An Introduction to Support Vector Machines

P. S. Sastry
Dept. Electrical Engineering
Indian Institute of Science, Bangalore 560012 *

1 Introduction

Support Vector Machines (SVMs) provide a learning technique useful in many areas such as Pattern Recognition (PR), regression estimation etc. The solution provided by SVMs for these problems is theoretically elegant, computationally efficient and is seen to be very effective in many large practical problems. The SVM approach grew out of some ideas from Statistical Learning Theory regarding controlling the generalization capabilities of learning machines. In any PR problem one is concerned with the question: how well does a classifier, designed or learnt based on certain finite training data, would perform on new patterns/data. This question of generalization has been addressed in many guises such as bias-variance tradeoff [1], overfitting etc. The issue is one of striking the correct balance between the accuracy obtained on a particular training set and the chosen 'capacity' of the learning machine (or algorithm) which is, roughly, an indicator of the ability of the machine to learn any arbitrary training set without error. Exploration of this question, which forms a major part of statistical learning theory, gave rise to the ideas of SVMs, which, arguably, offer an elegant way to effect proper level of generalization.

This chapter explains SVMs mainly as a technique for learning good pattern classifiers. The intent is to provide a tutorial introduction to SVMs aimed at practitioners and researchers interested in the general area of pattern recognition and regression. Sufficient details have been provided to help

*This material is published as a Chapter in J.C.Misra (Ed), 'Computing and Information Sciences: Recent Trends', Narosa Publishing House, New Delhi, 2003.

a reader, with no background in this area, to implement the SVM method on a PR problem. We start with a very brief overview of pattern recognition and then explain how one can use SVMs for this problem. It turns out that the main computational problem underlying the SVM methodology is the optimization of a quadratic cost function with linear constraints. We describe one specialized algorithm for solving this problem in detail. We also explain how these ideas can be used in regression and other related problems. As remarked earlier, the idea of SVM grew out of some results in statistical learning theory. Hence, we briefly discuss a few related concepts from statistical learning theory and explain why one can expect to get good generalization performance with SVMs. To make the material accessible to a large audience, we attempt to make the chapter largely self contained, and hence, keep to a minimum discussion of issues that may need specialized mathematical background.

2 Support Vector Machines for Pattern Classification

In this section we describe the SVM algorithm for 2-class Pattern Recognition (PR) problems. We begin with a brief introduction to PR and explain the SVM method in the simplest case, namely, the case of linearly separable pattern classes. We then explain how this idea can be used for learning general nonlinear discriminant functions.

2.1 The Pattern Recognition Problem

In a Pattern Recognition (PR) problem, the objective is to classify any given input pattern into one of finitely many classes [2]. The nature of input patterns as well as the classification labels to be output by the system depend on the specific application. For example, in Optical Character Recognition (OCR) applications, the input pattern is a two dimensional (grey-scale) image of a character symbol and the output label (also called the class label or class) is the name (or some other representation) of the character present in the image. In speech recognition systems, the input pattern is a (sampled version of the) time-varying voltage signal from a microphone and the output class label is the identity of the word (or any other speech units) that

compose the speech utterance. In a fingerprint based identity verification system, input is the image of the fingerprint pattern of a person (along with some other claim of identity such as a name) and the output class label is binary specifying whether or not the person is who he claims to be. While the first two examples represent multiclass problems, the last example is a two class problem. The methodology of SVMs is applicable for the 2-class PR problem. At the end of this subsection we indicate how we can tackle multiclass problems.

A general PR system can be viewed as implementing a two step procedure: feature extraction and classification. In the first step, the system extracts or measures some salient characteristics, called *features*, from the input pattern. In most cases the features are real numbers. Thus, though the input pattern may be in some arbitrary representation (such as an image), after feature extraction, each pattern is represented by a vector of real numbers, called the *feature vector*. Now the classification step is to assign a class label to each feature vector. Since we are considering only 2-class problems here, the classifier is a function that maps the set of feature vectors to the set, say, $\{-1, +1\}$. Any such function is called a decision rule. A good classifier or a decision rule is one that maps a feature vector to a class label that most often corresponds to the *true* class that the pattern represented by the feature vector belongs to. The feature extraction step is very much problem specific. For example, the features that we would like to measure from a fingerprint image so as to be able to make the correct identification decision would, in general, be different from those needed for correctly identifying a character from the image of a printed page. However, after fixing the set of features, design of a classifier admits some general procedures.

A general structure for a pattern classifier, that is relevant for our discussion of SVMs, is that of a discriminant function. Such a classifier or a decision rule can be specified as

> If $g(W, x) \geq 0$ then decide $x \in$ Class '+1'
> Else decide $x \in$ Class '-1'

where $g(\cdot, \cdot)$ is called a discriminant function, W is a parameter vector and x is the feature vector.[1] Having chosen the form of a discriminant function, the

[1]We always use x to denote feature vector and use subscripts when we want to denote different feature vectors. We make no distinction (such as boldface etc.) between vector

problem of obtaining a 'good' classifier is that of finding an 'optimal' value for the parameter vector. A special case which is relevant here is that of a linear discriminant function. Suppose the feature vector is n-dimensional. (That is, $x \in \Re^n$). Then a linear discriminant function is given by $g(w, b, x) = w^T x + b$. Here, the parameters of the discriminant function consist of a vector $w \in \Re^n$ and a scalar $b \in \Re$.

In most PR problems, it is not possible to design a classifier (which is required to map feature vectors to class labels) based on first principles. In most cases, one is provided with a set of examples, called training set or training patterns, of the form $\{(x_i, y_i),\ i = 1, \ldots, l\}$. Here each x_i is a feature vector whose 'true' class is given by y_i. Training samples are often obtained by getting some random patterns and having them classified by human experts. Now one can design a classifier by finding the values for the parameters of a discriminant function so that all training patterns are classified correctly (or with fewest errors). If one has statistically independent and representative set of training samples, then one can hope that a classifier that performs well on training set would also perform well on all the other unseen patterns of interest in the application. (See section 4 for some more discussion).

In a PR problem, the pattern classes are said to be linearly separable if there exists a linear discriminant function that would classify any pattern correctly. Thus, in a linearly separable case, there exist $w^* \in \Re^n$ and $b^* \in \Re$ such that the linear discriminant function with these parameters would classify all the training patterns correctly.

The methodology of SVMs offers an elegant and efficient technique for obtaining or learning general nonlinear discriminant functions using a given set of training patterns in a 2-class PR problem. The details are explained in the subsections to follow.

Since an SVM can be used only in a 2-class problem, the next obvious question is: how does one tackle a problem with, say, p classes? A simplest way to do this is to learn p discriminant functions. Each such discriminant function is trained to distinguish one of the classes from all others. After the learning is complete, the system functions as follows. Given a new feature vector, x, it is supplied to each of the discriminant functions. Then x is clas-

---

and scalar symbols. The distinction would be obvious from context. All vectors are column vectors and a superscript T denotes transpose.

sified as belonging to that class whose discriminant function has the highest positive value on x. If none of the discriminant functions has a positive value for the given value of x, then the system would reject x.

With this brief introduction to the PR problem we move on to the details of the SVM method.

2.2 SVMs for Linearly Separable Classes

We are given a finite training sample of patterns, $\{(x_i, y_i), i = 1, \ldots, l\}$, where $x_i \in \Re^n$ and $y_i \in \{-1, +1\}$, which is linearly separable. That is, we are considering n-dimensional real feature vectors[2] and the two class labels are denoted by $+1$ and -1. Since the pattern classes are linearly separable, there exist $w \in \Re^n$ and $b \in \Re$, such that for $i = 1, \ldots, l$,

$$\begin{aligned} w^T x_i + b &> 0, \quad \forall i \text{ such that } y_i = +1 \\ w^T x_i + b &< 0, \quad \forall i \text{ such that } y_i = -1. \end{aligned} \tag{1}$$

The hyperplane in $\Re^n$ described by w, b that satisfies (1) is called a separating hyperplane and is given by

$$w^T x + b = 0 \tag{2}$$

Since the classes are linearly separable, we can assume that there exists a separating hyperplane such that no training sample is on the hyperplane. That is why both the inequalities in (1) are strict. Since there are only finitely many samples, if w, b exist satisfying (1), then, by suitable scaling we can find w, b that satisfy

$$\begin{aligned} w^T x_i + b &\geq 1, \quad \forall i \text{ such that } y_i = +1 \\ w^T x_i + b &\leq -1, \quad \forall i \text{ such that } y_i = -1. \end{aligned} \tag{3}$$

The above set of inequalities can be written more compactly as

$$y_i[w^T x_i + b] \geq 1, \quad i = 1, \ldots, l. \tag{4}$$

Since the classes are linearly separable, there exist infinitely many separating hyperplanes. Figure 1 shows an example of two different separating hyperplanes. The hyperplane shown in fig. 1(b) is intuitively a 'better' separating

[2]In the sequel we use the terms feature vector and pattern vector interchangeably whenever there is no scope for confusion

hyperplane. This is because for this hyperplane the distance to the nearest pattern on either side is more and hence (intuitively) it can result in better generalization. We formalize this notion below.

For ensuring proper generalization, a central notion in SVM method is the concept of *optimal hyperplane*. If w, b satisfy (4), then our separating hyperplane is such that there is no training sample between the two parallel hyperplanes specified by $w^T x + b = 1$ and $w^T x + b = -1$. We call the distance between these two hyperplanes as the margin (of separation) of the separating hyperplane. The (perpendicular) distance between a point on the hyperplane $w^T x + b = 1$ and the hyperplane $w^T x + b = 0$ is $1/||w||$. Thus the margin is $\frac{2}{||w||}$. The margin is a good measure of how well a separating hyperplane separates the training patterns. Fig. 2 shows these notions schematically.

We define the optimal hyperplane to be the separating hyperplane with maximum margin. There is another way of looking at maximizing the margin. If we start with a separating hyperplane that satisfies (1) and scale w, b just enough to satisfy (4) then the training pattern closest to the separating hyperplane would be at a distance of $1/||w||$ from it.

The w^*, b^* corresponding to the optimal hyperplane constitute the SVM, which is our solution to the classification problem. The classification of any pattern vector, x, is now determined by the sign of $x^T w^* + b^*$.[3]

To maximize the margin we need to minimize $w^T w$. Hence the optimal hyperplane is a solution of the following constrained optimization problem.

$$\begin{aligned} \min \quad & \frac{1}{2} w^T w \\ \text{subject to} \quad & y_i [w^T x_i + b] \geq 1, \ i = 1, \ldots, l. \end{aligned} \tag{5}$$

It may be noted here that the variables for the optimization are w, b.

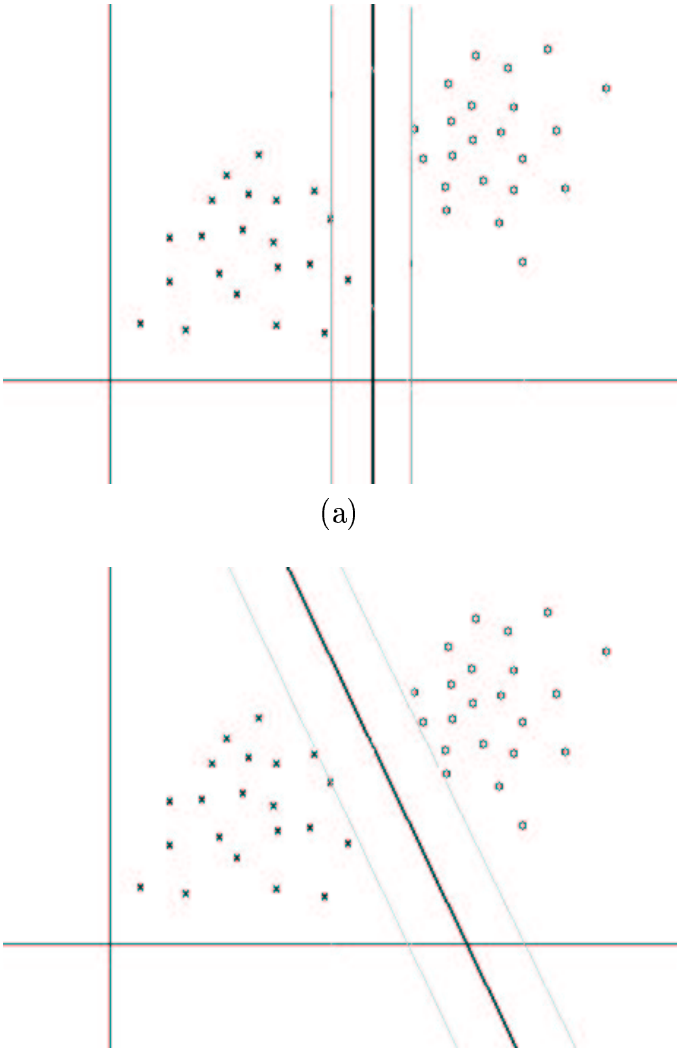
The Lagrangian[4] for this problem is

$$L(w, b, \mu) = \frac{1}{2} w^T w + \sum_{i=1}^{l} \mu_i \left[1 - y_i (w^T x_i + b)\right] \tag{6}$$

where $\mu = [\mu_1 \ldots \mu_l]^T \in \Re^l$ and μ_i are the Lagrange multipliers.

