithms that exploit the same ideas, for example Principal Components Analysis (PCA), Canonical Correlation Analysis (CCA), and extensions of the SVM algorithm to regression and novelty detection. In this short chapter we err in favour of giving a detailed description of a few standard methods, rather than a superficial survey of the majority.

The problem we will use as an example throughout the chapter is the one of learning a binary classification function using a real-valued function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ in the following way: the input $\mathbf{x} = (x_1, \ldots, x_n)'$ is assigned to the positive class, if $f(\mathbf{x}) \geq 0$, and otherwise to the negative class. We are interested in the case where $f(\mathbf{x})$ is a non-linear function of $\mathbf{x} \in X$, though we will solve the non-linear problem by using linear $f(\mathbf{x})$ in a space that is the image of a non-linear mapping.

We will use X to denote the input space and Y to denote the output domain. Usually we will have $X \subseteq \mathbb{R}^n$, while for binary classification $Y = \{-1, 1\}$ and for regression $Y \subseteq \mathbb{R}$. The *training set* is a collection of *training examples*, which are also called *training data*. It is denoted by

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \subseteq (X \times Y)^m ,$$

where m is the number of examples. We refer to the $\mathbf{x}_i$ as *examples* or *instances* and the y_i as their *labels*. Note that if X is a vector space, the input vectors are column vectors as are the weight vectors. If we wish to form a row vector from $\mathbf{x}_i$ we can take the transpose $\mathbf{x}_i'$. We denote by $\langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}'\mathbf{w} = \sum_i \mathbf{x}_i \mathbf{w}_i$ the inner product between the vectors $\mathbf{x}$ and $\mathbf{w}$.

5.1. Example: Kernel Perceptron

The main idea of Kernel Methods is to first embed the data into a suitable vector space, and then use simple linear methods to detect relevant patterns in the resulting set of points. If the embedding map is non-linear, this enables us to discover non-linear relations using linear algorithms. Hence, we consider a map

from the input space X to a feature space F,

$$\phi : x \in X \longmapsto \phi(x) \in F.$$

Such a mapping of itself will not solve the problem, but it can become very effective if coupled with the following two observations: 1) the information about the relative positions of the data points in the embedding space encoded in the inner products between them is all that is needed by many pattern analysis algorithms; 2) the inner products between the projections of data inputs into high dimensional embedding spaces can often be efficiently computed directly from the inputs using a so-called kernel function. We will illustrate these two points by means of the example of the kernel perceptron.

5.1.1 Primal and Dual Representation

A simple rewriting of the perceptron rule yields an alternative representation for functions and learning algorithms, known as the *dual representation*. In the dual representation, all that is needed are the inner products between data points. There are many linear learning algorithms that can be represented in this way.

Primal Perceptron. As already seen in Chapter 8, the perceptron algorithm learns a binary classification function using a real-valued linear function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ that can be written as

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{w}, \mathbf{x} \rangle + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\operatorname{sgn}(f(\mathbf{x}))$. These parameters must be learned from the data, and are the output of the perceptron algorithm.

A geometric interpretation of this kind of hypothesis is that the input space X is split into two parts by the hyperplane defined by the equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$. A hyperplane is an affine subspace of dimension $n - 1$ which divides the space into two half spaces corresponding to the inputs from the two classes. The vector $\mathbf{w}$ defines a direction perpendicular to the hyperplane, while the value of b determines the distance of the hyperplane from the origin of the coordinate system. It is therefore clear that a representation involving $n + 1$ free parameters is natural, if one wants to represent 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 [451]. We will refer to the quantities $\mathbf{w}$ and b as the *weight vector* and *bias*, terms borrowed from the neural network's literature. Sometimes $-b$ is replaced by θ, a quantity known as the *threshold*.

The simplest iterative algorithm for learning linear classifications is the procedure proposed by Frank Rosenblatt in 1959 for the perceptron [451]. The algorithm created a great deal of interest when it was first introduced. It is an 'on-line' and 'mistake-driven' procedure, which starts with an initial weight vector $\mathbf{w}_0$ (usually $\mathbf{w}_0 = \mathbf{0}$ the all zero vector) and adapts it each time a training point is misclassified by the current weights. The algorithm is shown in Table 5.1.

Table 5.1. The Perceptron Algorithm (primal form)

Given a linearly separable training set S
$\mathbf{w}_0 \leftarrow \mathbf{0}$; $b_0 \leftarrow 0$; $k \leftarrow 0$
$R \leftarrow \max_{1 \leq j \leq m} \|\mathbf{x}_j\|$
repeat
  for $j = 1$ to m
    if $y_j(\langle \mathbf{w}_k, \mathbf{x}_j \rangle + b_k) \leq 0$ then
      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y_j \mathbf{x}_j$
      $b_{k+1} \leftarrow b_k + y_j R^2$
      $k \leftarrow k + 1$
    end if
  end for
until no mistakes made within the *for* loop
return $(\mathbf{w}_k, b_k)$

The algorithm updates the weight vector and bias directly, something that we will refer to as the primal form in contrast to an alternative dual representation which we will introduce below.

This procedure is guaranteed to converge provided there exists a hyperplane that correctly classifies the training data. In this case we say that the data are *linearly separable*. If no such hyperplane exists the data are said to be non-separable.

Dual Representation. It is important to note that the perceptron algorithm works by adding misclassified positive training examples or subtracting misclassified negative examples to an initial arbitrary weight vector. If we take the initial weight vector to be the zero vector, this implies that the final weight vector will be a linear combination of the training points:

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j y_j \mathbf{x}_j,$$

where, since the sign of the coefficient of $\mathbf{x}_j$ is given by the classification y_j, the α_j are positive integral values equal to the number of times misclassification of $\mathbf{x}_j$ has caused the weight to be updated. Points that have caused fewer mistakes will have smaller α_j, whereas difficult points will have large values. Once a sample S has been fixed, the vector α is a representation of the weight vector in different

coordinates, known as the dual representation. This expansion is however not unique: different α can correspond to the same weight vector $\mathbf{w}$. Intuitively, one can also regard α_j as an indication of the information content of the example $\mathbf{x}_j$. In the case of non-separable data, the coefficients of misclassified points grow indefinitely. In the dual representation the decision function can be expressed as follows:

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{m} \alpha_j y_j \mathbf{x}_j, \mathbf{x} \right\rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{m} \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x} \rangle + b\right). \end{aligned} \tag{5.1}$$

Furthermore, the perceptron algorithm can also be implemented entirely in this dual form as shown in Table 5.2.

Table 5.2. The Perceptron Algorithm (dual form)

Given training set S
$\alpha \leftarrow \mathbf{0}$; $b \leftarrow 0$
$R \leftarrow \max_{1 \le i \le m} \|\mathbf{x}_i\|$
repeat
  for $i = 1$ to m
    if $y_i \left(\sum_{j=1}^{m} \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle + b\right) \le 0$ then
      $\alpha_i \leftarrow \alpha_i + 1$
      $b \leftarrow b + y_i R^2$
    end if
  end for
until no mistakes made within the *for* loop
return (α, b) to define function $h(\mathbf{x})$ of equation (5.1)

This alternative formulation of the perceptron algorithm and its decision function has many interesting properties. For example, the fact that the points that are harder to learn have larger α_i can be used to rank the data according to their information content.

It is important to note that the information from the training data only enters the algorithm through inner products of the type $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$: in other words we do not need the coordinates of the points, just the inner products between all pairs. We will see in the next section that, in order to run this algorithm in a feature space, it will be sufficient to compute the value of the inner products between the data in that space, and that these can often be efficiently computed using a special function known as a kernel.

