Chapter 1

Support Vector Machines

This chapter discusses the development and basic theoretic building blocks of Support Vector Machines. An overview of both linear and non-linear Support Vector Machines is given from the viewpoint of pattern recognition. Kernel methods are introduced, and the chapter concludes with the Support Vector Machine training problem that will play a key role in the chapters to follow.

1.1 Introduction to Support Vector Machines

Support Vector Machines (SVMs) are a young and important addition to the machine learning toolbox. Having been formally introduced at the 1992 Workshop on Computational Learning Theory [6], SVMs have proved their worth. In the following decade there has been a remarkable growth in both the theory and practice of these learning machines. The original treatments of Support Vector Machines (SVMs) are due to [6, 14, 22, 58, 60].

Traditionally, a SVM is a learning machine for two-class classification problems, and learns from a set of examples. The algorithm aims to do a separation between the two classes by creating a linear decision surface between them. This surface is, however, not created in input space, but rather in a very high-dimensional feature space. Because the feature space is non-linearly related to the input space, the resulting model is non-linear. Special properties of the decision surface ensures high generalisation abilities of SVMs.

Although the Support Vector (SV) algorithm appears to be a linear algorithm in a high-dimensional space, no computations are done in that high-dimensional space. All computations are performed directly in input space by making use of kernel functions. Due to the use of Kernel Methods (KMs), a seemingly complex algorithm for non-linear pattern recognition or regression can be implemented and analysed as a simple linear algorithm. KMs are very modular. Any kernel function can be used with any kernel-based learning algorithm, and any kernel-based learning algorithm can work with any kernel function. By combining simple kernels to complex ones, the kernel functions themselves can also be derived in a modular way.

The SV algorithm makes use of "support vectors" to define the decision surface. Support vectors are a subset of the training patterns, or training vectors. These patterns can be called the most informative, and it is this subset of informative patterns that define the architecture of a SVM. If all non-support vector training patterns (the "uninformative" patterns) are removed, and the SVM retrained, the solution will be exactly the same.

SVMs are popular due to two main reasons. Firstly, an important characteristic of SVMs is its mathematical tractability and geometric interpretation. The SV algorithm is based on very theoretical and intuitive ideas. Secondly, SVMs have shown to be accurate in practical applications, with successes in fields as diverse as text categorisation, bioinformatics and machine vision.

The algorithm holds learning theory in one hand, and practice in the other. Statistical learning theory can be used to identify factors needed for certain algorithms to learn successfully. Complex models and algorithms – such as neural networks – are often needed for practical real-world applications. These models are, however, hard to analyse theoretically. SVMs construct models that are complex enough, with the advantage that the models are relatively simple to analyse mathematically. These models include a large class of neural networks, radial basis function (RBF) networks, and special cases of polynomial classifiers.

SVMs have become an increasingly popular alternative to neural networks. In comparison to neural networks, SVMs have only a small number of tuneable parameters. The SV algorithm also defines the architecture of the learning machine. The SVM training process is characterised by solving a convex quadratic programming problem. The solutions to the training problem are global, and usually unique [10]. A great benefit of SVM training is the absence of local minima (or maxima), and the learning parameters converge monotonically toward the solution.

Applications and theory of SVMs have been extended far beyond basic classification tasks to handle pattern recognition, regression, operator inversion, density estimation, and novelty detection. For pattern recognition, SVMs have been successfully applied in the areas of isolated handwritten digit recognition [9, 14, 45, 46], speaker identification [43], text categorisation [24], face detection in images [40] and object recognition [4]. In the case of regression estimation problems, SVMs have been compared to benchmark time series prediction tests [35, 37]. SVMs have also been used for density estimation [61] and ANOVA decomposition [53].