[3] We can also decide to reject any x (that is, assign no classification to it) if $|x^T w^* + b^*| < 1$. This can sometimes improve the classification accuracy on new patterns (at the cost of rejecting a few patterns).

[4] To make this part self-contained, a few results from optimization theory that are needed are stated in the Appendix



(a)

(b)

Figure 1: Examples of separating hyperplanes. The one in (b) is intuitively a 'better' separating hyperplane

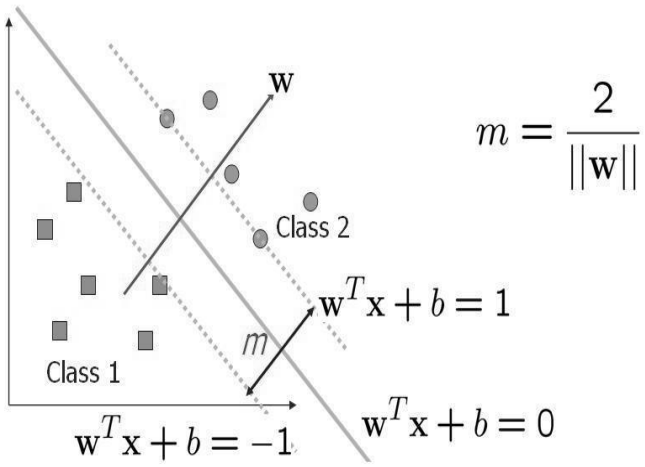


Figure 2: Margin of a separating hyperplane.

Since (5) is an optimization problem with quadratic cost function and linear constraints, w^*, b^* is a global solution of (5) if and only if there exist $\mu_i^*,\ i = 1, \ldots, l$, satisfying (see Theorem 1 in appendix)

$$\nabla_w\, L = w^* - \sum_{i=1}^{l} \mu_i^* y_i x_i \;=\; 0 \tag{7}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l} \mu_i^* y_i \;=\; 0 \tag{8}$$

$$1 - y_i(x_i^T w^* + b^*) \;\leq\; 0,\ \forall i, \tag{9}$$

$$\mu_i^* \;\geq\; 0 \tag{10}$$

$$\mu_i^*[1 - y_i(x_i^T w^* + b^*)] \;=\; 0,\ \forall i, \tag{11}$$

Conditions (7)-(11) are the Kuhn-Tucker conditions for problem (5). From (7) we see that the optimal hyperplane is a linear combination of training patterns. By (11), we must have $\mu_i^* = 0$ if $y_i(x_i^T w^* + b^*) > 1$. Let us call x_i a support vector if $y_i(x_i^T w^* + b^*) = 1$. Let S denote the set of indices of support vectors. Then, by (9) and (11), $\mu_i^* = 0$ if $i \notin S$. Now, from (7) we get

$$w^* = \sum_{i \in S} \mu_i^* y_i x_i. \tag{12}$$

Support vectors are those training patterns for which, at the optimal solution, the inequality constraints in (5) are satisfied by equality. Thus,

these are the samples closest to the separating hyperplane and hence, in a sense, are the most difficult patterns to classify. The optimal hyperplane is a linear combination of these support vectors and hence the name SVM for the method. Fig. 3 illustrates the optimal hyperplane and support vectors in an example.

If we can determine the optimal Langrange multipliers μ_i^* then we get w^* using (7) or equivalently (12). (While using (12), since we do not know support vectors a priori, we let the summation run over all i such that $\mu_i^* > 0$. In general, we can have $\mu_i^* = 0$ even when $i \in S$. However, this does not alter the w^* given by (12). Hence, from now on we think of support vectors as those corresponding to nonzero μ_i^*). We can also determine b^* as follows. Pick any i such that $\mu_i^* > 0$. Then, by (11), we must have $y_i(x_i^T w^* + b^*) = 1$ and hence

$$b^* = y_i - x_i^T w^* \tag{13}$$

because $y_i^2 = 1$.[5]

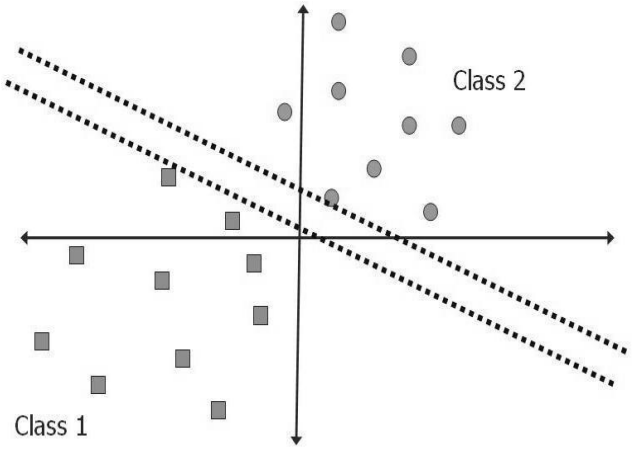
To find μ_i^* we will solve the dual of (5). We will use the formulation described in the appendix. In this notation, the dual function for (5) is given by

$$q(\mu) = \inf_{w,b} L(w, b, \mu) = \inf_{w,b} \{\frac{1}{2} w^T w + \sum_{i=1}^{l} \mu_i [1 - y_i(w^T x_i + b)]\}. \tag{14}$$

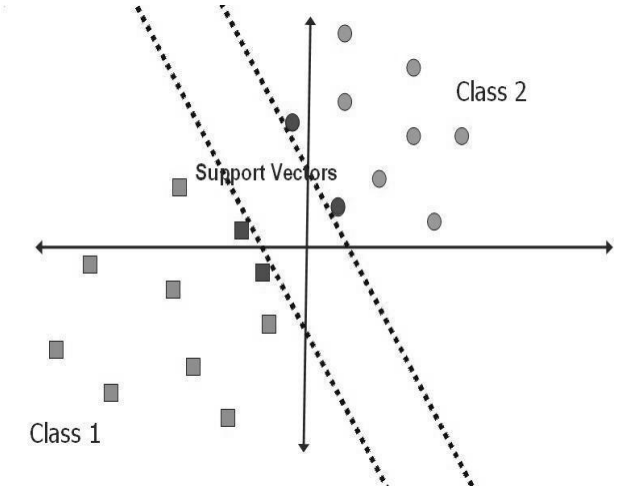
While finding infimum over $b \in \Re$, it is clear that $q(\mu)$ would be $-\infty$ unless we have $\sum \mu_i y_i = 0$. Since we need to maximize q, we would use this condition as a constraint. To find infimum with respect to $w \in \Re^n$, we need to solve $\nabla_w L = 0$ which is given by (7). Thus we get the infimum in (14) by substituting for w from (7) and taking $\sum \mu_i y_i = 0$. This gives us the dual of (5) as

$$\begin{aligned} \max \quad & q(\mu) = \sum_{i=1}^{l} \mu_i - \frac{1}{2} \sum_{i,j=1}^{l} \mu_i \mu_j y_i y_j x_i^T x_j \\ \text{subject to} \quad & \mu_i \geq 0, \ \forall i \\ & \sum_{i=1}^{l} \mu_i y_i = 0 \end{aligned} \tag{15}$$

[5] In practice, we would be using some numerical optimization technique for obtaining the μ_i^*'s. Hence, to take care of numerical errors, we may determine b^* as the average of the values given by (13) for all i such that $\mu_i^* > 0$.



(a)



(b)

Figure 3: Example of optimal hyperplane and support vectors. The (a) part shows a non-optimal separating hyperplane. The (b) part shows the optimal hyperplane for the same data. Here the examples shown in dark are the support vectors.

Here the optimization is over $\mu \in \Re^l$, where l is the number of training patterns. We note that in the dual problem, the training patterns appear only as inner products. The dual, given by (15), is once again an optimization problem with quadratic cost function and linear constraints.

From Theorem 2 in the appendix, w^*, b^* is a solution of the primal problem given by (5) and μ^* is a solution of the dual given by (15) if and only if the following are satisfied.

- C1. $\mu_i^* \geq 0$, $\sum\ y_i \mu_i^* = 0$.
- C2. $w^* = \sum_i\ \mu_i^* y_i x_i$, and $b^* = y_i - x_i^T w^*$ for i such that $\mu_i^* > 0$.
- C3. $y_i(x_i^T w^* + b^*) \geq 1$
- C4. $\mu_i^* > 0$ implies $y_i(x_i^T w^* + b^*) = 1$.

Thus in solving (15) we want to find $\mu^* \in \Re^l$ such that $\mu_i^* \geq 0$, $\sum \mu_i y_i = 0$ (that is, μ^* is feasible for (15)) and further

$$\begin{aligned} \mu_i^* = 0 \quad &\Rightarrow \quad y_i[x_i^T w^* + b^*] \geq 1 \\ \mu_i^* > 0 \quad &\Rightarrow \quad y_i[x_i^T w^* + b^*] = 1 \end{aligned} \tag{16}$$

Then with w^*, b^* as given by $C2$, we see that all conditions C1-C4 are satisfied. Thus the conditions given by (16) constitute the optimality conditions for checking whether a given feasible μ is optimal for (15). The interesting thing about these optimality conditions is that we can check for optimality of each component of μ using the corresponding sample pattern. We will discuss an algorithm for solving (15) in Section 3.

2.3 SVMs – Linearly Nonseparable case

Suppose the given training patterns are not linearly separable. Then we cannot find the optimal hyperplane because the optimization problem (5) has no feasible solution. In such a situation we can modify our objective by seeking a hyperplane to minimize errors. For this we modify our optimization problem as follows.

$$\begin{aligned} \min \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l}\ \xi_i \\ \text{subject to} \quad & y_i[w^T x_i + b] \geq 1 - \xi_i,\ i = 1, \ldots, l \\ & \xi_i \geq 0,\ i = 1, \ldots, l. \end{aligned} \tag{17}$$

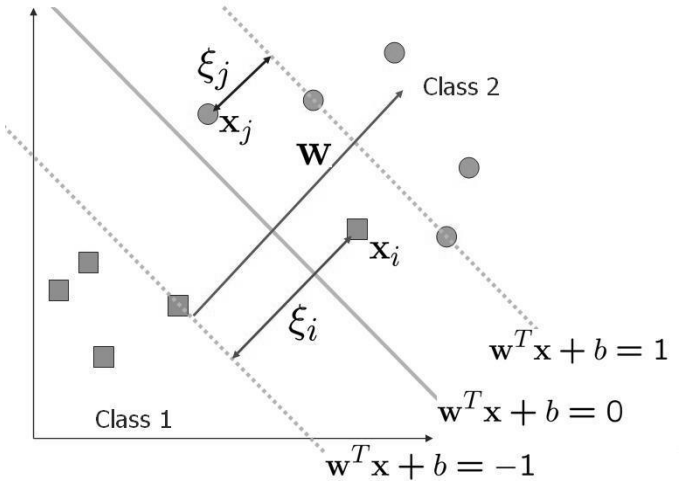


Figure 4: Illustration of slack variables and margin violations. For the hyperplane shown, we would have both $\xi_i > 0$ and $\xi_j > 0$. Of the two samples, only x_i would be misclassified by this hyperplane.

Here $\xi_i,\ i = 1, \ldots, l$ are positive (so called) slack variables and the optimization is over w, b, ξ_i. Suppose w^*, b^*, ξ_i^* are the optimal values of variables for the above problem. Then, the classification of a new pattern x would be determined, as earlier, by sign of $x^T w^* + b^*$. Thus $\xi_i^* > 0$ implies that the optimal hyperplane (that is, the solution of (17)) is not able to separate training patterns as demanded by (4). Further, if $\xi_i^* > 1$ then we are actually making an error in classifying the training pattern x_i. This is illustrated in fig. 4.