5.1.2 Implicit Embedding via Kernel Functions

Despite the deep theoretical understanding of their statistical and computational properties, the power of linear pattern recognition systems like the Perceptron in real applications is very limited. With some important exceptions, usually real data require non-linear decision boundaries. In practice, more results have been obtained by using less efficient and less well understood non-linear heuristics (such as neural networks or decision trees). In this section we will see how to exploit the dual representation of the perceptron algorithm derived in the previous section in order to obtain a non-linear algorithm. The technique presented here is very general and can be applied to many other algorithms.

Implicit mapping by kernels. In order to transform a linear algorithm into a non-linear one, one can first embed the data into a more powerful feature space F, where a linear machine is sufficiently expressive to capture the relevant relations in the data, by means of a map ϕ:

$$\phi : x \in X \mapsto \phi(x) \in F$$

This task presents both statistical and computational challenges. In this section we address the computational ones, leaving the statistical issues to later in the chapter. We have remarked that one important property of the dual representation of the perceptron obtained above is that the weight vector can be expressed as a linear combination of the training points, so that the decision rule can be evaluated using just inner products between the test point and the training points:

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle + b.$$

Similarly, we have seen that the training (learning the coefficients α_i) can be implemented using just inner products between training points. This makes it possible to use the technique of implicit mapping by replacing every inner product between the feature mapped data points with a kernel function that directly computes the inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle$ as a function of the original input points. In this way it becomes possible to merge the two steps needed to build a non-linear learning machine. We call such a direct computation method a *kernel* function [76, 11].

Definition 5.1. *A* kernel *is a function K, such that for all $\mathbf{x}, \mathbf{z} \in X$*

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle.$$

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

The name 'kernel' is derived from integral operator theory, which underpins much of the theory of the relation between kernels and their corresponding feature spaces. An important consequence of the dual representation is that the

dimension of the feature space need not affect the computation. As the feature vectors are not represented explicitly, the number of operations required to compute the inner product by evaluating the kernel function is not necessarily proportional to the number of features. The use of kernels makes it possible to map the data implicitly into a feature space, train a linear machine in that space and subsequently evaluate it on new examples, potentially side-stepping the computational problems inherent in computing the feature map.

Example 5.1. As a first example of a kernel function, consider two points $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$ in a 2-dimensional space, and the function $K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$:

$$\begin{aligned} \langle \mathbf{x}, \mathbf{z} \rangle^2 &= \langle (x_1, x_2), (z_1, z_2) \rangle^2 \\ &= (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 \\ &= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2), (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle \end{aligned}$$

Hence, this can be regarded as the inner product between two vectors in a feature space with corresponding feature map

$$(x_1, x_2) \longmapsto \phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2),$$

so that

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

In the same way using the easy to compute kernel $K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^d$ gives a feature space of $\binom{n+d-1}{d}$ dimensions, calculating $\phi(x)$ would soon become computationally infeasible for reasonable n and d.

Figure 5.1.2 shows an example of a feature mapping from a two dimensional input space to a two dimensional feature space, where the data cannot be separated by a linear function in the input space, but can be in the feature space. The aim of this section is to show how such mappings can be generated into very high dimensional spaces where linear separation becomes much easier.

One can use kernels even without knowing the feature mapping ϕ associated with them, as long as it is possible to prove that such a mapping exists (i.e. if it is possible to prove that the function being used is really a kernel). An important mathematical characterisation of kernels is given by Mercer's theorem [11], which provides sufficient conditions for a function $K(\mathbf{x}, \mathbf{z})$ to be a valid kernel.

Theorem 5.1. *(Mercer) Let X be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator $T_K : L_2(X) \to L_2(X)$,*

$$(T_K f)(\cdot) = \int_X K(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x},$$

is positive, that is

$$\int_{X \times X} K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0,$$

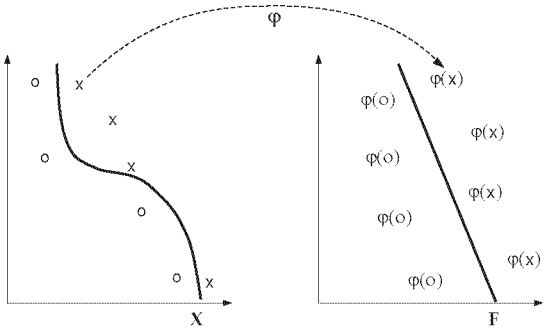

Fig. 5.1. An example of a feature mapping.

for all $f \in L_2(X)$*. Then we can expand* $K(\mathbf{x}, \mathbf{z})$ *in a uniformly convergent series (on* $X \times X$*) in terms of* T_K*s eigen-functions* $\phi_j \in L_2(X)$*, normalized in such a way that* $\|\phi_j\|_{L_2} = 1$*, and positive associated eigenvalues* $\lambda_j \geq 0$*,*

$$K(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z}).$$

The image of a vector $\mathbf{x}$ via the implicit mapping defined by a Mercer kernel is hence $\sum_{j=1}^{\infty} \sqrt{\lambda_j} \phi_j(\mathbf{x})$. Standard kernel functions are the gaussian

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

and the polynomial kernel of degree d:

$$K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^d$$

More information on the rich theory of kernel functions can be found in [135, Chapter 3].

The efficient access to high-dimensional spaces by means of kernels solves the computational problem of dealing with non-linear relations, but leaves open the statistical problem of overfitting. When the number of dimensions exceeds the number of training points, it is always possible to find a linear separation for any partitioning of a non-degenerate dataset. The resulting function is unlikely to generalize well on unseen data. In order to safely use such a flexible class of functions, it is necessary to introduce tools from Statistical Learning Theory.

5.2. Overfitting and Generalization Bounds

When mining for relations in a dataset, we need to pay particular attention to the risk of detecting 'spurious' relations, that are the effect of chance and

do not reflect any underlying property of the data. Such relations can also be defined as 'unsignificant' or unstable, and their characterization is a rather subtle matter. In general, one would like relations that are going to be found also in future datasets generated by the same source, so that they can be used to make predictions.

Statistical Learning Theory is a framework in which it is possible to study the predictive power of relations found in the data under the assumption that the dataset has been generated probabilistically in an independent, identically distributed way. Such model can then be used to design algorithms that are less prone to fitting irrelevant aspects of the data, or 'overfitting'.

We will use Learning Theory to obtain insight into which aspects of a learning algorithm need to be controlled in order to improve its performance on test points. Such indications take the form of upper bounds on the risk of mislabeling a test point (based on some assumptions) that depend on some observable features of the learning system or of the learned rule itself. This section presents a number of results in this sense, that will be used to motivate the algorithms introduced later.

The central assumption is that all the data points (training and test set) are drawn independently from the same (fixed but unknown) distribution. The fundamental quantities that control the generalization power of the system are the training set size and the 'effective capacity' of the hypothesis class used by the system. This quantity is roughly speaking a measure of the flexibility of the algorithm, counting how many different labellings of a dataset the algorithm could successfully separate using a function that is 'equivalent' to the solution obtained.

In order to maintain the focus on the main algorithmic issues, we will simply cite the fundamental bounds that apply to linear classification, referring the interested reader to Chapter 4 of [135] for more details.

We will now define a fundamental quantity in Statistical Learning Theory, the margin of a separating hyperplane with respect to a given labeled set of points. Although technical, the essential idea is that the (geometric) margin is the distance between the separating hyperplane and the nearest point or, equivalently, proportional to the distance between the convex hull of the positive points and the one of the negative points. The functional margin is the smallest value assumed by the linear function on the separated data set. Note that functional and geometric margin are directly proportional.