Although the SV algorithm is firmly rooted in statistical learning theory, learning theory is not included in this work. An excellent explanation can be found in [58, 59]. This chapter focuses on the creation of SVMs: The basic idea behind pattern recognition is explained, which is used in constructing an optimal hyperplane and linear SVMs for linearly separable data. The linear SVM is then adapted to handle nonseparable classification problems. Finally, SVMs are extended to non-linear classification models by the use of kernel functions.

1.2 Pattern recognition

By observing their environment, machines can learn to distinguish interesting patterns. These patterns can be any entity that can be given a name, for example a handwritten character or word, a fingerprint, a face, or a speech signal. After learning, the machine should be able to make intelligent decisions about the categories of similar patterns – this process is called pattern recognition.

Pattern recognition algorithms can be divided into two principal groups. Identifying a pattern as a member of a predefined class is called *supervised* learning and classification. If the algorithm learns classes of patterns based on a measure of similarity, the process is called *unsupervised* learning, or clustering. Unsupervised classification assigns a pattern to one of these determined classes. A SVM is an example of supervised classification, learning from example patterns with class labels.

For a given pattern recognition problem, the objective is to estimate a function $f : \mathbb{R}^N \to \{\pm 1\}$ using a finite set of training data. The training data set consists of a total of l N-dimensional patterns $\mathbf{x}_i$ and their respective class labels y_i, i.e.

$$\{\mathbf{x}_1, y_1\}, \ldots, \{\mathbf{x}_l, y_l\} \in \mathbb{R}^N \times \{\pm 1\} \tag{1.1}$$

If a new pattern $\{\mathbf{x}, y\}$ is generated from the same underlying probability density function $P(\mathbf{x}, y)$ as the training data, then f should correctly classify this example – that is, $f(\mathbf{x}) = y$.

1.3 Linear Support Vector Machines

When training data is linearly separable, a separating hyperplane (a hyperplane that separates the positive from the negative examples) of the form

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{1.2}$$

can be fitted to correctly classify training patterns. Here the vector $\mathbf{w}$ is normal to the hyperplane, and defines its orientation. This hyperplane is shown in Figure 1.1. From equation (1.2), a decision function

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b). \tag{1.3}$$

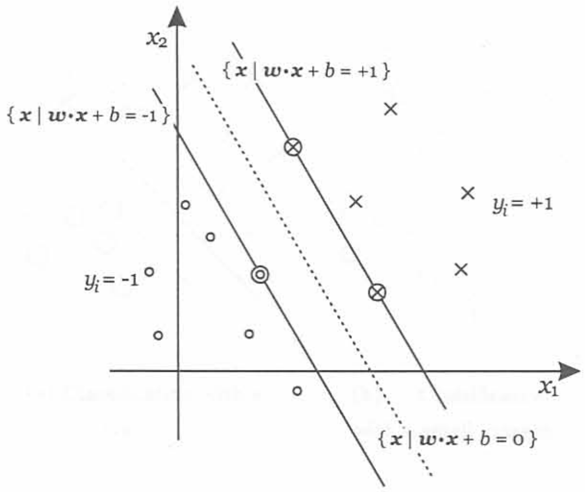

x_2
$\{x \mid w \cdot x + b = +1\}$
$\{x \mid w \cdot x + b = -1\}$
$y_i = +1$
$y_i = -1$
x_1
$\{x \mid w \cdot x + b = 0\}$

FIGURE 1.1: An example of a classification problem in two dimensions, with the support vectors encircled.

can be derived, with f classifying both positive ($y_i = +1$) and negative ($y_i = -1$) patterns.

Let d_+ (d_-) be the shortest distance from the separating hyperplane to the closest positive (negative) example, then the margin of the hyperplane is defined as the sum $d_+ + d_-$. An optimal hyperplane for a linearly seperable set of training data is here defined as the linear decision function with maximal margin between the vectors of the two classes, as is shown in Figures 1.2(a) and 1.2(b). The support vector algorithm will construct this optimal separating hyperplane.