Thus, $\sum\ \xi_i$ is a good measure of the error made and C, which is a parameter to be chosen by the user, controls how much error we can tolerate. High value of C would correspond to high penalty for errors. (We always take $C > 0$). (It may be noted that the optimization problem given by (5) is essentially a limiting case as $C \to \infty$). Since we make errors only when $\xi_i \geq 1$, $(\sum\ \xi_i)^k$ for any positive integer k is a good penalty term. Using such a penalty would result in a convex optimization problem for any k. For $k = 1, 2$, we would get a quadratic optimization problem. In (17) we have chosen $k = 1$. The advantage is that, in this case, neither ξ_i nor the corresponding Lagrange multipliers appear in the dual of (17). The dual

function for (17) is given by (see Appendix)

$$q(\mu,\lambda) = \inf_{w,b,\xi} \{\frac{1}{2}w^T w + C(\sum_{i=1}^{l} \xi_i) + \sum_{i=1}^{l} \mu_i[1-\xi_i - y_i(w^T x_i + b)] - \sum_{i=1}^{l} \lambda_i \xi_i\} \tag{18}$$

where λ_i are the Lagrange multipliers corresponding to the constraints $-\xi_i \leq 0$. As earlier, we would need $\sum \mu_i y_i = 0$ to ensure that the infimum is not $-\infty$. Similarly, we get a term $\sum(C - \lambda_i - \mu_i)\xi_i$ and hence we need to ensure $C = \lambda_i + \mu_i, \ \forall i$. Using these and taking infimum with respect to w by substituting $w = \sum \mu_i y_i x_i$, we get the dual of (17) as

$$\begin{aligned} \max \quad & q(\mu) = \sum_{i=1}^{l} \mu_i - \frac{1}{2} \sum_{i,j=1}^{l} \mu_i \mu_j y_i y_j x_i^T x_j \\ \text{subject to} \quad & 0 \leq \mu_i \leq C, \ \forall i \\ & \sum_{i=1}^{l} \mu_i y_i = 0 \end{aligned} \tag{19}$$

The only difference between the two optimization problems given by (15) and (19) is that μ_i are bounded from both sides.[6] Thus by the nature of the penalty term we employed, we get essentially the same optimization problem to solve even when patterns are not linearly separable. The Kuhn-Tucker conditions for (17) are the following.

- K1. $w - \sum_{i=1}^{l} \mu_i y_i x_i = 0$.
- K2. $\sum_{i=1}^{l} \mu_i y_i = 0$.
- K3. $C - \mu_i - \lambda_i = 0$.
- K4. $1 - \xi_i - y_i(w^T x_i + b) \leq 0$.
- K5. $\xi_i \geq 0$, $\mu_i \geq 0$, $\lambda_i \geq 0$, $\forall i$.
- K6. $\mu_i[1 - \xi_i - y_i(w^T x_i + b)] = 0$, $\forall i$.
- K7. $\lambda_i \xi_i = 0$, $\forall i$.

[6]Since λ_i do not appear explicitly in the dual function, all we need to ensure regarding them is $\lambda_i \geq 0$ and $\lambda_i + \mu_i = C$. So, if we ensure $0 \leq \mu_i \leq C$ then by taking $\lambda_i = C - \mu_i$ we can satisfy all constraints on λ_i. This is the reason for the upper bound on μ_i.

If $\mu_i = 0$ then by K3, $\lambda_i > 0$ and hence by K7 $\xi_i = 0$. Now, by K4, $y_i(w^T x_i + b) \geq 1$. If $0 < \mu_i < C$ then $\lambda_i > 0$ and hence, by K7 and K6, $y_i(w^T x_i + b) = 1$. If $\mu_i = C$ then by K4 and K6, $y_i(w^T x_i + b) = 1 - \xi_i \leq 1$. Putting all this together, as in the previous subsection, we can derive conditions for μ_i^* to be a solution of (19) as follows:

$$0 \leq \mu_i^* \leq C, \quad \text{and} \quad \sum_{i=1}^{l} \mu_i^* y_i = 0 \tag{20}$$

and

$$\begin{aligned} \mu_i^* = 0 &\Rightarrow y_i[x_i^T w^* + b^*] \geq 1 \\ 0 < \mu_i^* < C &\Rightarrow y_i[x_i^T w^* + b^*] = 1 \\ \mu_i^* = C &\Rightarrow y_i[x_i^T w^* + b^*] \leq 1 \end{aligned} \tag{21}$$

where w^*, b^*, which will be solution to the primal problem given by (17), are given by $w^* = \sum_{i=1}^{l} y_i \mu_i^* x_i$ and $b^* = y_j - x_j^T w^*$ for some j such that $0 < \mu_j^* < C$.

Thus in the case where the given pattern vectors are not linearly separable, the SVM is obtained by solving (19) (for which the conditions for checking optimality are given by (21)) which is the dual of (17). This solution is the 'best' linear discriminant function in the sense that (depending on C) it gives a hyperplane with 'least' error on the training set. However, in many applications, this error may be unacceptably high. In such cases we need to find a suitable nonlinear discriminant function which is explained in the next subsection.

2.4 Nonlinear SVMs

There is another way of viewing pattern sets that are not linearly separable. Suppose we transform the given pattern vectors as $z_i = \phi(x_i)$ where $\phi : \Re^n \rightarrow \mathcal{H}$ is some nonlinear function. We assume $\mathcal{H}$ to be some space on which an inner product is defined. (Generally we take $\mathcal{H} = \Re^m$ with $m > n$). We may be able to find a suitable ϕ such that the resulting vectors z_i may be linearly separable in $\mathcal{H}$ even though the original patterns x_i are not linearly separable in $\Re^n$. For example if $n = 2$ and x_i are separable only by a quadratic discriminant function, then, if we use $\phi : \Re^2 \rightarrow \Re^5$ with $\phi(v_1, v_2) = [v_1 \;\; v_2 \;\; v_1^2 \;\; v_2^2 \;\; v_1 v_2]^T$, the resulting z_i would be linearly separable.

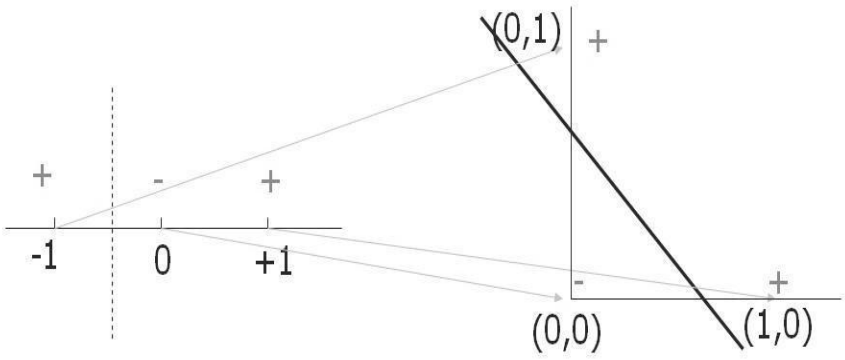


Figure 5: Example of a non-separable data set in one dimensional feature space. By transforming the patterns into $\Re^2$, we get linear separability.

Thus finding a quadratic discriminant function with the original training set is the same as finding a linear discriminant function with the transformed training set. In figs. 5, 6 and 7 we show some examples of this notion of transforming the patterns into some other space so that the classes become linearly separable.

However, this, by itself, is not very attractive for two reasons. The first reason is that, this naive method would be computationally very expensive. Suppose $x_i \in \Re^n$. If we want to find a discriminant function that is a p^{th} degree polynomial in this space, then, to use the above idea, we have to employ a ϕ function whose range space would have dimension of the order of n^p. Computing all the z_i, computing inner products in such a high dimensional space etc. would all be computationally expensive. The second reason why this method is not attractive is that, the learnt hyperplane in the high dimensional space may have poor generalization abilities. To learn a hyperplane in n-dimensional space needs learning $n+1$ parameters from the given samples. Hence, we should expect that we need correspondingly larger number of sample patterns as the dimension of the space becomes larger. (See Section 4 for further discussion of this issue). The attractiveness of SVM methodology is that it is able to address both these problems in an effective manner. In this subsection we explain how we are able to get around the problem of computational inefficiency even though we effectively find an optimal hyperplane in a very high dimensional space in this manner.

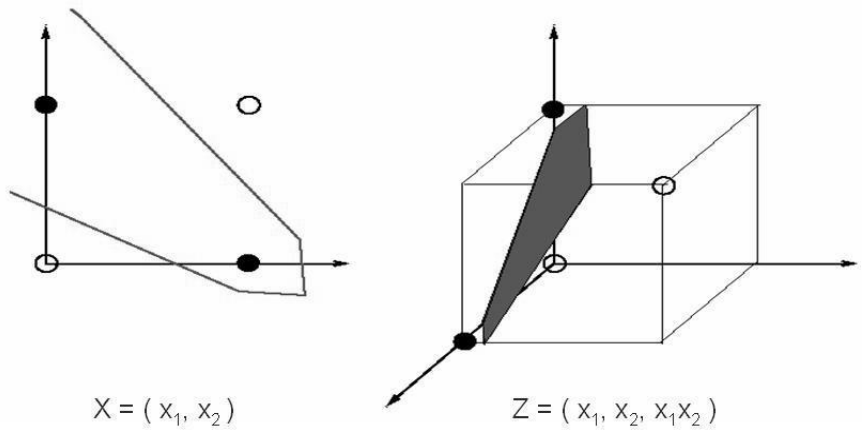


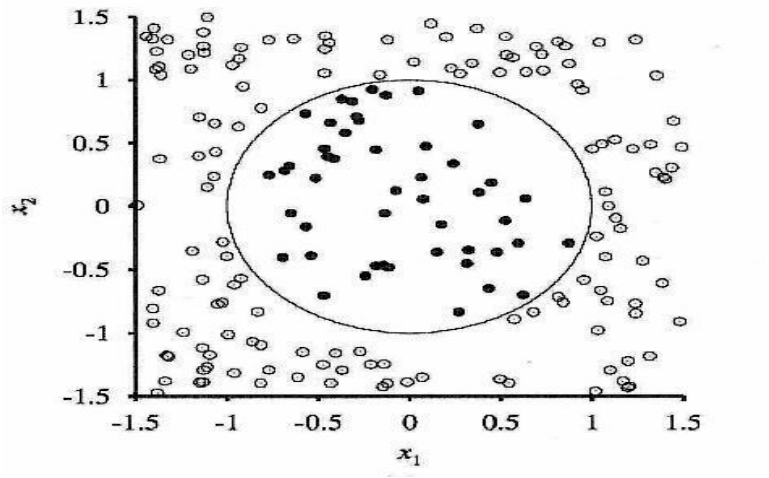
Figure 6: Example of a non-separable data set in $\Re^2$: the XOR problem. By transforming the patterns into $\Re^3$, we get linear separability.

We can obtain the optimal hyperplane for the transformed training set, $\{(z_i, y_i),\ i = 1, \ldots, l\}$, by solving (15) with x_i replaced by z_i. Thus the new optimization problem (in the dual formulation) now is:

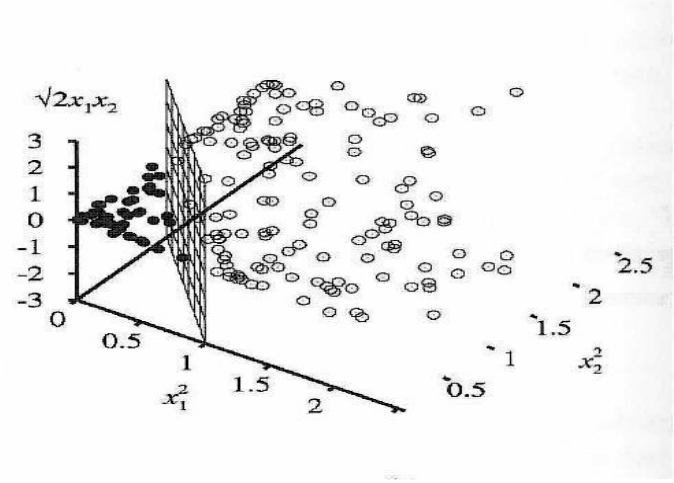
$$\begin{aligned} \max \quad & q(\mu) = \sum_{i=1}^{l} \mu_i - \frac{1}{2} \sum_{i,j=1}^{l} \mu_i \mu_j y_i y_j z_i^T z_j \\ \text{subject to} \quad & \mu_i \geq 0,\ \forall i \\ & \sum_{i=1}^{l} \mu_i y_i = 0 \end{aligned} \tag{22}$$

We note that we could have used the formulation of Section 2.3 and then the only difference to (22) is that the nonnegativity constraint on μ_i is replaced by $0 \leq \mu_i \leq C$ (see (19)). Because of the fact that the (transformed) pattern vectors appear only as an inner product in the objective function, it turns out (as explained below) that we can solve (22) without ever explicitly calculating z_i.

Define a function $K : \Re^n \times \Re^n \to \Re$ by $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. This will be called a kernel function. If we have a suitable kernel function (which is much less expensive to compute than $\phi(x_i)^T \phi(x_j)$) then we need not explicitly calculate z_i because we can substitute $K(x_i, x_j)$ for $z_i^T z_j$ in (22). This may not appear to be significant because the w^* of the optimal hyperplane obtained by solving (22) will be in space $\mathcal{H}$. However, we do not need to explicitly



(a)



(b)

Figure 7: Another example of transforming the patterns for getting linear separability. Part (a) shows the data (in $\Re^2$) where one class is inside a circle and another is outside. We need a quadratic function to separate the two classes. Part (b) shows a suitably transformed data that is linearly separable.

know w^* either. The remarkable feature of the SVM method is that by the use of kernel functions we can completely avoid the need for working in the range space of ϕ.