Definition 5.2. *We define the* (functional) margin of an example $(\mathbf{x}_i, y_i)$ with respect to a hyperplane $\mathbf{w}$ *to be the quantity*

$$\gamma_i = y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b).$$

Note that $\gamma_i > 0$ *implies correct classification of* $(\mathbf{x}_i, y_i)$. *The* (functional) margin distribution of a hyperplane $\mathbf{w}$ with respect to a training set S *is the distribution of the margins of the examples in* S. *We sometimes refer to the minimum of the margin distribution as the* (functional) margin of a hyperplane $\mathbf{w}$ with respect to

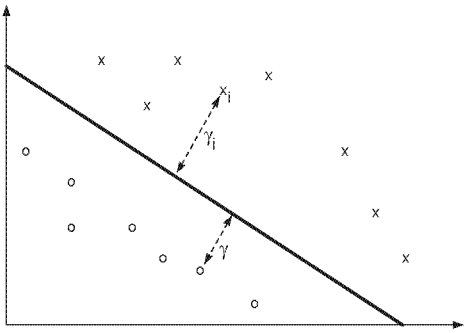

Fig. 5.2. The geometric margin of two points.

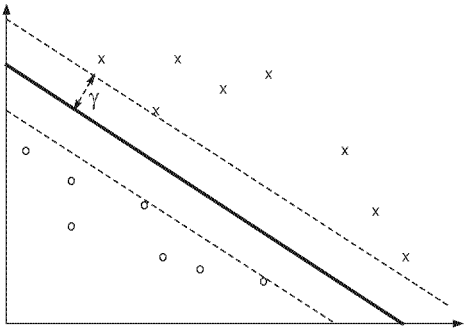

Fig. 5.3. The margin of a training set.

a training set S. *In both definitions if we replace functional margin by* geometric margin *we obtain the equivalent quantity for the normalised linear function* $\left(\frac{1}{\|\mathbf{w}\|}\mathbf{w}, \frac{1}{\|\mathbf{w}\|}b\right)$, *which therefore measures the Euclidean distances of the points from the decision boundary in the input space (see Figure 5.2 for an example). Finally, the* margin of a training set S *is the maximum geometric margin over all hyperplanes. A hyperplane realising this maximum is known as a* maximal margin hyperplane. *The size of its margin will be positive for a linearly separable training set (Figure 5.3).*

The importance of this definition is that the margin (and the related quantities) can be used to obtain information on the predictive power of the solution. The following theorem states that the risk of misclassification of a new point for a separating hyperplane can be bounded by $\varepsilon = O(\frac{R^2}{m\gamma^2})$, where R is the radius of the sphere containing the data, m is the sample size and γ is the (geometric) margin of the separation. In this case we assume the data to be linearly separable.

Theorem 5.2. *([476]) Consider thresholding real-valued linear functions $\mathcal{L}$ with unit weight vectors on an inner product space X and fix $\gamma \in \mathbb{R}^+$. For any*

probability distribution $\mathcal{D}$ on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability $1-\delta$ over m random examples S, any hypothesis $f \in \mathcal{L}$ that has margin $m_S(f) \geq \gamma$ on S has error no more than

$$err_{\mathcal{D}}(f) \leq \varepsilon(m, \mathcal{L}, \delta, \gamma) = \frac{2}{m}\left(\frac{64R^2}{\gamma^2}\log\frac{em\gamma}{8R^2}\log\frac{32m}{\gamma^2} + \log\frac{4}{\delta}\right),$$

provided $m > 2/\varepsilon$ and $64R^2/\gamma^2 < m$.

All the log factors can be neglected at a first reading, being partly the effect of the proof techniques employed. The important *qualitative* aspect of this result is that the dimension of the input space does not appear, indeed the result also applies to infinite dimensional spaces. This type of result is sometimes said to be *dimension free*, as it suggests that the bound may overcome the curse of dimensionality. It is important to note that avoidance of the curse of dimensionality will only be possible if the distribution generating the examples is sufficiently benign and renders the task of identifying the particular target function correspondingly easier. In other words, only if the margin happens to be sufficiently large is this bound useful, while the worst case bound will always depend on the dimensionality as well.

However, we consider here the 'lucky' case in which the margin is large, or -equivalently- the case where the kernel was chosen to match the problem at hand. In such cases the bound gives an assurance that with high probability we will make few errors on randomly chosen test examples. It is in this sense that we can view γ as providing a measure of how benign the distribution is and therefore how much better we can expect to generalise than the worst case.

Theorem 5.2 becomes trivial and hence gives no information for the case where the data are non-separable or noise in the data causes the margin to be very small. The next result discusses a method which can handle these situations by taking a different measure of the margin distribution.

Definition 5.3. *Consider using a class $\mathcal{F}$ of real-valued functions on an input space X for classification by thresholding at 0. We define the* margin slack variable *of an example $(\mathbf{x}_i, y_i) \in X \times \{-1, 1\}$ with respect to a function $f \in \mathcal{F}$ and target margin γ to be the quantity*

$$\xi\left((\mathbf{x}_i, y_i), f, \gamma\right) = \xi_i = \max\left(0,\ \gamma - y_i f(\mathbf{x}_i)\right).$$

Note that $\xi_i > \gamma$ implies incorrect classification of $(\mathbf{x}_i, y_i)$. The margin slack vector *$\xi(S, f, \gamma)$ of a training set*

$$S = \left((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\right)$$

with respect to a function f and target margin γ contains the margin slack variables

$$\xi = \xi(S, f, \gamma) = (\xi_1, \ldots, \xi_m),$$

where the dependence on S, f, and γ is dropped when it is clear from the context.

We can think of the slack variables as measures of noise in the data which has caused individual training points to have smaller or even negative margins. The approach derived from taking account of the slack variables is suitable for handling noisy data.

With this definition, we can give two extensions of the previous result to the noisy case. They will directly motivate the algorithms presented in the following section.

Theorem 5.3. *([477]) Consider thresholding real-valued linear functions $\mathcal{L}$ with unit weight vectors on an inner product space X and fix $\gamma \in \mathbb{R}^+$. There is a constant c, such that for any probability distribution $\mathcal{D}$ on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability $1 - \delta$ over m random examples S, any hypothesis $f \in \mathcal{L}$ has error no more than*

$$err_{\mathcal{D}}(f) \leq \frac{c}{m}\left(\frac{R^2 + \|\xi\|_2^2}{\gamma^2}\log^2 m + \log\frac{1}{\delta}\right),$$

where $\xi = \xi(f, S, \gamma)$ is the margin slack vector with respect to f and γ.

An analogous result can be proven using the 1-norm of the slack variables.

Theorem 5.4. *Consider thresholding real-valued linear functions $\mathcal{L}$ with unit weight vectors on an inner product space X and fix $\gamma \in \mathbb{R}^+$. There is a constant c, such that for any probability distribution $\mathcal{D}$ on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability $1-\delta$ over m random examples S, any hypothesis $f \in \mathcal{L}$ has error no more than*

$$err_{\mathcal{D}}(f) \leq \frac{c}{m}\left(\frac{R^2}{\gamma^2}\log^2 m + \frac{\|\xi\|_1}{\gamma}\log m + \log\frac{1}{\delta}\right),$$

where $\xi = \xi(f, S, \gamma)$ is the margin slack vector with respect to f and γ.

The conclusion to be drawn from the previous theorems is that the generalisation error bound takes into account the amount by which points fail to meet a target margin γ. The bound is in terms of a norm of the slack variable vector suggesting that this quantity should be minimised in order to improve performance. The bound does not rely on the training points being linearly separable and hence can also handle the case when the data are corrupted by noise or the function class cannot capture the full complexity of the decision rule. Optimising the norm of the margin slack vector does not necessarily mean minimising the number of misclassifications. This fact will be important, as we shall see that minimising the number of misclassifications is computationally more demanding than optimising the norms of the margin slack vector.