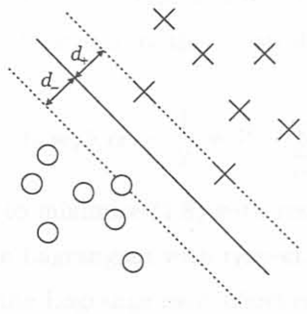
It was shown in [57] that the optimal hyperplane will have good generalisation abilities, and only a relatively small amount of training data is needed to construct this plane. The set of margin-determining training vectors are called the *support vectors*. It was also shown that if the training vectors are separated without errors by an optimal hyperplane, the expected value of the probability of committing an error on a test example is bounded by the ratio between the expected number of support vectors and the number of training vectors:

$$E[P(\text{error})] \leq \frac{E[\text{number of support vectors}]}{\text{number of training vectors}} \tag{1.4}$$

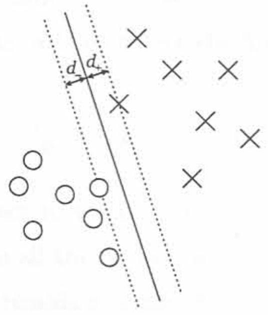
The bound given in equation (1.4) does not explicitly contain the dimensionality of the space of separation. If the optimal hyperplane can thus be constructed from a small number of support vectors relative to the training set size, the generalisation ability will be high, even in an infinite-dimensional space.

Assume all training data satisfy

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq +1 \quad \text{for} \quad y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 \quad \text{for} \quad y_i = -1 \end{aligned} \tag{1.5}$$



(a) Classification with a large margin



(b) Classification with a small margin

FIGURE 1.2: Constructing an optimal hyperplane

as shown in Figure 1.1. This can be combined into a single set of equalities:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0 \qquad i = 1, \ldots, l \tag{1.6}$$

where l is the training set size.

To find the optimal separating hyperplane, it is necessary to maximise the margin $d_+ + d_-$. Suppose $\mathbf{x}_1$ and $\mathbf{x}_2$ with $y_1 = +1$ and $y_2 = -1$ are positive and negative points closest to the hyperplane. For maximal separation, the hyperplane should be as far away as possible from each of them. By letting $|| \cdot ||$ be the l_2 norm of a vector, get

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_1 + b &= +1 \\ \mathbf{w} \cdot \mathbf{x}_2 + b &= -1 \\ \Rightarrow \mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) &= +2 \\ \Rightarrow \frac{\mathbf{w}}{||\mathbf{w}||} \cdot (\mathbf{x}_1 - \mathbf{x}_2) &= \frac{2}{||\mathbf{w}||} \end{aligned}$$

Maximising the margin is equivalent to maximising $\frac{2}{||\mathbf{w}||}$, which is in turn the same as solving

$$\min_{\mathbf{w},b} \quad \frac{1}{2}||\mathbf{w}||^2 \tag{1.7}$$

subject to the constraints in (1.6). Constructing the optimal hyperplane is therefore a convex quadratic problem.

A standard optimisation technique, Lagrange multipliers [20], is used in constructing this optimal hyperplane. There are two main reasons for doing this. The first is that the constraints in (1.6) will be replaced by constraints on the Lagrange multipliers themselves, which will be much easier to handle. The second reason is that, in the Lagrangian reformulation, the training data will only appear as dot products between vectors. This is the crucial property that allows generalisation to the non-linear case. The Lagrange multipliers, $\alpha_i \geq 0$, are introduced for each of the constraints in (1.6) to get the following Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{l} \alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \tag{1.8}$$

The objective is to minimise (1.8) with respect to $\mathbf{w}$ and b, under the requirement that the derivatives of the Lagrangian with respect to all the α_i vanish. This must be subject to the constraint that the Lagrange multipliers α_i remain non-negative.