As we have seen, the optimal hyperplane is a linear combination of support vectors. Let $S = \{i : \mu_i^* > 0\}$ denote the indices of strictly positive Lagrange multipliers. Then we have $w^* = \sum_{i \in S} \mu_i^* y_i z_i$. Now to find the classification of any new pattern vector, x, we need to calculate $f(x) = \phi(x)^T w^* + b^* = \sum \mu_i^* y_i \phi(x_i)^T \phi(x) + b^*$. We know that $b^* = y_j - \phi(x_j)^T w^*$, for some j such that $0 < \mu_j^* < C$. Thus we can calculate $f(x)$ as

$$f(x) = \sum_{i \in S} \mu_i^* y_i K(x_i, x) + y_j - \sum_{i \in S} \mu_i^* y_i K(x_i, x_j). \tag{23}$$

The classification of x would be $+1$ if $f(x) > 0$ and it will be -1 otherwise. Hence, all we need for learning the nonlinear SVM is to solve the optimization problem given by (22) with $z_i^T z_j$ substituted by $K(x_i, x_j)$. (It may be noted that the optimization in (22) is over $\Re^l$ irrespective of the dimension of $\mathcal{H}$). Once we obtain the optimal Lagrange multipliers, $\mu_i^*,\ i = 1, \ldots, l$, we can completely specify the SVM by the nonzero Lagrange multipliers and their corresponding training patters. Given this, we can decide on the classification of any new pattern using (23). Thus, by the use of kernel functions we can effectively find the separating hyperplane in the high dimensional $\mathcal{H}$ (which is the range space of ϕ) without having to compute either $z_i = \phi(x_i)$ or w which are in $\mathcal{H}$. This would mean that $\mathcal{H}$ can even be infinite dimensional!

How do we choose kernel functions? The function $K(\cdot, \cdot)$ should be such that there is a vector space $\mathcal{H}$ and a $\phi : \Re^n \to \mathcal{H}$ such that $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. A simple example is $K(x_i, x_j) = (x_i^T x_j)^2$. Suppose $n = 2$. We can choose $\mathcal{H} = \Re^3$ and a $\phi : \Re^2 \to \Re^3$ as $\phi(a, b) = [a^2 \ \ \sqrt{2}ab \ \ b^2]^T$. Now let $U = [u_1 \ u_2]^T$ and $V = [v_1 \ v_2]^T$ be two vectors in $\Re^2$. Then we have $\phi(U)^T \phi(V) = u_1^2 v_1^2 + 2u_1 u_2 v_1 v_2 + u_2^2 v_2^2 = (U^T V)^2 = K(U, V)$. It is easy to see that for a given kernel function neither the choice of $\mathcal{H}$ nor that of ϕ is unique. In this example, we could have used $\phi : \Re^2 \to \Re^4$ given by $\phi(a, b) = [a^2 \ \ ab \ \ ab \ \ b^2]^T$. However, we need the space $\mathcal{H}$ to be endowed with an inner product.

The necessary and sufficient condition for a kernel function to have the required property is given by the so called Mercer's theorem.

Mercer's Theorem: Given a function $K : \Re^n \times \Re^n \to \Re$, there ex-

ists a mapping ϕ and an expansion $K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$ if and only if for every real valued function g on $\Re^n$ with $\int g(x)^2 \, dx < \infty$, we have

$$\int K(x_1, x_2) \; g(x_1) \; g(x_2) \; dx_1 \; dx_2 \geq 0.$$

We can easily verify that this condition is satisfied by the kernel function $K(U, V) = (U^T V)^p$ where $U = [u_1 \ldots u_n]^T \in \Re^n$, $V = [v_1 \ldots v_n]^T \in \Re^n$ and p is a positive integer. We need to show that

$$\int (\sum_{i=1}^{n} u_i v_i)^p \; g(U) \; g(V) \; dU \; dV \geq 0 \tag{24}$$

The typical term in the multinomial expansion of $(\sum u_i v_i)^p$ contributes a term of the form

$$\frac{p!}{r_1! r_2! \ldots (p - r_1 - \ldots)!} \int \; u_1^{r_1} u_2^{r_2} \ldots v_1^{r_1} v_2^{r_2} \ldots \; g(U) \; g(V) \; dU \; dV$$

to the LHS of (24). This term factorises as

$$\frac{p!}{r_1! r_2! \ldots (p - r_1 - \ldots)!} \; (\int \; u_1^{r_1} u_2^{r_2} \ldots \; g(U) \; dU)^2 \geq 0$$

From the above, it is clear that any kernel of the form $K(x_1, x_2) = \sum_{p=0}^{\infty} c_p \; (x_1^T x_2)^p$ where $c_p \geq 0$ are real coefficients and the series is uniformly convergent, satisfies the Mercer's condition. A special case of this is

$$K_p(x_1, x_2) = (1 + x_1^T x_2)^p = 1 + p(x_1^T x_2) + \ldots + (x_1^T x_2)^p. \tag{25}$$

This is one of the standard kernels used and is called the polynomial kernel with degree p. This results in a nonlinear SVM that represents separating surface given by a polynomial of degree p in the original pattern space.

Another kernel that is popular in applications is the Gaussian kernel defined by

$$K_G(x_1, x_2) = \exp\left(\frac{-(x_1 - x_2)^T (x_1 - x_2)}{2\sigma^2}\right). \tag{26}$$

By writing $\exp[-(x_1 - x_2)^T (x_1 - x_2)] = \exp[-x_1^T x_1] \exp[-x_2^T x_2] \exp[2 x_1^T x_2]$ and then using $\exp[2 x_1^T x_2] = \sum_{p=0}^{\infty} (2 x_1^T x_2)^p / p!$, we can easily show that this kernel also satisfies the Mercer condition.

A nonlinear SVM with a Gaussian kernel results in a classifier that is structurally equivalent to a Radial Basis Function (RBF) network. To see this, recall that with a nonlinear SVM, the classification of a new pattern is determined by the sign of $f(x)$ given by (23). With a Gaussian kernel we have

$$f(x) = \sum_{i \in S} \mu_i y_i \exp\left(\frac{||x_i - x||^2}{2\sigma^2}\right) + b \tag{27}$$

where x_i are the support vectors. It is easily seen that $f(x)$ given by (27) is the output of RBF network having Gaussian basis functions with centers x_i, widths σ and the weights into the linear output neuron being $\mu_i y_i$ and the bias of the output neuron being b. With an RBF network, learning centers is a very difficult problem. The SVM method offers an elegant solution to this and the centers turn out to be the support vectors.

Now we can sum up the complete SVM method as follows. Given the training set of patterns $\{(x_i, y_i),\ i = 1, \ldots l\}$, we need to solve the optimization problem specified by (22) with $z_i^T z_j$ replaced by $K(x_i, x_j)$ for some suitable kernel function. After obtaining $\mu_i^*,\ i = 1, \ldots, l$, the SVM is completely specified by the nonzero Lagrange multipliers and the corresponding training patterns. Given any new pattern x, its classification is determined by the sign of $f(x)$ given by (23). If, with the chosen kernel function, we are not sure of perfect separation even in the new high dimensional feature space, then we can use the technique of Section 2.3 with a suitable penalizing constant C. The only difference in the optimization problem is that the bound constraints on μ_i would now be $0 \leq \mu_i \leq C$. It is easy to see that the optimality conditions given by (21) are expressible in terms of $f(x_i)$.

3 The Optimization Problem

Learning an SVM involves the following. We are given a training sample $\{(x_i, y_i),\ i = 1, \ldots, l\}$. We need to find μ_i for solving the following optimization problem.

$$\begin{aligned} \max \quad & q(\mu) = \sum_{i=1}^{l} \mu_i - \frac{1}{2} \sum_{i,j=1}^{l} \mu_i \mu_j y_i y_j K(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \mu_i \leq C,\ \ i = 1, \ldots, l, \end{aligned}$$

$$\sum_{i=1}^{l} \mu_i y_i = 0 \tag{28}$$

The conditions for optimality for the above problem can be stated as

$$\begin{aligned} \mu_i = 0 &\Rightarrow y_i f(x_i) \geq 1 \\ 0 < \mu_i < C &\Rightarrow y_i f(x_i) = 1 \\ \mu_i = C &\Rightarrow y_i f(x_i) \leq 1 \end{aligned} \tag{29}$$

where $f(x)$ is given by

$$f(x) = \sum_{i \in S} \mu_i y_i K(x_i, x) + b. \tag{30}$$

Here, $S = \{i \ : \ \mu_i > 0\}$ and b is given by

$$b = y_j - \sum_{i \in S} \mu_i y_i K(x_i, x_j) \tag{31}$$

for some j such that $0 < \mu_j < C$. Our optimization problem is over $\Re^l$ where l is the number of training examples. Each Lagrange multiplier, μ_i, is associated with an example, (x_i, y_i). The structure of the optimality conditions given by (29) is such that we can talk about different μ_i (or different examples) violating optimality conditions.

The problem specified by (28) is a quadratic programming (QP) problem with bound constraints on variables and one equality constraint. However, using any standard QP technique directly may be inefficient. The objective function of (28) is $q(\mu) = \mu^T e - \frac{1}{2}\mu^T H \mu$ where e is vector all of whose components equal 1 and H is a $l \times l$ matrix with $(i,j)^{th}$ element $H_{ij} = y_i y_j K(x_i, x_j)$. Since l is generally large, even evaluating the quadratic form at each iteration may be expensive. There are some efficient ways of solving this QP problem that exploit the special characteristics of this problem.

3.1 Sequential Minimal Optimization

The main idea of the SVM method is that while the number of examples may be large, the number of support vectors would be small. Thus we expect that the optimal value of a large number of μ_i would be zero. The QP problem of SVM involves the quadratic form of a matrix with l^2 elements. However, the

quadratic form is unchanged if we remove rows and columns of the matrix H that correspond to zero Lagrange multipliers. This gives rise to the idea of 'chunking'. Chunking based algorithms solve a series of QP subproblems as follows. At each iteration we solve a QP subproblem that includes all nonzero Lagrange multipliers of the previous iteration plus some M more μ_i that correspond to the 'worst' violators of the optimality conditions (29), for some fixed M. (If there are fewer than M violators, all of them are included). The size of these QP subproblems may both grow and shrink as the iterations proceed and in the last iteration the size of the QP subproblem would be same as the number of support vectors. Even chunking may involve solving large QP problems in some iterations. A further simplification is possible by choosing a fixed size for each subproblem. Here at each iteration we select, say, M (or fewer) number of μ_i for optimization based on which μ_i violate (29). Each of the subproblems would be solved using a standard numerical method for QP. As long as, at each iteration we include at least one μ_i that violates (29), and each step improves the overall objective function and maintains the feasibility of all μ_i at all iterations, this process of solving a series of QP subproblems will converge to the optimal solution.

Sequential minimal optimization (SMO) is a technique based on this idea of solving a series of fixed size QP subproblems. Here each QP subproblem involves only two variables. This is the smallest size for the subproblem because we need to maintain the overall feasibility of all μ_i at each step and the μ_i have to satisfy an equality constraint. Since a two variable QP problem can be solved analytically, at each iteration we can update the variables using the analytically derived equations (rather than using a numerical routine) which makes the method very fast. SMO algorithm uses some heuristics to select the two Lagrange multipliers to be optimized at each iteration so that the overall objective function is improved and the feasibility of all μ_i is maintained. The overall algorithm turns out to be very simple to program and is seen to be very efficient in applications.

Each iteration of the SMO algorithm contains two main computations: selection of the two Lagrange multipliers and updating their values using analytically derived equations for solving a two variable constrained QP problem. We first derive the expressions for updating the two chosen Lagrange multipliers. To keep the notation simple, we will refer to the two Lagrange multipliers selected for optimization as μ_1 and μ_2. Let $K_{ij} = K(x_i, x_j)$.

(The values of K_{ij} do not change during the algorithm and hence they can be computed beforehand and stored. Also note that $K_{ij} = K_{ji}$). While describing the algorithm, we will distinguish between the old and new values of Lagrange multipliers by using a superscript as appropriate.