Optimising the norms of the margin slack vector has a diffuse effect on the margin. For this reason it is referred to as a *soft margin* in contrast to the maximal margin, which depends critically on a small subset of points and is therefore often called a *hard margin*. We will refer to the bound in terms of

the 2-norm of the margin slack vector as the 2-norm soft margin bound, and similarly for the 1-norm soft margin.

The next section will give a number of algorithms that directly minimise the (upper bound on the) risk of making wrong predictions. Importantly, such algorithms reduce to optimizing a convex function.

5.3. Support Vector Machines

We are now ready to derive the main algorithm of this chapter: the celebrated Support Vector Machine, introduced by Vapnik and coworkers in 1992 [76, 525]. The aim of Support Vector classification is to devise a computationally efficient way of learning 'good' separating hyperplanes in a high dimensional feature space, where by 'good' hyperplanes we will understand ones that optimise the generalisation bounds described in the previous section, and by 'computationally efficient' we will mean algorithms able to deal with sample sizes of hundreds of thousands of examples. The generalisation theory gives clear guidance about how to control capacity and hence prevent overfitting by controlling the hyperplane margin measures. Optimisation theory will provide the mathematical techniques necessary to find hyperplanes optimising these measures, and to study their properties.

Different generalisation bounds exist, motivating different algorithms: one can for example optimise the maximal margin, the margin distribution, and other quantities defined later in this section. We will first consider the most common and well-established approach, which reduces the hard margin problem to minimising the norm of the weight vector.

The aim of this algorithm is to find a (generally nonlinear) separation between the data by using kernels, and to ensure that such separation has strong statistical properties aimed at preventing overfitting. This is done efficiently, by finding the global minimum of a convex cost function.

5.3.1 The Maximal Margin Classifier

The simplest type of Support Vector Machine, which was also the first to be introduced, is the so-called maximal margin classifier that optimises the hard margin. It works only for data which are linearly separable in the feature space, and hence cannot be used in many real-world situations. Nonetheless it is the easiest algorithm to understand, and it forms the main building block for the more complex Support Vector Machines. It exhibits the key features that characterise this kind of learning machine, and its description is therefore useful for understanding the more realistic systems introduced later.

Theorem 5.2 bounds the generalisation error of linear machines in terms of the margin $M_S(f)$ of the hypothesis f with respect to the training set S. The dimensionality of the space in which the data are separated does not appear in this theorem. The maximal margin classifier optimises this bound by separating the data with the maximal margin hyperplane, and given that the bound does

not depend on the dimensionality of the space, this separation can be sought in any kernel-induced feature space. The maximal margin classifier forms the strategy of the first Support Vector Machine, namely to separate the data by using the maximal margin hyperplane in an appropriately chosen kernel-induced feature space.

This strategy is implemented by reducing it to a convex optimisation problem: minimising a quadratic function under linear inequality constraints. The cost function depends on the norm of the weight vector, although this may not be immediately obvious.

In order to see it, first we note that in the definition of linear classifiers there is an inherent degree of freedom, as the function associated with the hyperplane $(\mathbf{w}, b)$ does not change if we rescale the hyperplane to $(\lambda\mathbf{w}, \lambda b)$, for $\lambda \in \mathbb{R}^+$. There will, however, be a change in the functional margin as opposed to the geometric margin. Theorem 5.2 involves the *geometric margin*, that is the functional margin of a normalised weight vector. Hence, we can equally well optimise the geometric margin by fixing the functional margin to be equal to 1 (hyperplanes with functional margin 1 are sometimes known as *canonical hyperplanes*) and minimising the norm of the weight vector. The resulting geometric margin will be equal to $1/\|\mathbf{w}\|_2$, since this will be the functional margin if we scale by this amount to create the required unit weight vector. The following proposition summarises the situation.

Proposition 5.1. *Given a linearly separable training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$$

the hyperplane $(\mathbf{w},b)$ *that solves the optimisation problem*

$$\begin{array}{ll} \textit{minimise}_{\mathbf{w},b} & \langle \mathbf{w}, \mathbf{w} \rangle, \\ \textit{subject to} & y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) \geq 1, \\ & i = 1, \ldots, m, \end{array}$$

realises the maximal margin hyperplane with geometric margin $\gamma = 1/\|\mathbf{w}\|_2$.

The solution of this problem requires the use of quadratic optimization theory. Surprisingly, it naturally produces a dual representation analogous to the one encountered for the dual perceptron. The primal Lagrangian is

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^{m} \alpha_i \left[y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1 \right]$$

where $\alpha_i \geq 0$ are Lagrange multipliers, and the factor $\frac{1}{2}$ has been inserted for algebraic convenience and does not affect the generality of the result. The corresponding dual is found by differentiating with respect to $\mathbf{w}$ and b, imposing stationarity,

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i = \mathbf{0},$$
$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \sum_{i=1}^{m} y_i \alpha_i = 0,$$

and resubstituting the relations obtained,

$$\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i,$$
$$0 = \sum_{i=1}^{m} y_i \alpha_i,$$

into the primal we obtain

$$\begin{aligned} L(\mathbf{w}, b, \alpha) &= \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle - \sum_{i=1}^{m} \alpha_i \left[y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1 \right] \\ &= \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{m} \alpha_i \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle . \end{aligned}$$

The first of the substitution relations shows that the hypothesis can be described as a linear combination of the training points: the application of optimisation theory naturally leads to the dual representation already encountered for the perceptron. This means we can use maximal margin hyperplanes in combination with with kernels.

It is rather remarkable that one can achieve a nonlinear separation of this type by solving a convex problem, since neural networks often used for this purpose are affected by local minima in the training. The situation is summarized in the following proposition (following from 5.1).

Proposition 5.2. *Consider a linearly separable training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) ,$$

and suppose the parameters α^* *solve the following quadratic optimisation problem:*

$$\left.\begin{array}{l} \text{maximise } W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \\ \text{subject to } \sum_{i=1}^{m} y_i \alpha_i = 0, \\ \qquad \alpha_i \geq 0,\ i = 1, \ldots, m. \end{array}\right\} \tag{5.2}$$

Then the weight vector $\mathbf{w}^* = \sum_{i=1}^{m} y_i \alpha_i^* \mathbf{x}_i$ *realises the maximal margin hyperplane with geometric margin*

$$\gamma = 1 / \left\| \mathbf{w}^* \right\|_2 .$$

The value of b does not appear in the dual problem and so b^* must be found making use of the primal constraints:

$$b^* = -\frac{\max_{y_i=-1}(\langle \mathbf{w}^*, \mathbf{x}_i \rangle) + \min_{y_i=1}(\langle \mathbf{w}^*, \mathbf{x}_i \rangle)}{2}$$

The Kuhn-Tucker Theorem from optimization theory provides further information about the structure of the solution. The Karush-Kuhn-Tucker (KKT) conditions state that the optimal solutions α^*, $(\mathbf{w}^*, b^*)$ must satisfy

$$\alpha_i^* \left[y_i \left(\langle \mathbf{w}_i^*, \mathbf{x}_i \rangle + b^* \right) - 1 \right] = 0, \qquad i = 1, \ldots, m.$$

This implies that only for inputs $\mathbf{x}_i$ for which the functional margin is one and that therefore lie closest to the hyperplane are the corresponding α_i^* non-zero. All the other parameters α_i^* are zero. Hence, in the expression for the weight vector only these points are involved. It is for this reason that they are called *support vectors*. We will denote the set of indices of the support vectors with sv.