Since all the constraints are linear and thus convex, their intersection is also convex. Because the objective function is also convex, the problem is a convex quadratic programming problem. Thus it is possible to *equivalently* solve the dual optimisation problem of maximising (1.8), such that the gradient of L with respect to $\mathbf{w}$ and b vanishes, and requiring that $\alpha_i \geq 0$. That is,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \qquad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \tag{1.9}$$

and thus

$$\sum_{i=1}^{l} y_i \alpha_i = 0, \qquad \mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \tag{1.10}$$

By substituting (1.10) into (1.8), the dual form of the optimisation problem is derived. Determine

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{1.11}$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, l \qquad \text{and} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.12}$$

Thus, by solving the dual optimisation problem, the coefficients α_i are obtained. These coefficients are then used to calculate $\mathbf{w}$ from equation (1.10). The vector $\mathbf{w}$ will be a solution to problem (1.7). The decision function from equation (1.3) can be rewritten as

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{l} y_i \alpha_i \mathbf{x} \cdot \mathbf{x}_i + b\right) \tag{1.13}$$

The decision surface of (1.13) is determined by the l Lagrange multipliers α_i. These multipliers are either zero or positive. The subset of zero multipliers will have no effect on the decision function, and can be omitted. It is the subset of positive multipliers that

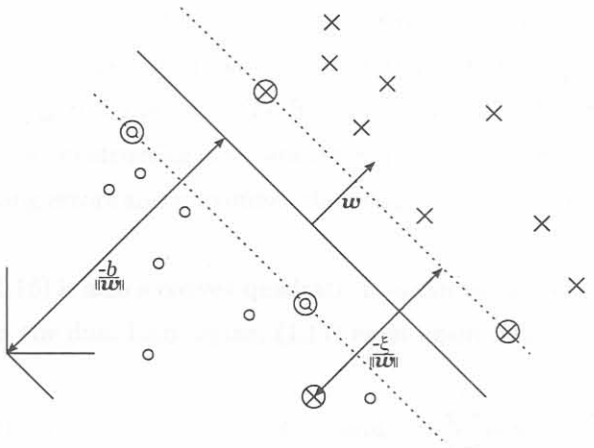


FIGURE 1.3: An example of a linear separating hyperplane for the non-separable case.

influences the classification, and their corresponding training vectors are called the *support vectors*.

The ideas presented in this section only handle separable data. Real data are usually non-separable data, and some examples might violate (1.6). In the following section, SVMs are extended to handle misclassifications.

1.4 Soft margin hyperplanes

In many cases it is impossible to separate the training data without errors, as illustrated in Figure 1.3. If separation by a hyperplane is impossible, the margin between patterns of the two classes becomes arbitrarily small, and the constrained dual Lagrangian (1.11) will grow arbitrarily large.

In this case the separation of the training set can be done with a minimal number of errors (misclassifications), by relaxing the constraints given in (1.6). Here the notion of "soft margin classifiers" are introduced. Add l nonnegative slack variables ξ_i to relax the hard-margin constraints:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0, \qquad i = 1, \ldots, l \tag{1.14}$$

Thus for an error to occur, the value of ξ_i must exceed one. It is clear that $\sum_i \xi_i$ is an upper bound on the number of training errors. The natural way to assign an extra cost for errors is to change to objective function to be minimised from (1.7), to solving

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l} \xi_i \tag{1.15}$$

Here $C > 0$ is an arbitrarily chosen – and problem dependent – parameter. A larger value of C assigns a *greater* penalty to errors, since it constrains $\sum_i \xi_i$ to a smaller value. A smaller C allows $\sum_i \xi_i$ to be larger. The functional in (1.15) describes (for sufficiently large C) the problem of constructing a separating hyperplane which minimises the sum of deviations, ξ, of training errors and maximises the margin for the correctly classified vectors [14].

The problem in (1.15) is also a convex quadratic programming problem. Since the values of ξ_i do not appear in the dual Lagrangian, (1.11) must again be maximised subject to

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l \qquad \text{and} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.16}$$

A crucial property of the quadratic programming problem in (1.11, 1.12) and the decision function $f(\mathbf{x}) = \text{sign}(\sum_i y_i \alpha_i \mathbf{x} \cdot \mathbf{x}_i + b)$ is that they depend only on dot products between patterns. It is this property that allows generalisation to the non-linear case.