For maintaining the feasibility of solution, we need to ensure $\sum \mu_i \; y_i = 0$ at all times. Hence we must have (note that we are changing only μ_1 and μ_2)

$$\mu_1^{old} y_1 + \mu_2^{old} y_2 = \mu_1^{new} y_1 + \mu_2^{new} y_2. \tag{32}$$

Since $y_1, y_2 \in \{+1, -1\}$, we can rewrite the above as

$$\mu_1^{old} + y_1 y_2 \mu_2^{old} = \mu_1^{new} + y_1 y_2 \mu_2^{new} = \gamma, \quad \text{say.} \tag{33}$$

To maintain feasibility of μ_i's after updating μ_1, μ_2, we should ensure that μ_1^{new}, μ_2^{new} satisfy (33) and also satisfy $0 \leq \mu_1^{new}, \mu_2^{new} \leq C$. With a little algebra, this implies that we should satisfy

$$L \leq \mu_2^{new} \leq H, \quad \text{and} \quad \mu_1^{new} = \gamma - y_1 y_2 \mu_2^{new} \tag{34}$$

where L, H, γ are given by:

If $y_1 \neq y_2$ then

$$\begin{aligned} L &= \max(0, \; \mu_2^{old} - \mu_1^{old}) \\ H &= \min(C, \; C + \mu_2^{old} - \mu_1^{old}) \\ \gamma &= \mu_1^{old} - \mu_2^{old} \end{aligned} \tag{35}$$

If $y_1 = y_2$ then

$$\begin{aligned} L &= \max(0, \; \mu_2^{old} + \mu_1^{old} - C) \\ H &= \min(C, \; \mu_2^{old} + \mu_1^{old}) \\ \gamma &= \mu_1^{old} + \mu_2^{old} \end{aligned} \tag{36}$$

Now our task is to find values for μ_1^{new}, μ_2^{new}, that satisfy (34) with L, H, γ given by (35) or (36), such that we maximize $q(\mu)$ with respect to the chosen variables. Since, $\mu_j, j = 3, \ldots, l$ are left unchanged, q is to be viewed as a function of two variables only. With a little abuse of notation we use the

same symbol q for all such functions. By simple algebraic manipulations, we can write q as a function of μ_1, μ_2 as

$$q(\mu_1,\mu_2) = \mu_1+\mu_2-\frac{1}{2}K_{11}\mu_1^2-\frac{1}{2}K_{22}\mu_2^2-sK_{12}\mu_1\mu_2-\mu_1 y_1 v_1-\mu_2 y_2 v_2+W_c \quad (37)$$

where $K_{ij} = K(x_i, x_j)$, $s = y_1 y_2$, W_c is a term that depends only on μ_j, $j = 3, \ldots, l$, and $v_i = \sum_{j=3}^{l} y_j \mu_j^{old} K_{ij}$, $i = 1, 2$. By (34) we must have $\mu_1 = \gamma - s\mu_2$. Hence we can rewrite q as a function of only μ_2 as

$$\begin{aligned} q(\mu_2) &= \gamma + \mu_2(1-s) - \frac{1}{2}K_{11}(\gamma - s\mu_2)^2 - \frac{1}{2}K_{22}\mu_2^2 - sK_{12}\mu_2(\gamma - s\mu_2) \\ &\quad -y_1 v_1(\gamma - s\mu_2) - y_2 v_2 \mu_2 + W_c \end{aligned} \quad (38)$$

To maximize q with respect to μ_2 we set $dq/d\mu_2 = 0$ which gives (using $s^2 = 1$ and $y_1 s = y_2$)

$$s\gamma(K_{11} - K_{12}) + y_2(v_1 - v_2) + 1 - s - \mu_2(K_{11} + K_{22} - 2K_{12}) = 0 \quad (39)$$

For the q function to attain a maximum at a μ_2 satisfying (39), we must have the second derivative negative. Let $\eta = d^2q/d\mu_2^2$, which is given by

$$\eta = -(K_{11} + K_{22} - 2K_{12}). \quad (40)$$

For now, let us assume $\eta < 0$. Writing, from (30), $f^{old}(x) = \sum \mu_j^{old} y_j K(x_i, x) + b^{old}$, we can show that

$$v_1 - v_2 = f^{old}(x_1) - f^{old}(x_2) + y_2\mu_2^{old}(K_{11}+K_{22}-2K_{12}) + y_1\gamma(K_{21}-K_{11}). \quad (41)$$

From (39),(40) and (41), we see that $q(\mu_2)$ attains a maximum at μ_2' where

$$\mu_2' = \mu_2^{old} - \frac{y_2(E_1 - E_2)}{\eta} \quad (42)$$

where $E_i = f^{old}(x_i) - y_i, \quad i = 1, 2$. Now using (42) and (34), we get the expressions for μ_1^{new}, μ_2^{new} as

$$\begin{aligned} \mu_2^{new} &= H \quad \text{if } \mu_2' \geq H \\ &= \mu_2' \quad \text{if } L < \mu_2' < H \\ &= L \quad \text{if } \mu_2' \leq L \end{aligned}$$

$$\mu_1^{new} = \mu_1^{old} + s(\mu_2^{old} - \mu_2^{new}) \quad (43)$$

The SMO algorithm is an iterative algorithm that has two loops. The outer loop chooses the first Lagrange multiplier, the inner one the second Lagrange multiplier. Inside the two loops we compute new values of the two selected Lagrange multipliers using (43). (For this we need to compute L, H, γ, E_i etc.) Now to complete the algorithm we need to state how the two Lagrange multipliers (referred to as μ_1, μ_2 above) are to be selected and what to do if η given by (40) is nonnegative. Essentially any pair of Lagrange multipliers that currently violate the optimality conditions (given by (29)) can be chosen as long as we ensure that all possible pairs would eventually be considered again and again. The SMO algorithm has a hierarchy of heuristics to guide the choice of the two Lagrange multipliers which are useful for improving the speed of convergence. We describe below a simple set of choices. For this discussion, we say a Lagrange multiplier is bound if its (current) value is either 0 or C; otherwise it is nonbound.

To choose the first Lagrange multiplier, in the outer loop we go over all the Lagrange multipliers till we find one for which the optimality conditions given by (29) are violated. (If we cannot find such a Lagrange multiplier then we have the optimal μ_i's and hence the desired SVM). What is normally done is to loop through all the μ_i's for the first time. Then we keep looping through only the nonbound Lagrange multipliers. If at some iteration we cannot find a nonbound multiplier that violates the optimality conditions, then we loop through all μ_i's one more time and so on. Choosing the first multiplier fixes E_1 in (42). Now, in the inner loop we select the second multiplier (which is also one that does not currently satisfy the optimality conditions) such that $E_1 - E_2$ is maximized. For this chosen pair, if η given by (40) is negative then we can proceed and update the values as given by (43). Even if η is nonnegative we can often make progress with the chosen pair. Note that the constraints given by (34) specify a line in the μ_1–μ_2 space as the feasible region. So, if η is nonnegative we calculate the value of q at the two ends of this line segment and if these two are not same we take μ_1^{new}, μ_2^{new} to be the values corresponding to the end point of this line segment that has larger value for q. If the value of q at both end points is same then we leave this second Lagrange multiplier and proceed with the inner loop for another choice of the second multiplier. If no choice exits, then we change the choice of first Lagrange multiplier by proceeding with the outer loop.

In the algorithm we need to keep checking for optimality conditions.

While calculating expressions such as $\sum \mu_j y_j K_{ij}$, we should note that, between successive iterations only two of the μ_j's would have their value changed. Hence many caching type techniques can be used to get efficient implementation. Another point to note is that the algorithm (43) does not give us the new value of b even though it is needed for checking optimality conditions. With a little algebra, we can derive the following equations for getting the new value of b efficiently after obtaining μ_1^{new} and μ_2^{new}. Suppose μ_1^{new} is nonbound. Then b_1 given below would be the new value of b.

$$b_1 = E_1 + y_1(\mu_1^{new} - \mu_1^{old})K_{11} + y_2(\mu_2^{new} - \mu_2^{old})K_{12} + b^{old} \tag{44}$$

If μ_2^{new} is nonbound then b_2 given below would be the new value for b.

$$b_2 = E_2 + y_1(\mu_1^{new} - \mu_1^{old})K_{12} + y_2(\mu_2^{new} - \mu_2^{old})K_{22} + b^{old} \tag{45}$$

If both μ_1^{new} and μ_2^{new} are nonbound then b_1 and b_2 would be the same. If both μ_1^{new} and μ_2^{new} are bound then all values between b_1 and b_2 would be consistent with the optimality conditions. In such a case the SMO algorithm uses the average of the two.

In the SMO algorithm we need to check repeatedly for equality and inequality constraints. (For example, we need to check whether $f(x_i) = 1$ for some x_i, whether $\mu_i = C$ and so on.) For all such comparisons we need to allow for some numerical error. Hence, we keep a parameter ϵ and assume any constraint to be satisfied if it holds to within ϵ. Too small a value of ϵ would impair the quality of solution obtained while too large a value may mean we go through a large number of iterations without really improving the solution. Often, ϵ in the range of 10^{-3} to 10^{-6} works well.

In many large empirical studies, this algorithm was found to be very efficient. Since implementation of the algorithm is very simple, SMO is one of the popular algorithms for learning SVMs.

3.1.1 Examples

In this subsection we illustrate the SVM method through two examples.

Example 1: In this example, the feature vectors are in $\Re^2$ and they are drawn from a uniform distribution on $[0, 4] \times [0, 4]$. Fig. 8 shows the training sample of 2000 examples. All circles are of one class and stars are in the other class. As is easy to see, the underlying class structure is a 4×4 checker

board pattern. That is, the feature space consists of 16 squares on a 4×4 grid with adjacent squares being regions of opposite class. We have used the SMO algorithm with Gaussian kernel function. The parameters used for simulation are: σ=2.0 in the kernel function, $\epsilon = 10^{-3}$ in the SMO algorithm, and $C = 5$ in the objective function. The final SVM obtained gives 99.9% accuracy on the training set and 99% accuracy on a separately generated test set.

This example is meant only for illustration and hence the classification accuracies *per se* are not important. However, the class boundary to be learnt is highly nonlinear and hence the performance is impressive. What is more interesting, however, is to look at the those training samples that turn out to be support vectors. These are shown in Fig. 9. It is easy to see that only the patterns that are closest to the separating surface between the classes turn out to be the support vectors. As remarked earlier, the support vectors are the most critical patterns for the classification task at hand. Identification of such critical patterns is, so to say, a bonus from the SVM method.

Example 2: As our second example, we discuss a realistic PR application, namely, fingerprint classification. The results presented here are from [19]. Finger prints are the characteristic structure of flow lines of ridges and furrows that are present on the skin on one's finger. It is well known that finger prints are unique and are well suited for person identification. When finger print image database is large (*e.g.*, in application relating to law enforcement), it is desirable to have a systematic partitioning to reduce time and complexity involved in finger print matching. Fingerprint classification is one way of establishing such a partitioning. (The idea is that if we know the class of the test fingerprint then it needs to be matched with only those fingerprints in the database which are of the same class). As established in literature, fingerprints are classified in six classes: Right loop (R), Left loop (L), Arch (A), Tented arch (T), Scar (S), and Whorl (W). The categorization of finger prints into these classes is based on the global shape of the orientation field of ridge lines in the finger print image. In the example here we use a 96-dimensional feature vector.[7]

[7]We first detect all the ridge lines along with orientations and then extract features which essentially characterize the distribution of orientations of ridge lines around the fingerprint image. For our purposes of illustrating the SVM method, the details of feature extraction are not relevant.

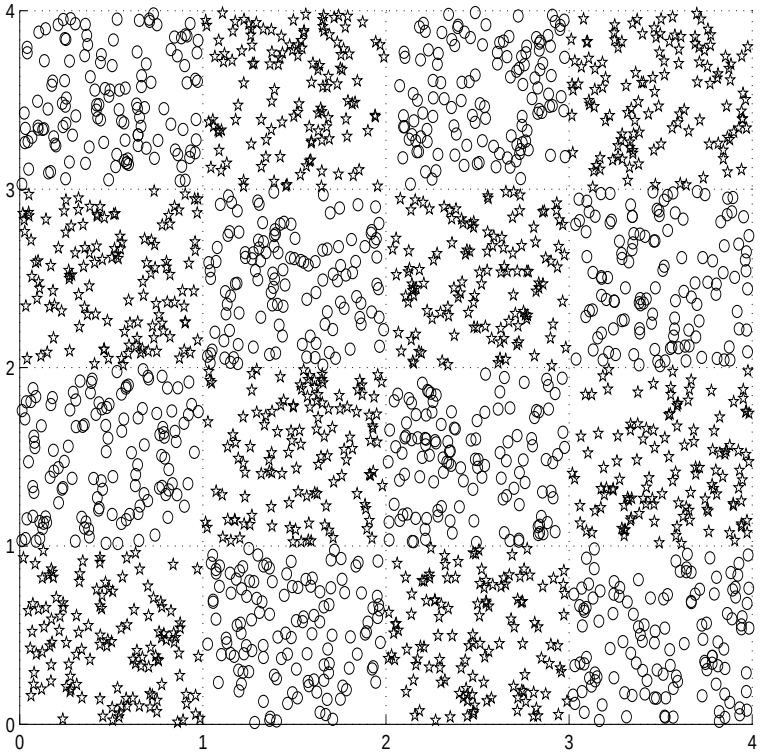


Figure 8: Training patterns in Example 1.