In other words, the optimal hyperplane can be expressed in the dual representation in terms of this subset of the parameters:

$$\begin{aligned} f(\mathbf{x}, \alpha^*, b^*) &= \sum_{i=1}^{m} y_i \alpha_i^* \langle \mathbf{x}_i, \mathbf{x} \rangle + b^* \\ &= \sum_{i \in \mathrm{SV}} y_i \alpha_i^* \langle \mathbf{x}_i, \mathbf{x} \rangle + b^*. \end{aligned}$$

The Lagrange multipliers associated with each point become the dual variables, giving them an intuitive interpretation quantifying how important a given training point is in forming the final solution. Points that are not support vectors have no influence, so that in non-degenerate cases slight perturbations of such points will not affect the solution. A similar meaning was found in the case of the dual representations for the perceptron learning algorithm, where the dual variable was proportional to the number of mistakes made by the algorithm on a given point during the training.

Another important consequence of the Karush Kuhn Tucker complementarity conditions is that for $j \in \mathrm{sv}$,

$$y_j f(\mathbf{x}_j, \alpha^*, b^*) = y_j \left(\sum_{i \in \mathrm{SV}} y_i \alpha_i^* \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b^* \right) = 1,$$

and therefore we can express the norm of the weight vector and the margin separating the data as a function of the multipliers α:

$$\begin{aligned}\langle \mathbf{w}^*, \mathbf{w}^* \rangle &= \sum_{i,j=1}^{m} y_i y_j \alpha_i^* \alpha_j^* \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &= \sum_{j \in \mathrm{SV}} \alpha_j^* y_j \sum_{i \in \mathrm{SV}} y_i \alpha_i^* \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &= \sum_{j \in \mathrm{SV}} \alpha_j^* (1 - y_j b^*) \\ &= \sum_{i \in \mathrm{SV}} \alpha_i^* .\end{aligned}$$

This gives us the following remarkable proposition, connecting all the main quantities of the algorithm:

Proposition 5.3. *Consider a linearly separable training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) ,$$

and suppose the parameters α^* and b^* solve the dual optimisation problem (5.2). Then the weight vector $\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i^* \mathbf{x}_i$ realises the maximal margin hyperplane with geometric margin

$$\gamma = 1/\left\|\mathbf{w}\right\|_2 = \left(\sum_{i \in sv} \alpha_i^* \right)^{-1/2} .$$

For convenience of the Reader, we summarize all the results obtained above in a single proposition, and we express them by explicitly using kernel functions.

Proposition 5.4. ***(Maximal Margin Algorithm)*** *Consider a training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$$

that is linearly separable in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ and suppose the parameters α^* and b^* solve the following quadratic optimisation problem:

$$\left.\begin{array}{l} \text{maximise } W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to } \sum_{i=1}^{m} y_i \alpha_i = 0, \\ \qquad\qquad \alpha_i \geq 0,\ i = 1, \ldots, m. \end{array}\right\} \tag{5.3}$$

Then the decision rule given by $sgn(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, is equivalent to the maximal margin hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ and that hyperplane has geometric margin

$$\gamma = \left(\sum_{i \in sv} \alpha_i^* \right)^{-1/2} .$$

Note that the requirement that the kernel satisfy Mercer's conditions is equivalent to the requirement that the matrix with entries $(K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$ be positive definite for all training sets. This in turn means that the optimisation problem (5.3) is convex since the matrix $(y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^m$ is also positive definite. Hence, the property required for a kernel function to define a feature space also ensures that the maximal margin optimisation problem has a unique solution that can be found efficiently. This rules out the problem of local minima encountered in training neural networks.

The fact that only a subset of the Lagrange multipliers is non-zero is often referred to as *sparseness*, and means that the support vectors contain all the information necessary to reconstruct the hyperplane. Even if all of the other points were removed the same maximal separating hyperplane would be found for the remaining subset of the support vectors. This can also be seen from the dual problem, since removing rows and columns corresponding to non-support vectors leaves the same optimisation problem for the remaining submatrix. Hence, the optimal solution remains unchanged. This shows that the maximal margin hyperplane is a compression scheme in the sense that we can reconstruct the maximal margin hyperplane from just the support vectors. This fact makes it possible to prove the following theorem, connecting generalization power of the function with its sparseness.

Theorem 5.5. *Consider thresholding real-valued linear functions $\mathcal{L}$ with unit weight vectors on an inner product space X. For any probability distribution $\mathcal{D}$ on $X \times \{-1, 1\}$, with probability $1 - \delta$ over m random examples S, the maximal margin hyperplane has error no more than*

$$err_{\mathcal{D}}(f) \leq \frac{1}{m-d}\left(d \log \frac{em}{d} + \log \frac{m}{\delta}\right),$$

where $d = \#sv$ is the number of support vectors.

The only degree of freedom left in the maximal margin algorithm is the choice of kernel, which amounts to model selection. Any prior knowledge we have of the problem can help in choosing a parametrised kernel family, and then model selection is reduced to adjusting the parameters. For most classes of kernels, for example polynomial or Gaussian, it is always possible to find a kernel parameter for which the data become separable. In general, however, forcing separation of the data can easily lead to overfitting, particularly when noise is present in the data.

In this case, outliers would typically be characterised by a large Lagrange multiplier, and the procedure could be used for data cleaning, since it can rank the training data according to how difficult they are to classify correctly.

This algorithm provides the starting point for the many variations on this theme proposed in the last few years and attempting to address some of its weaknesses: that it is sensitive to the presence of noise; that it only considers two classes; that it is not expressly designed to achieve sparse solutions.

5.3.2 Soft Margin Optimisation

The maximal margin classifier described in the previous subsection is an important concept, as a starting point for the analysis and construction of more sophisticated Support Vector Machines, but it cannot be used in many real-world problems: if the data are noisy, there will in general be no linear separation in the feature space (unless we are ready to use very powerful kernels, and hence overfit the data). The main problem with the maximal margin classifier is that it always produces a consistent classification function, that is a classification function with no training error. This is of course a result of its motivation in terms of a bound that depends on the margin, a quantity that is negative unless the data are perfectly separated.

The dependence on a quantity like the margin opens the system up to the danger of falling hostage to the idiosyncracies of a few points. In real data, where noise can always be present, this can result in a brittle (non robust) estimator. Furthermore, in the cases where the data are not linearly separable in the feature space, the optimisation problem cannot be solved as the primal has an empty feasible region and the dual an unbounded objective function. These problems motivate using the more robust measures of the *margin distribution* introduced above. Such measures can tolerate noise and outliers, and take into consideration the positions of more training points than just those closest to the boundary.

Algorithms motivated by such robust bounds are often called Soft Margin Classifiers and will be presented in this subsection. Recall that the primal optimisation problem for the maximal margin case is the following:

$$\begin{aligned} &\text{minimise}_{\mathbf{w},b}\ \langle \mathbf{w}, \mathbf{w} \rangle, \\ &\text{subject to} \quad y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) \geq 1,\ i = 1, \ldots, m. \end{aligned}$$

In order to optimise the margin slack vector we need to introduce slack variables to allow the margin constraints to be violated, so that the constraints become:

$$\begin{aligned} \text{subject to } y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) &\geq 1 - \xi_i,\ i = 1, \ldots, m, \\ \xi_i &\geq 0,\ i = 1, \ldots, m. \end{aligned}$$

and at the same time we need to somehow limit the overall size of the violations ξ_i.