1.5 Non-linear Support Vector Machines

A set of linear classifiers, as presented in the method above, is often not rich enough for more diverse classification problems. What is needed is a method that handles non-linear classification equally well. Linear SVMs can very easily be generalised to include these non-linear decision functions: Boser *et al* [6] showed that the so-called *kernel trick* [1] can accomplish this generalisation. Notice that the training patterns only appear in the form of dot products $\mathbf{x}_i \cdot \mathbf{x}_j$ in equations (1.11, 1.13). A non-linear transformation can be done on the set of input vectors to a higher dimensional space (where the dot product is defined), and the linear separation can be done in this higher dimensional space. The data are thus mapped into some other dot product space – a *feature space* – $\mathcal{F}$ via the non-linear map

$$\Phi : \mathbb{R}^N \rightarrow \mathcal{F} \tag{1.17}$$

The only requirement on $\mathcal{F}$ is that it is equipped with the dot product operator. No assumptions are made on the dimensionality of $\mathcal{F}$; it can possibly be an infinite-dimensional space. For a given training data set, the SVM is now constructed in $\mathcal{F}$ instead of $\mathbb{R}^N$, i.e. using the set of examples

$$\{\Phi(\mathbf{x}_1), y_1\}, \ldots, \{\Phi(\mathbf{x}_l), y_l\} \in \mathbb{R}^N \times \{\pm 1\} \tag{1.18}$$

From this mapped set of examples a decision function in $\mathcal{F}$ has to be estimated. Intuitively, the difficulty of constructing a decision function in input space should grow with the dimension of the patterns. Statisticians call this difficulty the *curse of dimensionality*

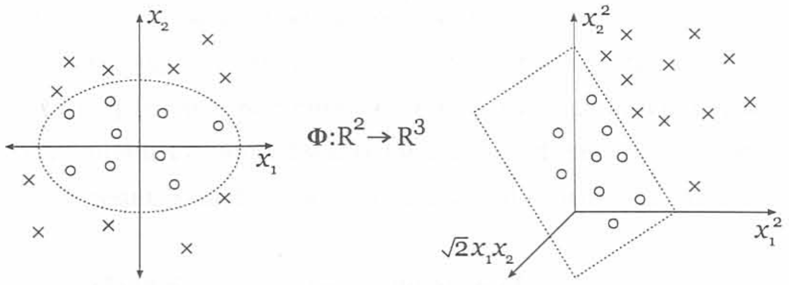


FIGURE 1.4: An example of two-dimensional classification. The three-dimensional feature space is defined by monomials x_1^2, $\sqrt{2}x_1x_2$, and x_2^2, where a linear decision surface is constructed. This construction corresponds to a non-linear ellipsoidal decision boundary in $\mathbb{R}^2$.

– a function of dimension N needs exponentially many patterns to sample the space properly. Considering the curse of dimensionality, mapping to a higher dimensional feature space seems like a dubious idea.

The contrary can, however, be true. Statistical learning theory [59] shows that learning in $\mathcal{F}$ can be simpler if functions of a lower complexity are used. It is the complexity of the function class, not the dimensionality, that matters. The richness of a powerful function class is then introduced by the mapping Φ.

This idea can be understood by considering a simple class of decision rules, namely linear classifiers. Consider the toy example in Figure 1.4, where the training vectors are two-dimensional. A complicated non-linear decision surface is needed to separate the training examples in input space. By defining the mapping

$$\begin{aligned} \Phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (x_1, x_2)^T &\mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T \end{aligned} \tag{1.19}$$

a *linear* hyperplane separates the mapped training vectors in a three-dimensional feature space. The feature space is defined by the second order monomials x_1^2, $\sqrt{2}x_1x_2$, and x_2^2. This construction corresponds to a non-linear ellipsoidal decision boundary [36].