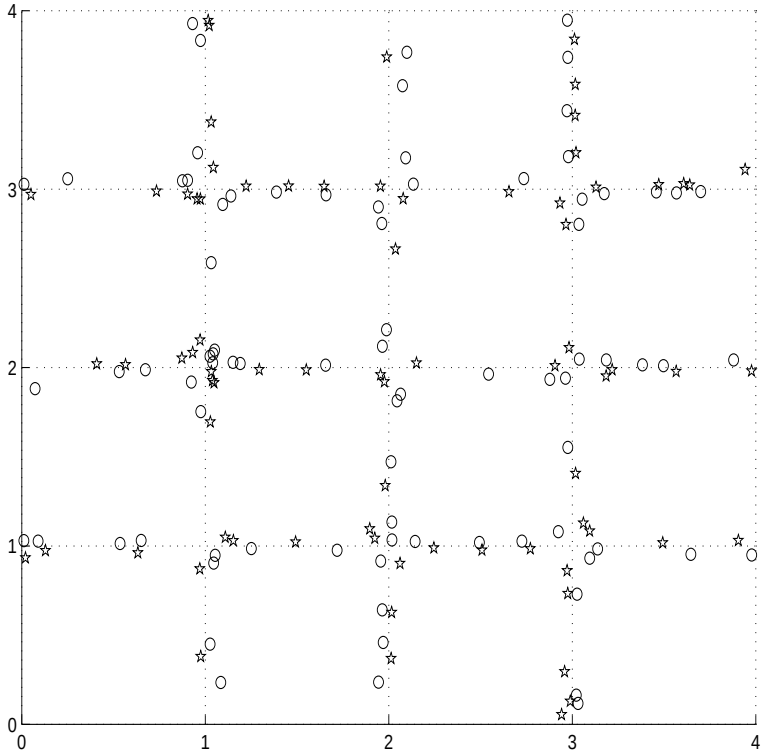


Figure 9: Support vectors learnt in Example 1.

We present results obtained on a fingerprint database from National Institute of Standards and Technology, USA (NIST). The NIST database has about 2700 finger print images of which only 5 are of class Scar. Also the classes A and T together have only 133 examples. So, for our illustration here we consider classifying a fingerprint into one of four classes: L, R, W, or AT. (Here class AT stands for the combined set of Arch or Tented Arch). We used four SVMs. Our first SVM is for separating class AT from L, R and W. The remaining three SVMs are for distinguishing each of L, R, and W from the other two. We used 1500 images from the database as training samples. The specific images that constitute the training set are randomly selected. It is observed that the classification accuracy is about the same with different random choice of training sets.

In this example also we have used the SMO algorithm with a Gaussian kernel function. As remarked earlier, learning an SVM involves solving a quadratic programming problem, and hence the method is very efficient. With 1500 training patterns, learning of each of the SVMs took about 2.2 seconds on a Pentium 300 MHz PC.

The accuracy obtained on the full NIST database using our classifier consisting of the four SVMs is presented in Table 1 in the form of a confusion matrix. Each row of the table shows the number of patterns of that class from the database classified into different classes by our classifier. The last column of the table gives the total number of patterns of that class in the database. As can be seen, in some rows the sum of the first four columns is not equal to the last column. This is because, as explained in Section 2.1, in a multiclass classifier using SVMs, we reject a pattern if none of the SVMs give positive output. However, for calculating the accuracy shown in the table, we counted the rejected patterns also as errors. From Table 1, it is seen that the final classification accuracy obtained is very good. (The performance is also seen to be as good as or better than any other method on this database [19]).

A simple classifier that is often used in applications is the so called nearest neighbour classifier (NNC). In an NNC, we store some (or all) of the training patterns as *prototypes*. Whenever a new pattern is to be classified we find the prototype pattern (feature vector) that is closest to this pattern (feature vector) and classify the new pattern into the same class as this nearest prototype. We can, for example, use the Euclidean distance for deciding on the

Class	L	R	W	AT	%Accuracy	Total patterns
L	801	0	0	3	99.6	804
R	0	688	0	16	93.6	735
W	0	2	998	0	97.6	1023
AT	2	0	0	129	97.0	133

Table 1: Confusion matrix with SVMs

Class	L	R	W	AT	%Accuracy
L	804	0	0	0	100
R	0	735	0	0	100
W	6	7	1009	1	98.63
AT	3	7	0	123	92.48

Table 2: Confusion matrix for Nearest Neighbour classifier – all training samples as prototypes.

nearest prototype.

To illustrate the role played by support vectors here, we show in Table 2 the results obtained with an NNC classifier where we used all the 1500 training patterns as prototypes. As can be seen, the accuracy of the SVM classifier is as good as that with the nearest neighbour classifier. In this example, for all the four SVMs together we had about 90 distinct patterns as support vectors. Thus, the SVM classifier needs to store only about 90 patterns rather than the 1500 needed for the NNC.

There is another way of appreciating the fact that support vectors are the most important patterns in the training set. Table 3 shows the accuracy obtained with a NNC classifier if we had used the support vectors as the prototypes. As can be seen, this NNC performs as well as the NNC that uses all the 1500 samples as prototypes. For comparison, Table 4 shows the accuracy obtained when we use 100 randomly selected training patterns as prototypes. As is easily seen, the performance is quite poor. As illustrated here, in many applications, the support vectors themselves constitute a very useful output from the SVM method.

Class	L	R	W	AT	%Accuracy
L	804	0	0	0	100
R	0	745	0	0	100
W	3	8	1011	1	98.83
AT	3	4	0	126	94.7

Table 3: Confusion matrix for Nearest Neighbour classifier – support vectors as prototypes.

Class	L	R	W	AT	%Accuracy
L	732	0	0	72	91.04
R	0	678	0	57	92.2
W	38	93	892	0	87.2
AT	14	33	17	69	51.9

Table 4: Confusion matrix for Nearest Neighbour classifier – 100 random examples as prototypes.

3.2 Other Methods for solving the Optimization Problem

There are many other specialized methods for solving the optimization problem given by (28). We briefly mention two such ideas below.

Let S_1 denote the convex hull of all training patterns belonging to class $+1$ and let S_2 denote the convex hull of training patterns belonging to class -1. Let $x \in S_1$ and $x' \in S_2$ be such that $d(x, x') \leq d(z, z') \; \forall \; z \in S_1, \; z' \in S_2$ where $d(x, x')$ is the distance between the two vectors. It is easy to show that the optimal separating hyperplane is the perpendicular bisector of the line joining x and x'. (We assume here that S_1 and S_2 are disjoint so that a separating hyperplane exists). There are efficient techniques for finding a point from a convex set that is closest to another convex set. These can be adopted to find the SVM and this method is seen to be more efficient than SMO [11].

Another interesting idea is the following. Suppose we change the primal objective function from $0.5w^Tw + C \sum \; \xi_i$ to $0.5w^Tw + C \sum \; \xi_i + b^2$.

That is, in addition to the margin we are including the bias or location of the hyperplane also. The main advantage of this change is that the new dual will have only bound constraints on the variables (i.e., $0 \leq \mu_i \leq C$) and it will not have any equality constraints. Then the dual can be efficiently solved using a gradient projection type algorithm which essentially involves iterative solution of a set of linear equations with bounds on the variables. Successive Overrelaxation is an efficient technique for solving large systems of linear equations and it can be adapted to solve this new optimization problem. It can be shown that the solution of the new optimization problem would be close to the required optimal hyperplane. This algorithm is also found to be more efficient than SMO when the size of the training set is very large [10].

4 Connections with Statistical Learning Theory

In this section we briefly discuss the SVM approach from the point of view of statistical learning theory. Unlike the rest of this chapter, here we assume that the reader has good background in probability theory.

Consider a model of learning from examples as follows. We are given a sample of l observations (or examples) $\{(x_i, y_i),\ i = 1, \ldots l\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. $\mathcal{X}$ is called the instance space and $\mathcal{Y}$ is called the outcome space. The examples are drawn in *independent and identically distributed* (*iid*) manner according to a probability distribution $P(x, y)$ on $\mathcal{X} \times \mathcal{Y}$. However, we do not know this probability distribution. We have a learning machine that is capable of implementing a set of parameterized functions, from $\mathcal{X}$ to $\mathcal{Y}$, denoted by $\mathcal{F} = \{f(x, \alpha),\ \ \alpha \in \Lambda\}$. The learning problem is to find the best approximator from $\mathcal{F}$ for the relationship between x_i and y_i using the given sample of *iid* examples. We are given a loss function $L(y, f(x, \alpha))$ which is a measure of discrepancy between the output of the approximator with parameter α and the true or desired output y for an instance x. Define a risk functional on the parameter set Λ which is the expectation of loss as

$$R(\alpha) = \int L(y, f(x, \alpha))\ dP(x, y) \tag{46}$$

We formally define the goal of learning as finding the function $f(x, \alpha_o)$ where α_o minimizes the risk, $R(\alpha)$ over all $\alpha \in \Lambda$. This general framework en-

compasses problems such as Pattern Recognition, regression estimation, and density estimation.

Consider a 2-class PR problem. We take the instance space to be the space in which the feature vectors take values. Thus for us, $\mathcal{X} = \Re^n$. We can take $\mathcal{Y} = \{+1, -1\}$. We take $\mathcal{F}$ to be a set of indicator functions. For example, we can have $f(x, \alpha) = \Theta(w^T x + b)$ with $\alpha = (w, b) \in \Re^{n+1}$ and Θ is defined by: $\Theta(v) = -1$ if $v < 0$ and $\Theta(v) = 1$ otherwise. Then the set $\mathcal{F}$ will be the set of hyperplane classifiers that we have considered earlier. If we choose the so called 0–1 loss function given by

$$\begin{aligned} L(a, b) &= 0 \text{ if } a = b \\ &= 1 \text{ if } a \neq b \end{aligned} \tag{47}$$

then the risk functional is the probability of misclassification. Hence minimizing $R(\alpha)$ amounts to finding a discriminant function that minimizes probability of misclassification.

The problem of regression estimation is also a special case of our general framework. Suppose $\mathcal{X} = \Re^n$ and $\mathcal{Y} = \Re$. Choose the so called quadratic loss function which is given by $L(a, b) = (a - b)^2$. The regression estimate of y as a function of x is given by

$$g(x) = \int y \, dP(y|x)$$

where $P(y|x)$ is the conditional distribution of y given x. If the regression function is in the set of functions $\mathcal{F}$, then it is easy to see that the minimizer of the risk functional would be the regression function.

In the rest of this section, we use the notation $z = (x, y)$ (and similarly $z_i = (x_i, y_i)$) and $Q(z, \alpha) = L(y, f(x, \alpha))$. Now, the goal is to minimize the risk functional defined by

$$R(\alpha) = \int Q(z, \alpha) \, dP(z), \quad \alpha \in \Lambda. \tag{48}$$

The special characteristic of learning problems is that the distribution $P(z)$ is unknown. Thus given any α we cannot calculate $R(\alpha)$. We need to find the minimizer of risk using only the *iid* samples drawn according to $P(z)$. Almost all learning algorithms are based on the so called Empirical Risk Minimization (ERM) principle which is explained below.

Define another functional called empirical risk (based on l samples) by

$$\hat{R}_l(\alpha) = \frac{1}{l} \sum_{i=1}^{l} Q(z_i, \alpha). \tag{49}$$

The RHS of (49) is the expectation of loss with respect to the empirical distribution determined by the l *iid* samples. Since $\hat{R}_l(\alpha)$ can be calculated for any α from the given sample, we can try and minimize the empirical risk using some optimization technique. Under the ERM principle we approximate the minimizer of $R(\alpha)$ by the minimizer of $\hat{R}_l(\alpha)$.

Let α_l denote the minimizer of $\hat{R}_l$ (defined by (49)) and let α_o denote the minimizer of R (defined by (48). For the ERM principle to be effective we want $R(\alpha_l) \to R(\alpha_o)$ and $\hat{R}_l(\alpha_l) \to R(\alpha_o)$ in probability as $l \to \infty$. (Note that $\hat{R}_l$ and α_l are random because they depend on the *iid* samples given). Since we do not know the probability distribution $P(z)$, we want the above convergence to hold no matter what this probability distribution is.

Suppose our loss function is bounded in the sense that for some constants A, B,

$$A \leq \int Q(z, \alpha)\, dP(z) \leq B, \quad \forall \alpha \in \Lambda$$

for all probability distributions $P(z)$. Then, to ensure convergence of $\hat{R}_l(\alpha_l)$ and $R(\alpha_l)$ to $R(\alpha_o)$ as specified earlier, it is necessary and sufficient that $\hat{R}_l(\alpha) \to R(\alpha)$ as $l \to \infty$ uniformly over $\alpha \in \Lambda$. It may be noted here that whether or not this uniform convergence holds is dependent on the chosen set of functions $\mathcal{F}$, which determines the capability or capacity of our learning machine. A necessary and sufficient condition for this uniform convergence to hold is that a 'size' parameter that we can associate with the set of functions $\mathcal{F}$, to be called *VC dimension*[8] of $\mathcal{F}$, is finite.