Notice that optimizing a combination of the violations and the margin is directly equivalent to optimizing the statistical bounds given in Section 5.2. This can be easily seen, for example in the case of Theorem 5.3, that bounds the generalisation error in terms of the 2-norm of the margin slack vector, the so-called 2-norm soft margin, which contains the ξ_i scaled by the norm of the weight vector $\mathbf{w}$. The equivalent expression on which the generalisation depends is $\frac{R^2 + \frac{\|\xi\|_2^2}{\|\mathbf{w}\|_2^2}}{\gamma^2} = \|\mathbf{w}\|_2^2 \left(R^2 + \frac{\|\xi\|_2^2}{\|\mathbf{w}\|_2^2} \right) = \|\mathbf{w}\|_2^2 R^2 + \|\xi\|_2^2$, motivating the following optimization problem:

$$\left.\begin{aligned} &\text{minimise}_{\xi,\mathbf{w},b}\ \langle \mathbf{w}, \mathbf{w} \rangle + C \textstyle\sum_{i=1}^{m} \xi_i^2, \\ &\text{subject to} \quad y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i,\ i = 1, \ldots, m, \end{aligned}\right\} \tag{5.4}$$

The parameter C controls the trade-off between classification errors and margin. In practice it is varied through a wide range of values and the optimal performance assessed using a separate validation set or techniques of cross-validation. As the parameter C runs through a range of values, the norm of the solution $\|\mathbf{w}\|_2$ varies smoothly through a corresponding range. Hence, for a particular problem, choosing a particular value for C corresponds to choosing a value for $\|\mathbf{w}\|_2$, and then minimising $\|\xi\|_2$ for that size of $\mathbf{w}$. This approach is also adopted in the 1-norm case where the optimisation problem minimises a combination of the norm of the weights and the 1-norm of the slack variables that does not exactly match that found in Theorem 5.4:

$$\left.\begin{array}{ll} \text{minimise}_{\xi,\mathbf{w},b} & \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{i=1}^{m} \xi_i, \\ \text{subject to} & y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i, \; i = 1, \ldots, m, \\ & \xi_i \geq 0, \; i = 1, \ldots, m. \end{array}\right\} \tag{5.5}$$

Since there is a value of C corresponding to the optimal choice of $\|\mathbf{w}\|_2$, that value of C will give the optimal bound as it will correspond to finding the minimum of $\|\xi\|_1$ with the given value for $\|\mathbf{w}\|_2$. We will devote the next two subsections to investigating the duals of the two margin slack vector problems creating the so-called soft margin algorithms. This will give considerable insight into the structure and the properties of the solutions, as well as being practically important in the implementation.

2-Norm Soft Margin. The primal Lagrangian for the problem of equation (5.4) is

$$L(\mathbf{w}, b, \xi, \alpha) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 - \sum_{i=1}^{m} \alpha_i \left[y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right]$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. The corresponding dual is found by differentiating with respect to $\mathbf{w}$, ξ and b, imposing stationarity,

$$\begin{aligned} \frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i = \mathbf{0}, \\ \frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial \xi} &= C\xi - \alpha = \mathbf{0}, \\ \frac{\partial L(\mathbf{w}, b, \xi, \alpha)}{\partial b} &= \sum_{i=1}^{m} y_i \alpha_i = 0, \end{aligned}$$

and resubstituting the relations obtained into the primal to obtain the following objective function:

$$\begin{aligned} L(\mathbf{w}, b, \xi, \alpha) &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \frac{1}{2C} \langle \alpha, \alpha \rangle - \frac{1}{C} \langle \alpha, \alpha \rangle \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \frac{1}{2C} \langle \alpha, \alpha \rangle . \end{aligned}$$

Hence, maximising the above objective over α is equivalent to maximising

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \frac{1}{C} \delta_{ij} \right),$$

where δ_{ij} is the Kronecker δ defined to be 1 if $i = j$ and 0 otherwise. The corresponding Karush Kuhn Tucker complementarity conditions are

$$\alpha_i \left[y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 + \xi_i \right] = 0, \, i = 1, \ldots, m.$$

Therefore, we have the following result (where we use kernels instead of inner products):

Proposition 5.5. ***(2-Norm Soft Margin Classifier)*** *Consider classifying a training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters α^* solve the following quadratic optimisation problem:

$$\begin{array}{l} \textit{maximise } W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \left(K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C} \delta_{ij} \right), \\ \textit{subject to } \sum_{i=1}^{m} y_i \alpha_i = 0, \\ \qquad\qquad \alpha_i \geq 0, \, i = 1, \ldots, m. \end{array}$$

Let $f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $y_i f(\mathbf{x}_i) = 1 - \alpha_i^*/C$ for any i with $\alpha_i^* \neq 0$. Then the decision rule given by $sgn(f(\mathbf{x}))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ which solves the optimisation problem (5.4), where the slack variables are defined relative to the geometric margin

$$\gamma = \left(\sum_{i \in sv} \alpha_i^* - \frac{1}{C} \langle \alpha^*, \alpha^* \rangle \right)^{-1/2}.$$

Proof. The value of b^* is chosen using the relation $\alpha_i = C\xi_i$ and by reference to the primal constraints which by the Karush Kuhn Tucker complementarity conditions

$$\alpha_i \left[y_i \left(\langle \mathbf{w}_i, \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0, \, i = 1, \ldots, m,$$

must be equalities for non-zero α_i. It remains to compute the norm of $\mathbf{w}^*$ which defines the size of the geometric margin.

$$\begin{aligned}\langle \mathbf{w}^*, \mathbf{w}^* \rangle &= \sum_{i,j=1}^{m} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in \mathrm{SV}} \alpha_j^* y_j \sum_{i \in \mathrm{SV}} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in \mathrm{SV}} \alpha_j^* \left(1 - \xi_j^* - y_j b^*\right) \\ &= \sum_{i \in \mathrm{SV}} \alpha_i^* - \sum_{i \in \mathrm{SV}} \alpha_i^* \xi_i^* \\ &= \sum_{i \in \mathrm{SV}} \alpha_i^* - \frac{1}{C} \langle \alpha^*, \alpha^* \rangle .\end{aligned}$$

This is still a quadratic programming problem, and can be solved with the same methods used for the maximal margin hyperplane. The only change is the addition of $1/C$ to the diagonal of the kernel matrix. This has the effect of adding $1/C$ to the eigenvalues of the matrix, rendering the problem better conditioned. We can therefore view the 2-norm soft margin as simply a change of kernel

$$\hat{K}(\mathbf{x}, \mathbf{z}) = K(\mathbf{x}, \mathbf{z}) + \frac{1}{C} \delta_{\mathbf{x}}(\mathbf{z}).$$

1-Norm Soft Margin. The Lagrangian for the 1-norm soft margin optimisation problem presented above is

$$\begin{aligned}L(\mathbf{w}, b, \xi, \alpha, \mathbf{r}) = \frac{1}{2} \langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{i=1}^{m} \xi_i \\ - \sum_{i=1}^{m} \alpha_i \left[y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 + \xi_i \right] - \sum_{i=1}^{m} r_i \xi_i\end{aligned}$$

with $\alpha_i \geq 0$ and $r_i \geq 0$. The corresponding dual is found by differentiating with respect to $\mathbf{w}$, ξ and b, imposing stationarity,

$$\begin{aligned}\frac{\partial L(\mathbf{w}, b, \xi, \alpha, \mathbf{r})}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i = \mathbf{0}, \\ \frac{\partial L(\mathbf{w}, b, \xi, \alpha, \mathbf{r})}{\partial \xi_i} &= C - \alpha_i - r_i = 0, \\ \frac{\partial L(\mathbf{w}, b, \xi, \alpha, \mathbf{r})}{\partial b} &= \sum_{i=1}^{m} y_i \alpha_i = 0,\end{aligned}$$

and resubstituting the relations obtained into the primal. We obtain the following objective function:

$$L(\mathbf{w}, b, \xi, \alpha, \mathbf{r}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle ,$$

which curiously is identical to that for the maximal margin. The only difference is that the constraint $C-\alpha_i-r_i=0$, together with $r_i \geq 0$, enforces $\alpha_i \leq C$, while $\xi_i \neq 0$ only if $r_i = 0$ and therefore $\alpha_i = C$. The Karush Kuhn Tucker complementarity conditions are therefore

$$\begin{aligned} \alpha_i \left[y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 + \xi_i \right] &= 0, \; i = 1, \ldots, m, \\ \xi_i \left(\alpha_i - C \right) &= 0, \qquad\qquad i = 1, \ldots, m. \end{aligned}$$

Notice that the KKT conditions imply that non-zero slack variables can only occur when $\alpha_i = C$. The points with non-zero slack variables are $1/\|\mathbf{w}\|$-margin errors, as their geometric margin is less than $1/\|\mathbf{w}\|$. Points for which $0 < \alpha_i < C$ lie at the target distance of $1/\|\mathbf{w}\|$ from the hyperplane. We therefore have the following proposition.