In the above example, both the statistical complexity and the algorithmic complexity of the learning machine were controlled. The statistical complexity was controlled by using a simple linear hyperplane classifier. Using a three-dimensional feature space kept the algorithmic complexity low.

A technical problem arises in real-world problems, since the algorithmic complexity cannot be kept low: Patterns may be images of 16×16 pixels, a 256-dimensional input space. When fourth order monomials are used as mapping Φ, the feature space would contain all the fourth order products of 256 pixels, and its dimension will be $\binom{4+256-1}{4} \approx 200$ million.

In 1992 it was shown that the problem of treating such high-dimensional spaces could be overcome [6]. Instead of making a non-linear transformation of the input vectors followed by dot products with support vectors in the feature space $\mathcal{F}$, the order of operations is interchanged. A comparison is first done between two vectors in input space (for example by taking their dot product or some distance measure), and then a non-linear transformation of the value of the result is made. The comparison and transformation is done by a *kernel function*.

In the toy example of Figure 1.4, the computation of a dot product between two feature space vectors can be reformulated in terms of a kernel function k:

$$\begin{aligned}
\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) &= \begin{pmatrix} x_{i1}^2 \\ \sqrt{2}x_{i1}x_{i2} \\ x_{i2}^2 \end{pmatrix} \cdot \begin{pmatrix} x_{j1}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{pmatrix} \\
&= x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 \\
&= \left(\begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix} \cdot \begin{pmatrix} x_{j1} \\ x_{j2} \end{pmatrix} \right)^2 \\
&= (\mathbf{x}_i \cdot \mathbf{x}_j)^2 \\
&= k(\mathbf{x}_i, \mathbf{x}_j) \qquad (1.20)
\end{aligned}$$

Training a non-linear SVM thus requires the computation of dot products $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ in the feature space $\mathcal{F}$, and can be reduced by defining a suitable kernel function, k, such that

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \qquad (1.21)$$

The question, which function k corresponds to a dot product in some feature space $\mathcal{F}$, arises. In other words, how can a map Φ be constructed such that k computes the dot product in the space Φ maps to? This has been dealt with by [6, 58], and the answer is seen from Mercer's theorem of functional analysis [15]:

To guarantee that there exits a mapping Φ and an expansion

$$k(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \sum_i \Phi(\mathbf{u})_i \Phi(\mathbf{v})_i \qquad (1.22)$$

it is necessary and sufficient that the condition

$$\iint k(\mathbf{u}, \mathbf{v}) g(\mathbf{u}) g(\mathbf{v}) \, d\mathbf{u} \, d\mathbf{v} \geq 0 \qquad (1.23)$$

be valid for all g for which

$$\int g^2(\mathbf{u}) \, d\mathbf{u} < \infty \qquad (1.24)$$

As an example, consider the toy example of Figure 1.4, with the kernel defined in equation (1.20), and $\mathbf{x}$ a two-dimensional vector. To show that Mercer's condition is satisfied for $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, it must be shown that

$$\iint (\mathbf{x}_i \cdot \mathbf{x}_j)^2 g(\mathbf{x}_i) g(\mathbf{x}_j)\, d\mathbf{x}_i\, d\mathbf{x}_j \geq 0 \tag{1.25}$$

hold for all g with finite L_2 norm, i.e. g must satisfy equation (1.24). Expanding and factorising the left-hand side of the above inequality gives the needed result.