Let $\mathcal{F} = \{f(x, \alpha),\ \alpha \in \Lambda\}$ be a set of indicator functions. That is, each of the functions takes only two values, say, $+1$ and -1. We define the VC dimension of $\mathcal{F}$ as follows. A finite set $B = \{x_1, \ldots x_m\} \subset \mathcal{X}$ is said to be *shattered* by $\mathcal{F}$ if given any $C \subset B$, we can find a $\alpha \in \Lambda$ such that $f(x, \alpha) = +1$ if $x \in C$ and $f(x, \alpha) = -1$ if $x \in B - C$. What this means is the following. B is shattered if for every arbitrary assignment of class labels to elements of B, there is a function in $\mathcal{F}$ which realizes this classification of elements of B. The VC Dimension of $\mathcal{F}$ is defined to be the cardinality

[8]This is named after Vapnik and Chervonenkis, who derived this result.

of the largest shattered subset of $\mathcal{X}$. If arbitrarily large finite subsets can be shattered then the VC dimension is infinite. VC dimension of $\mathcal{F}$ is equal to h implies: there is *at least* one subset of $\mathcal{X}$ with h elements that is shattered; and *no* subset of $\mathcal{X}$ with $h+1$ elements is shattered. The VC dimension of hyperplane classifiers in $\Re^n$ (which is the example of $\mathcal{F}$ we considered earlier in this section) is $n+1$.

VC dimension of a set of functions (or of a learning machine capable of implementing those functions) is a measure of the capacity of the learning machine. In our framework, a learning machine finds one of the functions from the set of functions, $\mathcal{F}$, that best accounts for the data of training samples given. That is, the learning machine finds an α which minimizes the empirical risk on the given sample. The objective of learning is to find a function that does well (on the average) on all instances, not just the ones in the training samples. That is, we want the minimizer of true risk. Hence a natural question is how can we be confident that good performance (by the classifier) on the training data translates into good performance on all the other unseen patterns. This is the issue of whether the learning machine is able to achieve proper generalization based on the given data. Often, in practice, we keep a separate sample of patterns, called test set, and assess the performance of the classifier (learnt using the training set) on the test set. The framework described above gives us some theoretical guidelines on assessing the generalization capabilities of classifiers. Higher VC dimension would mean that, unless we have a correspondingly larger number of examples, we cannot hope for the learning machine to be able to do proper generalization from the given data. This informal understanding can be stated rigorously in the form of a bound on the risk functional as in the following theorem.

Theorem 4.1: With probability at least $1-\eta$, the inequality

$$R(\alpha) \leq \hat{R}_l(\alpha) + \frac{a}{2}\left(1 + \sqrt{1 + \frac{4\hat{R}_l(\alpha)}{a}}\right) \tag{50}$$

holds simultaneously for all α (that is, for all functions in $\mathcal{F}$) with

$$a = 4\frac{h(\ln(2l/h)+1) - \ln\eta}{l} \tag{51}$$

if $\mathcal{F}$ is a set of indicator functions with VC dimension $h < \infty$ and we are using a positive bounded loss function.

The remarkable thing about the bound given by (50) is that for every α, the true risk (which depends on the unknown probability distribution $P(z)$) is bounded above by the empirical risk on a sample of l *iid* examples plus a term that depends only on the VC dimension and the number of examples. Since this holds for any sample of examples with any functional relationship between x and y, it is a worst case bound. There are also other similar bounds that can be derived. Since it is a worst case bound it may not be useful for predicting the true risk based on the empirical risk. However, the form of this bound is very interesting from the viewpoint judging the ability of a learning machine in making valid generalizations from the data.

The first term on the RHS of (50) is empirical risk. So, for the function learnt by the machine this tells how well the best function from $\mathcal{F}$ approximates the data. We can call this the data error. The second term, to be called the generalization error, increases with increasing values of the ratio (h/l). For a given number of examples, with higher h we get higher generalization error. With a set of functions $\mathcal{F}$ of higher VC dimension we can more closely approximate many different data sets and hence lower data error does not necessarily mean lower value of true risk. (A simple but extreme example of this is: with a data of a hundred pairs of real numbers we can get zero data error by fitting a hundredth degree polynomial but it is very unlikely that this polynomial correctly captures the underlying functional relationship). Thus, to obtain a low value of true risk we should strive to find a set of functions with 'low enough' VC dimension using which we can get 'low enough' data error. The bound given by (50) gives us some theoretical guidelines for effecting a tradeoff between the quality of approximation (in terms of error on the specific data set) and the complexity of the chosen class of functions (in terms of VC dimension of the set $\mathcal{F}$). These ideas gave rise to the so called structural risk minimization (SRM) principle under which one looks for methods which, along with minimizing the value of empirical risk, control the VC dimension of the learning machine.

Now, we discuss the relevance of these ideas to Support Vector machines. The support vector machines are found to be very effective in learning good classifiers in many practical applications. At first sight this is very surprising. As mentioned earlier, the VC dimension of hyperplane classifiers in

n-dimensional space is $n+1$. The nonlinear SVMs try to find a separating hyperplane in a very high dimensional space. With patterns in $\Re^n$ if we use a polynomial kernel with degree p, the feature space will have dimension $\frac{(n+p-1)!}{p!\,(n-1)!}$ which can be very large even with modest p. Since the VC dimension is thus very high, we do not expect to learn (in this space) a hyperplane classifier with good generalization abilities unless we can have correspondingly larger number of training points. Hence the question is why are SVMs found to be very effective. A good partial answer to this question is provided by theorem 4.2 below.

Definition: A hyperplane in $\Re^m$ given by $w^T x + b = 0$ with $||w|| = 1$, is called a Δ-margin separating hyperplane if it classifies vectors x as follows

$$\begin{aligned} y &= 1 \text{ if } w^T x + b \geq \Delta \\ &= -1 \text{ if } w^T x + b \leq \Delta \end{aligned} \tag{52}$$

The classification of vectors x for which $w^T x + b$ falls in $(-\Delta, \Delta)$ is undefined (and such patterns are not counted while evaluating the errors made by any classifier in this class).

Theorem 4.2: Let all vectors x belong to a sphere of radius ρ. Then the set of Δ-margin separating hyperplanes has VC dimension bounded by

$$h \leq \min \left(\left[\frac{\rho^2}{\Delta^2} \right], n \right) + 1, \tag{53}$$

where $[y]$ denotes the smallest integer greater than y.

The class of classifiers learnt by SVMs are essentially Δ-margin separating hyperplanes. (It is easy to see that if we divide the inequalities in (3) by $||w||$, then we get the classifier in (52). Thus, the Δ in (52) is essentially same as what we called the margin of the separating hyperplane). Suppose that, through the use of kernel function we mapped the pattern vectors into a high dimensional feature space where we were able to find a separating hyperplane with good margin. Then, though the feature space may be very high dimensional, the VC dimension of the effective class of functions in which we are searching could be much smaller. We can think of the SVM method as

follows. Suppose we arrange $\mathcal{F}$, the set of all hyperplane classifiers into a nested sequence of subsets: $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots \mathcal{F}$ where the i^{th} subset contains all hyperplanes with $||w|| \leq c_i$ for some increasing sequence of real numbers c_i. If we think of searching for best Δ-margin separating hyperplanes in each of these subsets then this sequence of subsets will have decreasing VC dimension. The SVM method, in finding the highest margin separating hyperplane, is essentially searching for the subset of $\mathcal{F}$ with least VC dimension and in this subset it is finding a function with zero data error. Thus, the SVM method is using the bound given by (50) in a very novel manner. Operating under the constraint of keeping the data error zero, the method is minimizing the generalization error by searching for a subclass of functions with least VC dimension. This is the idea of Structural Risk Minimization. All these comments apply only to the case where we can find an optimal *separating* hyperplane. However, in a general SVM method we use a penalty term in the objective function. In this general case, based on the parameter C, the method finds a good tradeoff between data error and generalization error. This seems to be the reason why SVM classifiers are found to be very effective in applications.

5 Discussion

In this chapter, we described the Support Vector Machine (SVM) method as an efficient algorithm for learning nonlinear discriminant functions in PR problems. The basic idea is as follows. When the patterns are linearly separable, the optimal hyperplane (i.e., the separating hyperplane with maximum margin) has very good generalization capabilities. When patterns are not linearly separable (which is the case most often), we can map the patterns into a very high dimensional space $\mathcal{H}$ through a suitable nonlinear transform ϕ and find the optimal hyperplane in $\mathcal{H}$. By the use of kernel functions, we do not need to ever explicitly construct ϕ or work in $\mathcal{H}$. (As a matter of fact, $\mathcal{H}$ can be infinite-dimensional). Also, by using a penalizing term in the objective function, we can take care of the case where the transformed patterns in $\mathcal{H}$ also may not be linearly separable.

This basic idea can be used in many other learning problems as well. In this section we briefly outline this for the case of nonlinear regression. Then we comment on how this idea can be exploited in other problems also.

Regression problem is closely related to the PR problem. Here we are given examples $\{(x_i, y_i), i = 1, \ldots, l\}$ with $x_i \in \Re^n$ and $y \in \Re$. We want to find the 'best' functional relationship between x and y. We search in some parameterized class of functions $g(x, W)$, where W is the parameter vector, for the best function. In case of linear regression, we take $g(x, w, b) = w^T x + b$ and the parameter vector, W, here contains $w \in \Re^n$ and $b \in \Re$. If we need to use a nonlinear approximator, then, using the SVM idea, we can choose the structure of our approximator as $g(x, w, b) = w^T \phi(x) + b$ where $w \in \Re^m$, $b \in \Re$ and $\phi : \Re^n \to \Re^m$ is a suitable nonlinear transformation. This is the structure we consider here. We find the best parameters w, b by minimizing $EL(y, g(x, w, b))$ where L is a suitable loss function. One loss function that is interesting from the point of view of SVM method is the so called ϵ-insensitive loss function defined by

$$\begin{aligned} L(y, g(x, w, b)) &= 0 \qquad\qquad \text{if } |y - g(x, w, b)| < \epsilon \\ &= |y - g(x, w, b)| - \epsilon \quad \text{otherwise.} \end{aligned} \tag{54}$$

Now (in analogy with SVM method for PR) we can minimize the expected loss as follows: Find w, b, ξ_i, ξ_i' to solve the following optimization problem.

$$\begin{aligned} \min \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} (\xi_i + \xi_i') \\ \text{subject to} \quad & y_i - w^T \phi(x_i) - b \leq \epsilon + \xi_i, \ i = 1, \ldots, l \\ & w^T \phi(x_i) + b - y_i \leq \epsilon + \xi_i', \ i = 1, \ldots, l \\ & \xi_i \geq 0, \ i = 1, \ldots, l \\ & \xi_i' \geq 0, \ i = 1, \ldots, l. \end{aligned} \tag{55}$$

The dual of this problem is

$$\begin{aligned} \max \quad & \sum_{i=1}^{l} y_i(\mu_i - \mu_i') - \epsilon \sum_{i=1}^{l} (\mu_i + \mu_i') - \frac{1}{2} \sum_{i,j=1}^{l} (\mu_i - \mu_i')(\mu_j - \mu_j') \phi(x_i)^T \phi(x_j) \\ \text{subject to} \quad & 0 \leq \mu_i, \mu_i' \leq C, \ \forall i \\ & \sum_{i=1}^{l} (\mu_i - \mu_i') = 0 \end{aligned} \tag{56}$$

Once again the dual is a QP problem with one equality constraint and bound constraints on variables, and the training samples appear only as inner prod-

ucts. We can replace $\phi(x_i)^T\phi(x_j)$ by $K(x_i, x_j)$ where K is a suitable kernel function. By solving (56) we obtain the optimal Lagrange multipliers, $\mu_i^*, \mu_i'^*,\ i = 1, \ldots, l$. Then the solution of (55), w^*, is given by

$$w^* = \sum_{i=1}^{l} (\mu_i^* - \mu_i'^*)\phi(x_i).$$

Here, the support vectors are those x_i for which we have $\mu_i^* = \mu_i'^*$. Thus our optimal approximation for y as a function of x is given by

$$g(x, w^*, b^*) = \sum_{i=1}^{l} (\mu_i^* - \mu_i'^*)K(x_i, x) + b^*.$$

(As in the case of PR problem, we can obtain b^* also in terms of the optimal Lagrange multipliers and the kernel function). Thus, by using kernel functions, we can efficiently solve the nonlinear regression problem without ever explicitly constructing ϕ or working in the high dimensional range space of ϕ. As in the case of PR problems, it may be noted that if we have used Gaussian kernel, then, this method can efficiently learn an approximating function in the form of a RBF network.