Proposition 5.6. ***(1-Norm Soft Margin Classifier)*** *Consider classifying a training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) ,$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters α^* solve the following quadratic optimisation problem:

$$\left.\begin{array}{l} \text{maximise } W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to } \sum_{i=1}^{m} y_i \alpha_i = 0, \\ \qquad\qquad C \geq \alpha_i \geq 0, \; i = 1, \ldots, m. \end{array}\right\} \tag{5.6}$$

Let $f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $y_i f(\mathbf{x}_i) = 1$ for any i with $C > \alpha_i^* > 0$. Then the decision rule given by $sgn(f(\mathbf{x}))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ that solves the optimisation problem (5.5), where the slack variables are defined relative to the geometric margin

$$\gamma = \left(\sum_{i,j \in sv} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \right)^{-1/2} .$$

The value of b^* is chosen using the Karush Kuhn Tucker complementarity conditions which imply that if $C > \alpha_i^* > 0$ both $\xi_i^* = 0$ and

$$y_i(\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^*) - 1 + \xi_i^* = 0.$$

The norm of $\mathbf{w}^*$ is clearly given by the expression

$$\begin{aligned} \langle \mathbf{w}^*, \mathbf{w}^* \rangle &= \sum_{i,j=1}^{m} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in \mathrm{SV}} \sum_{i \in \mathrm{SV}} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j). \end{aligned}$$

So surprisingly this problem is equivalent to the maximal margin hyperplane, with the additional constraint that all the α_i are upper bounded by C. This gives rise to the name *box constraint* that is frequently used to refer to this formulation, since the vector α is constrained to lie inside the box with side length C in the positive orthant. The trade-off parameter between accuracy and regularisation directly controls the size of the α_i. This makes sense intuitively as the box constraints limit the influence of outliers, which would otherwise have large Lagrange multipliers. The constraint also ensures that the feasible region is bounded and hence that the primal always has a non-empty feasible region.

It is now easy to see why the maximal (or hard) margin case is an important concept in the solution of more sophisticated versions of the machine: both the 1- and the 2-norm soft margin machines lead to optimisation problems that are solved by relating them to the maximal margin case.

One problem with the soft margin approach suggested is the choice of parameter C. Typically a range of values must be tried before the best choice for a particular training set can be selected. Furthermore the scale of the parameter is affected by the choice of feature space. It has been shown, however, that the solutions obtained for different values of C in the optimisation problem (5.6) are the same as those obtained as ν is varied between 0 and 1 in the optimisation problem

$$\begin{array}{ll} \text{maximise } W(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subj. to} \quad \sum_{i=1}^{m} y_i \alpha_i = 0, \\ \qquad\qquad \sum_{i=1}^{m} \alpha_i \geq \nu \\ \qquad\qquad 1/m \geq \alpha_i \geq 0,\ i = 1, \ldots, m. \end{array}$$

In this parametrisation ν places a lower bound on the sum of the α_i, which causes the linear term to be dropped from the objective function. It can be shown that the proportion of the training set that are margin errors is upper bounded by ν, while ν provides a lower bound on the total number of support vectors. Therefore ν gives a more transparent parametrisation of the problem which does not depend on the scaling of the feature space, but only on the noise level in the data.

5.3.3 Kernel Ridge Regression

We now discuss a classical approach to regression known as Ridge Regression, that generalizes Least Squares regression, and we will show how this simple method can be used in combination with kernels to obtain an algorithm that is equivalent to a statistical method known as a Gaussian Process. We will give it an independent derivation, which highlights the connections with the systems from the Support Vector family [460].

The problem is the one of finding a linear regression function $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ that fits a dataset $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ where the labels are real numbers: $y_i \in \mathbb{R}$. The quality of the fit is measured by the squares of the deviations $y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle$ between the predicted and the given labels, and at the same time we attempt to keep the norm of the function as small as possible.

The resulting trade-off can be stated as the following optimization problem:

$$\left.\begin{array}{l}\text{minimise } \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m \xi_i^2, \\ \text{subject to } y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle = \xi_i,\ i = 1, \ldots, m,\end{array}\right\} \tag{5.7}$$

from which we derive the following Lagrangian

$$\text{minimise } L(\mathbf{w}, \xi, \alpha) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m \xi_i^2 + \sum_{i=1}^m \alpha_i (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - \xi_i).$$

Differentiating and imposing stationarity, we obtain that

$$\mathbf{w} = \frac{1}{2\lambda} \sum_{i=1}^m \alpha_i \mathbf{x}_i \text{ and } \xi_i = \frac{\alpha_i}{2}.$$

Resubstituting these relations gives the following (unconstrained) dual problem:

$$\text{maximise } W(\alpha) = \sum_{i=1}^m y_i \alpha_i - \frac{1}{4\lambda} \sum_{i,j=1}^m \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \frac{1}{4} \sum \alpha_i^2,$$

that for convenience we rewrite in vector form:

$$W(\alpha) = \mathbf{y}'\alpha - \frac{1}{4\lambda} \alpha' \mathbf{K} \alpha - \frac{1}{4} \alpha' \alpha,$$

where $\mathbf{K}$ denotes the Gram matrix $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, or the kernel matrix $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, if we are working in a kernel-induced feature space. Differentiating with respect to α and imposing stationarity we obtain the condition

$$-\frac{1}{2\lambda} \mathbf{K} \alpha - \frac{1}{2} \alpha + \mathbf{y} = 0,$$

giving the solution

$$\alpha = 2\lambda (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

and the corresponding regression function

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{y}' (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}$$

where $\mathbf{k}$ is the vector with entries $k_i = \langle \mathbf{x}_i, \mathbf{x} \rangle$, $i = 1, \ldots, m$. Hence, we have the following result.

Proposition 5.7. ***(Kernel Ridge-Regression)*** *Suppose that we wish to perform regression on a training sample*

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and let $f(\mathbf{x}) = \mathbf{y}'(\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{k}$, where $\mathbf{K}$ is the $m \times m$ matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}$ is the vector with entries $\mathbf{k}_i = K(\mathbf{x}_i, \mathbf{x})$. Then the function $f(\mathbf{x})$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ that solves the ridge regression optimisation problem (5.7).

This algorithm has appeared independently under a number of different names. Apart from being the Gaussian Process solution, it is also known as Krieging and the solutions are known as regularisation networks, where the regulariser has been implicitly selected by the choice of kernel. It is very simple to implement, essentially requiring only a matrix inversion, and is a very effective way to solve non-linear regression problems. Also in this case, it is important to note how the training is not affected by local minima.

5.4. Kernel PCA and CCA

In this section we briefly describe two techniques for discovering hidden structure respectively in a set of unlabeled data and in a 'paired' dataset, that is formed by two datasets in bijection. In their linear version they were both introduced by Hotelling, and both can be adapted for use with kernel functions. The first one is the classical Principal Components Analysis (PCA), aimed at finding a low dimensional representation of the data, the second one is Canonical Correlation Analysis (CCA), aimed at finding correlations between a dataset formed by pairs of vectors or, equivalently, between two 'matched' datasets, whose elements are in bijection [329, 36, 465].