$$\begin{aligned}
& \iint (x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2) g(\mathbf{x}_i) g(\mathbf{x}_j)\, d\mathbf{x}_i\, d\mathbf{x}_j \\
= & \int x_{i1}^2 g(\mathbf{x}_i)\, d\mathbf{x}_i \int x_{j1}^2 g(\mathbf{x}_j)\, d\mathbf{x}_j + 2\int x_{i1}x_{i2} g(\mathbf{x}_i)\, d\mathbf{x}_i \cdots \\
& \cdots \int x_{j1}x_{j2} g(\mathbf{x}_j)\, d\mathbf{x}_j + \int x_{i2}^2 g(\mathbf{x}_i)\, d\mathbf{x}_i \int x_{j2}^2 g(\mathbf{x}_j)\, d\mathbf{x}_j \\
= & \Big(\int x_{i1}^2 g(\mathbf{x}_i)\, d\mathbf{x}_i\Big)^2 + 2\Big(\int x_{i1}x_{i2} g(\mathbf{x}_i)\, d\mathbf{x}_i\Big)^2 + \Big(\int x_{i2}^2 g(\mathbf{x}_i)\, d\mathbf{x}_i\Big)^2 \\
\geq & \ 0
\end{aligned} \tag{1.26}$$

In many specific cases it is not as easy to check Mercer's condition, since equation (1.23) must hold for every g with finite L_2 norm. Mercer's condition does give information on whether some kernel computes a dot product in some feature space, but it does not tell what the mapping Φ or the space $\mathcal{F}$ is.

When a kernel function does not comply with Mercer's condition, training data may exist that give rise to an indefinite Hessian matrix in the dual Lagrangian (1.11). The objective function can become arbitrarily large, and the quadratic programming problem will have no solution. Many training sets can still result in a positive semi-definite Hessian, and a SVM's constrained objective function can be maximised. The results, however, will not have the usual geometric interpretation of support vectors.

By definition of the kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, the SVM decision function becomes

$$\begin{aligned}
f(\mathbf{x}) &= \operatorname{sign}\Big(\sum_{i=1}^{l} y_i \alpha_i \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i) + b\Big) \\
&= \operatorname{sign}\Big(\sum_{i=1}^{l} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\Big)
\end{aligned} \tag{1.27}$$

The architecture of the above decision function defines the architecture of the SVM, as shown in Figure 1.5. Examples of kernel functions most commonly used in pattern recognition problems are:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p \tag{1.28}$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2} \tag{1.29}$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i \cdot \mathbf{x}_j - \delta) \tag{1.30}$$

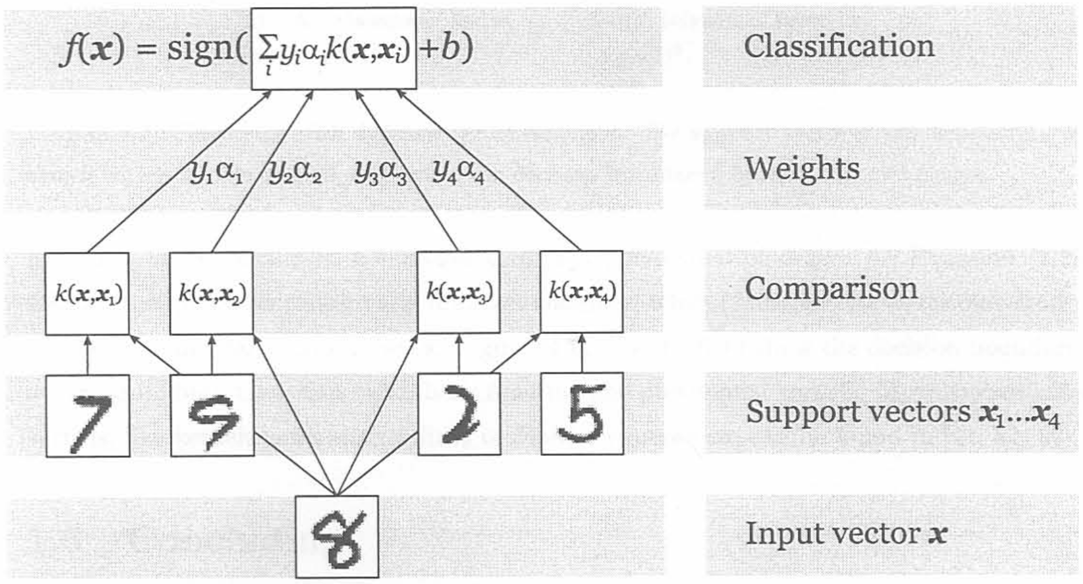
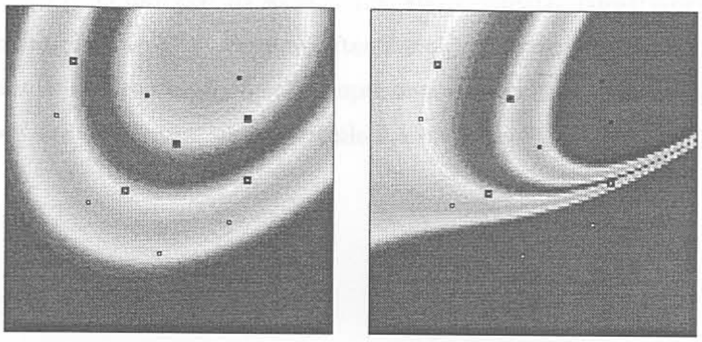