We can sum up the SVM methodology as follows. We have data vectors, $x_i,\ i = 1, \ldots, l$, with $x_i \in \Re^n$. We can map the data into a high dimensional space $\mathcal{H}$ using a suitable nonlinear transformation ϕ so that now the data is $(\phi(x_i), y_i)$. (Here, $y_i \in \{-1, +1\}$ for PR problems, $y_i \in \Re$ for regression and so on). In the high dimensional space we can hope to find a simple (meaning linear) classifier or regression function. Statistical learning theory tells us that, normally, if we want to learn a high dimensional parameter vector we need correspondingly larger number of examples to be able to learn properly. However, structural risk minimization framework gives us a handle to learn a classifier or regressor with small enough generalization error. Further, if the resulting optimization problem has the right structure (that is, if the data vectors appear only as inner products), then by clever use of kernel functions we can completely avoid working in the space $\mathcal{H}$. We can do all our computations efficiently in $\Re^n$ though we are effectively finding a linear classifier in the high dimensional space $\mathcal{H}$.

The utility of kernel functions is that we can implicitly compute inner products (in the space $\mathcal{H}$) without even knowing the transformation ϕ and thus never working in the space $\mathcal{H}$. This would imply that any algorithm that

uses only inner products can be recast so that we are actually transforming the data into a very high dimensional space and using the same method there. Thus kernel functions allow us to elegantly construct nonlinear versions of many linear algorithms [17]. For example, we are able to learn a nonlinear discriminant function using an algorithm which is computationally as efficient as finding a linear discriminant function.

This idea of kernel functions can be used in many other learning problems also. Consider a nearest neighbour classifier where one needs to find distance between the test pattern and the stored prototypes. Since distances can be computed as inner products, we can effectively find distance in a transformed space using kernel functions. Such a thing is often done in a nearest neighbour classifier. For example, using a Gaussian kernel would amount to using Mahalonobis distance which is a popular choice in nearest neighbour classifiers. Another example which is more interesting is the so called Fisher linear discriminant in pattern recognition. Here one is interested in finding a linear projection of the feature vector so that the separation between the classes is maximized. Using kernel functions one can find a good nonlinear projection without incurring any significant additional computational overhead. In a similar way one can use kernel functions to reformulate the principal component analysis problem to obtain a good nonlinear transformation for dimensionality reduction [17].

As said in the introduction, the SVM method grew out of the research in statistical learning theory, mainly from the work of Vapnik and his associates, starting from late sixties. The SVM algorithm was first proposed in [3] and was later generalized in [4]. The SVM algorithm resulted in the best performing classifier for the US Zip code data (see the description in [5]) and generated a lot of interest in the method. The tutorial by Burgess[6] provides a very good introduction to SVMs. It also contains a number of references to applications. The book by Cherkaasky and Muller [7] is a good source for related material. The QP problem that needs to be solved to obtain the SVM classifier, has also been a subject of much interest. One of the first algorithms that exploited the chunking idea is the so called SVM-light algorithm due to Joachim [8]. The code for this algorithm is freely downloadable from the net. The SMO algorithm is due to Platt [9] and it is faster than SVM-light on most problems, often by a factor of 4. The other two algorithms briefly mentioned in section 3 are from [11] and [10]. Algorithms such as SVM-light

and SMO rely on decomposing a large quadratic programming problem into a series of low dimensional problems. Theoretical analysis of such methods can be found in [12, 13]. We have presented the SVM method as a 2-class pattern classification technique and have briefly indicated how we can use a number of SVMs for solving a multiclass problems. A good discussion of the available design choices for multiclass problems can be found in [14, 15]. A good tutorial on support vector regression is downloadable from www.kernel-machines.org/papers/tr-30-1998.ps.gz. The book by Vapnik[5] provides an excellent overview of statistical learning theory. More rigorous accounts of the theory and its relevance to different learning techniques is available in [20, 21]. The recent paper by Vapnik [22] provides a good summary of relevant results. The theorems quoted in Section 4 are all taken from this paper. The basic result regarding uniform convergence of empirical expectations using the idea of what is now called the VC dimension, is proved in [23]. The VC theory has given rise to much interest in studying complexity of learning algorithms and the so called Probably Approximately Correct (PAC) learning model (see, e.g., [24, 25]). Some discussion of the SVM method in the context of related ideas from statistics can be found in [26] and [27]. A good discussion on kernel-based learning algorithms is in [17, 18, 16]. The paper [17] also gives references to a large number of recent applications. Some of the applications of SVM in PR problems can be found in [28]-[31]. A good source for information on various applications of SVM method is the web site: www.clopinet.com/isabelle/Projects/SVM/applist.html. Another good source for information various issues on kernel based learning algorithms is the web site www.kernel-machines.org. This site contains many tutorials that can be downloaded and it also provides some sets of data for benchmarking learning algorithms. A very good list of all publications related to this area is found in www.kernel-machines.org/publications.html.

Appendix

In this appendix we state some results from optimization theory which are used in developing the SVM algorithm. This material follows[32].

Consider the following constrained optimization problem.

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & a_j^T x \leq b_j, \quad j = 1, \ldots, r, \end{aligned} \tag{57}$$

where $f : \Re^n \to \Re$ is a convex function and $a_j \in \Re^n$, $b_j \in \Re$, $\forall j$. Any point $x \in \Re^n$ that satisfies the constraints, $a_j^T x \leq b_j, \quad j = 1, \ldots, r$, is said to be *feasible* for (57). An $x^* \in \Re^n$ that is feasible and satisfies $f(x^*) \leq f(x)$, $\forall x \in \Re^n$ such that x is feasible, is called a *constrained global minimum* of f or a *global solution* (or simply a *solution*) of the optimization problem given by (57). We define the so called *Lagrangian* corresponding to (57) as

$$L(x, \mu) = f(x) + \sum_{j=1}^{r} \mu_j (a_j^T x - b_j) \tag{58}$$

where $\mu = [\mu_1 \ldots \mu_r]^T \in \Re^r$ and μ_j are called the *Lagrange multipliers.*

> *Theorem 1*: Let f be convex and continuously differentiable. Then x^* is a global solution of (57) if and only if x^* is feasible for (57) and there exist $\mu_j^*, \ j = 1, \ldots, r$, such that
>
> 1. $\nabla_x L(x^*, \mu^*) = 0$
> 2. $\mu_i^* \geq 0$
> 3. $\mu_j^* (a_j^T x^* - b_j) = 0, \ \forall j$

where $\nabla_x L$ is the gradient of L with respect to the vector x. The conditions in the above theorem are called the Kuhn Tucker conditions for the problem (57).

Define a function, $q : \Re^r \to [-\infty, \infty)$, called the *dual function* for (57), by

$$q(\mu) = \inf_x L(x, \mu) \tag{59}$$

where L is the Lagrangian. ($q(\mu)$ may take value $-\infty$ if the infimum is not attained).

Now the dual (or the dual optimization problem) for (57) is defined to be the optimization problem

$$\begin{aligned} \max \quad & q(\mu) \\ \text{subject to} \quad & \mu_j \geq 0,\ j = 1, \ldots, r. \end{aligned} \tag{60}$$

The optimization problem given by (57) is called the *primal problem.* The following theorem establishes the relationship between the solutions of the primal and dual problems.

Theorem 2:

a. If the primal problem has optimal solution then so does the dual and the corresponding optimal values are equal.

b. In order for x^* to be optimal primal solution and μ^* to be optimal dual solution, it is necessary and sufficient that x^* is feasible for the primal problem, μ^* is feasible for the dual problem and

$$f(x^*) = L(x^*, \mu^*) = \min_x L(x, \mu^*). \tag{61}$$

Since (61) demands $f(x^*) = L(x^*, \mu^*)$, and since x^* is feasible for (57), from (58), we must have

$$\mu_j^*(a_j^T x^* - b_j) = 0, \quad j = 1, \ldots, r$$

which is called complementary slackness condition. Also, we must have x^* as the unconstrained minimum of $L(x, \mu^*)$ which means x^* is a solution of $\nabla_x L(x, \mu) = 0$ at $\mu = \mu^*$. These are the conditions we have used in deriving the SVM algorithm in section 2.

References

[1] S. Geman and E. Bienenstock, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[2] R. O. Duda and P. E. Hart, *Pattern Classification and scene analysis*. New York: Wiley, 1973.

[3] B. Boser, I.Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifier," in *Proc. of Fifth Annual workshop on computational learning theory,Pittsburgh*, pp. 144–152, ACM, 1992.

[4] C. Corties and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[5] V. N. Vapnik, *The nature of statistical learning theory*. Springer Verlag, NY, 1995.

[6] C. J. C. Burges, "A tutorial on support vector machine for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 1–47, 1998.

[7] V. Cherkassky and F. Mulier, *Learning from Data: Concepts, Theory and Methods*. Wiley, New York, 1998.

[8] T. Joachim, "Making large scale support vector machine learning practical," in *Advance in Kernal Methods: Support Vector Learning* (B. Scholkopf, C. J. C. Burges, and A. J. Smola, eds.), pp. 169–184, MIT Press, Cambridge,MA, 1999.
The source code for this algorithm can be obtained from http://svmlight.joachims.org.

[9] J. Platt, "Sequential minimal optimisation: A fast algorithm for training support vector machine," in *Advance in Kernal Methods: Support Vector Learning* (B. Scholkopf, C. J. C. Burges, and A. J. Smola, eds.), pp. 185–208, MIT Press, Cambridge,MA, 1999. available at: http://www.research.microsoft.com/users/jplatt/smo-book.pdf.

[10] O. L. Mangasarian and D. R. Musicant, "Successive overrelaxation for support vector machine," *IEEE Transactions on Neural Networks*, pp. 1032–1037, Sept. 1999.

[11] S. S. Keerthi, S. K. Shevade, C. Bhattacharya and K. R. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," *IEEE Transactions on Neural Networks*, pp. 124–136, Jan. 2000.

[12] C. J. Lin, "On the convergence of the decomposition method for support vector machines," *IEEE Transactions on Neural Networks*, Vol. 12, pp. 1288–1298, Nov. 2001.

[13] C. J. Lin, "A formal analysis of stopping criteria of decomposition methods for support vector machines," *IEEE Transactions on Neural Networks*, Vol. 13, pp. 1045–1052, Sept. 2002.

[14] C. W. Hsu and C. J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, Vol. 13, pp. 415–425, Mar. 2002.

[15] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, Vol. 5, pp. 101–141, Jan., 2004.

[16] M. G. Genton, "Classes of Kernels for Machine Learning: A Statistics Perspective," *Journal of Machine Learning Research*, Vol. 2, pp. 299–312, Dec., 2001.

[17] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, pp. 181–201, Mar. 2001.

[18] A. Ruiz and P. E. Lopez-de-Teruel, "Nonliner kernel-based statistical pattern analysis,", *IEEE Transactions on Neural Networks*, Vol. 12, pp. 16–32, Jan. 2001.

[19] S. Shah and P. S. Sastry, "Finger print classification using a feedback based line detector," *IEEE Transactions on Systems Man and Cybernetics Part B*, Vol. 34, pp. 85–94, Feb. 2004.

[20] M. Vidyasagar, *A theory of learning and generalisation.* Springer Verlag, NY, 1997.

[21] V. N. Vapnik, *Statistical learning theory.* Wiley, NY, 1998.

[22] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, Vol. 10, pp. 988–999, Sept. 1999.

[23] V. N. Vapnik and A. J. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory Probab. Apl.*, vol. 16, pp. 264–280, 1971.

[24] B. K. Natarajan, *Machine Learning: A Theoretical Approach*. Morgan Kaufmann, San Mateo, CA, 1991.

[25] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *J. ACM*, vol. 36, pp. 929–965, 1989.

[26] D. Mattera, F. Palmeri, and S. Haykin, "Simple and robust method for support vector machine expansions," *IEEE Transactions on Neural Networks*, pp. 1038–1047, Sept. 1999.

[27] J. T. Y. Kwok, "Moderating the output of support vector machine classifier," *IEEE Transactions on Neural Networks*, pp. 1018–1031, Sept. 1999.

[28] H. Drucker, D. Wu and V. N. Vapnik, "Support vector machines for spam categorisation," *IEEE Transactions on Neural Networks*, Vol. 10, pp. 1048–1054, Sept. 1999.

[29] S. Ben-Yacoub, Y. Abdeljaoued and E. Mayoraz, "Fusion of face and speech data for person identity verification," *IEEE Transactions on Neural Networks*, Vol. 10, pp. 1065–1074, Sept. 1999.

[30] T. Furey, N. Christianini, N. Duffy, D. Bednarski, M. Schummer and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, Vol. 16, pp. 906–914, 2000.

[31] A. Zien, G. Ratsch, S. Mika, B. Scholkopf, T. Lengauer and K. R. Muller, "Engineering support vector machine kernels that recognize translation initiation sites in DNA," *Bioinformatics*, Vol. 16, pp. 799–807, 2000.

[32] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.