Principal Components Analysis is a classical technique for analysing high dimensional data, and extracting hidden structure by finding a (small) set of coordinates that carry most of the information in the data. It can be proven that the most informative directions are given by the k principal eigenvectors of the data, in the sense that this choice minimizes - for any fixed k - the mean square distance between the original and the reconstructed data (see discussion in Chapter 3). The eigenvectors are called the *principal axes* of the data, and the new coordinates of each point are obtained by projecting it onto the first k principal axes.

As before, we will represent such vectors in dual form, as linear combinations of data vectors $\mathbf{v} = \sum_i \alpha_i \mathbf{x}_i$ and we will need to find the parameters α_i. Given a set of (unlabeled) observations $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ that are centered, $\sum_i \mathbf{x}_i = 0$, the (empirical) covariance matrix is defined as:

$$C = \frac{1}{m-1} \sum_i \mathbf{x}_i \mathbf{x}_i^T$$

and is a positive semi-definite matrix. Its eigenvectors and eigenvalues can be written as:

$$\lambda \mathbf{v} = C\mathbf{v} = \sum_i \langle \mathbf{x}_i, \mathbf{v} \rangle \mathbf{x}_i$$

that is each eigenvector can be written as linear combination of the training points,

$$\mathbf{v} = \sum_i \alpha_i \mathbf{x}_i$$

for some α, hence allowing a dual representation and the use of kernels. The new coordinates of a point are then given by projecting it onto the eigenvectors, and it is possible to prove that for any k using the first k eigenvectors gives the best approximation in that it minimises the sum of the 2-norms of the residuals of the training points.

By performing the same operation in the feature space, that is using the images of the points $\phi(\mathbf{x}_i)$, with simple manipulations we can find that the coefficients α^n of the n-th eigenvector can be obtained by solving the eigenvalue problem

$$m\lambda\alpha = K\alpha$$

and subsequently imposing the normalization $1 = \lambda_n\langle\alpha^n, \alpha^n\rangle$, $n = 1, \ldots, n$. Although we do not have the explicit coordinates of the eigenvectors, we can always use them for calculating the projections of the data points onto the n-th eigenvector $\mathbf{v}^n$, as follows

$$\langle\phi(\mathbf{x}), \mathbf{v}^n\rangle = \sum_{i=1}^{m} \alpha_i^n K(\mathbf{x}_i, \mathbf{x})$$

and this information is all we need for recoding our data and extracting hidden regularities.

Canonical Correlation Analysis (CCA) is a technique (also introduced by Hotelling) that can be used when we have two datasets that might have some underlying correlation. Assume there is a bijection between the elements of the two sets, possibly corresponding to two alternative descriptions of the same object (e.g., two views of the same 3D object; or two versions of the same document in 2 languages).

Given a set of pairs $S = \{(\mathbf{x}^1, \mathbf{x}^2)_i\}$, the task of CCA is to find linear combinations of variables in each of the two sets that have the maximum mutual correlation. Given two real valued variables a and b with zero mean, we define their correlation as

$$r = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2 \sum_i b_i^2}}.$$

where $\{a_i\}$ and $\{b_i\}$ are realizations of the random variable. The problem of CCA can be formalized as follows: given two sets of paired vectors, $\mathbf{x}_i^1 \in X_1$ and $\mathbf{x}_i^2 \in X_2$, $i = 1, \ldots, m$, find vectors $\mathbf{w}^1 \in X_1$ and $\mathbf{w}^2 \in X_2$ such that the projections of the data onto these vectors $a_i = \langle\mathbf{x}_i^1, \mathbf{w}^1\rangle$ and $b_i = \langle\mathbf{x}_i^2, \mathbf{w}^2\rangle$ have maximal correlation.

Solving this task can be transformed into the following generalized eigenvector problem

$$\begin{bmatrix} 0 & C_{12} \\ C_{21} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w}^1 \\ \mathbf{w}^2 \end{bmatrix} = \lambda \begin{bmatrix} C_{11} & 0 \\ 0 & C_{22} \end{bmatrix} \begin{bmatrix} \mathbf{w}^1 \\ \mathbf{w}^2 \end{bmatrix},$$

where

$$C_{jk} = \sum_{i=1}^{m} \mathbf{x}_i^j \left(\mathbf{x}_i^k\right)^T, \quad j, k = 1, 2.$$

The approach can be kernelized following the same procedures discussed above. We obtain that $\mathbf{w}^1 = \sum_i \alpha_i^1 \phi(\mathbf{x}_i^1)$ and $\mathbf{w}^2 = \sum_i \alpha_i^2 \phi(\mathbf{x}_i^2)$ and this leads to another generalized eigenvalue problem to find the dual variables α:

$$\begin{bmatrix} 0 & K_1 K_2 \\ K_2 K_1 & 0 \end{bmatrix} \begin{bmatrix} \alpha^1 \\ \alpha^2 \end{bmatrix} = \lambda \begin{bmatrix} K_1^2 & 0 \\ 0 & K_2^2 \end{bmatrix} \begin{bmatrix} \alpha^1 \\ \alpha^2 \end{bmatrix},$$

where K_1 and K_2 are the kernel matrices for the vectors $\mathbf{x}_i^1 \in X_1$ and $\mathbf{x}_i^2 \in X_2$, that by assumption are in bijection, that is the ij-th entry in each matrix corresponds to the same pair of points.

By solving this problem, one can find nonlinear transformations of the data both in the first and in the second set that maximise the correlation between them. One use of this approach can be to analyze two different representations of the same object, possibly translations in different languages of the same documents, or different views of the same object in machine vision.

5.5. Conclusion

We have seen how the problem of finding nonlinear relations between points in a space can be reduced to finding linear relations in a kernel-induced feature space, and how this can first be reduced to solving a convex optimization problem. This approach includes methods for classification, regression and unsupervised learning. Furthermore, we have shown how the use of concepts from Statistical Learning Theory can give insight into which parameters of the algorithm should be controlled in order to avoid overfitting.

The result is a very versatile family of algorithms, known as Kernel Methods, that combine the statistical robustness provided by rigorous analysis with the computational efficiency given by convex optimization. The absence of local minima, however, is not the only reason for their fast uptake among data analysts. Their modular design, for example, by which the algorithm and the kernel function can be implemented and studied separately, means that any algorithm can work with any kernel, and hence previous work can be easily reused and prior knowledge naturally incorporated.

We have not discussed another important feature of such approach, that is worth mentioning at this point. Kernels can be defined between pairs of data items that are not vectors: we can consider embedding two symbol sequences of different length into some vector space where their inner product is defined, even if the sequences themselves are not vectors. This can be done efficiently by means of special string-matching kernels, and has proven useful in the analysis of biological data. Similarly, kernels exist for text, images, graphs and other data structures other than vectors. The design of a good kernel function requires some prior knowledge of the domain and the task, and once a good kernel for a type of data has been found, it can be used for a number of tasks, from classification to regression to PCA and many more.

Data Analysis practitioners can now rely on a rich toolbox containing several kernel-based algorithms together with a choice of kernels, each of which can

be combined in a modular way. The development of such systems in the last few years, within a common theoretical framework, opens unprecedented opportunities for intelligent data analysis. In this Chapter we have presented just Support Vector Machines, kernel Ridge Regression and, briefly, kernel Principal Components and Canonical Correlation Analysis. Methods for anomaly detection, ranking, Time Series analysis, even Reinforcement Learning have been designed for kernels. The interested Reader is referred to the papers and websites in the bibliography, but in particular to **www.kernel-machines.org** and **www.support-vector.net** for further information, papers, books and online software related to this fast growing literature [134, 488, 135].