$f(x) = \text{sign}(\sum_i y_i \alpha_i k(x,x_i) + b)$
Classification
$y_1\alpha_1$ $y_2\alpha_2$ $y_3\alpha_3$ $y_4\alpha_4$
Weights
$k(x,x_1)$ $k(x,x_2)$ $k(x,x_3)$ $k(x,x_4)$
Comparison
Support vectors $x_1 \ldots x_4$
Input vector x

FIGURE 1.5: Architecture of a Support Vector Machine: The input vector $\mathbf{x}$ and the support vectors $\mathbf{x}_i$ (in this example optical digits) are non-linearly mapped (by Φ) into a feature space $\mathcal{F}$, where dot products between their mapped representations are computed. By the use of the kernel k, these two steps are in practice combined. The results are linearly combined by weights α_i found by solving a quadratic program. The linear combination is then fed into a decision function f, which determines the classification of $\mathbf{x}$.



(a) A Gaussian kernel $e^{-||\mathbf{x}_i - \mathbf{x}_j||^2}$.

(b) A polynomial kernel $(\mathbf{x}_i \cdot \mathbf{x}_j + 1)^5$.

FIGURE 1.6: Classifying with different kernel functions. The support vectors, with nonzero α_i, are shown with a double outline, and define the decision boundaries between the two classes.

Equation (1.28) results in a classifier that is a polynomial of degree p. Equation (1.29) results in a Gaussian radial basis function classifier, while (1.30) gives a particular kind of two-layer sigmoidal neural network. Figures 1.6(a) and 1.6(b) show the decision boundaries arising from both Gaussian radial basis function and polynomial kernels. More sophisticated kernels, like kernels generating splines or Fourier expansions, can be found in [44, 51, 59].

1.6 Concluding

This chapter presented the SVM optimisation problem: In training a non-linear SVM, the following quadratic problem needs to be maximised:

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.31}$$

subject to

$$0 \leq \alpha_i \leq 0, \quad i = 1, \ldots, l, \qquad \text{and} \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.32}$$

By constructing a matrix Q such that $(Q)_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ the problem at hand is to find

$$\begin{aligned} \max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) &= \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} \\ \text{subject to} \quad \boldsymbol{\alpha}^T \mathbf{y} &= 0 \\ \boldsymbol{\alpha} &\geq \mathbf{0} \\ C\mathbf{1} - \boldsymbol{\alpha} &\geq \mathbf{0} \end{aligned} \tag{1.33}$$

The following chapter is devoted to solving the above linearly constrained quadratic programming problem (1.33). The problem often involves a matrix with an extremely large number of entries, which make off-the-shelf optimisation packages unsuitable. Several SVM training methods are presented, and a detailed decomposition method of solving (1.33) is discussed.