8. Support Vector Machines

8.1 Introduction

In previous lectures, we studied multilayer perceptron (MLP) neural network trained with the back-propagation algorithm. We also studied another class of feed-forward neural network, radial basis function (RBF) neural network. Both of these neural networks are universal approximators in their own ways. In this lecture, we study another category of universal feed-forward neural networks, known as support vector machines (SVM), pioneered by Vapnic (1992) etc. Like MLP and RBF neural networks, SVM can be used for pattern classification and nonlinear regression.

Basically, SVM is a linear machine with some very nice properties. To explain how it works, it is perhaps easiest to start with the case of separable patterns that could arise in the context of pattern classification.

8.2 Optimal Hyperplane for Linearly Separable Patterns

Consider N training samples:

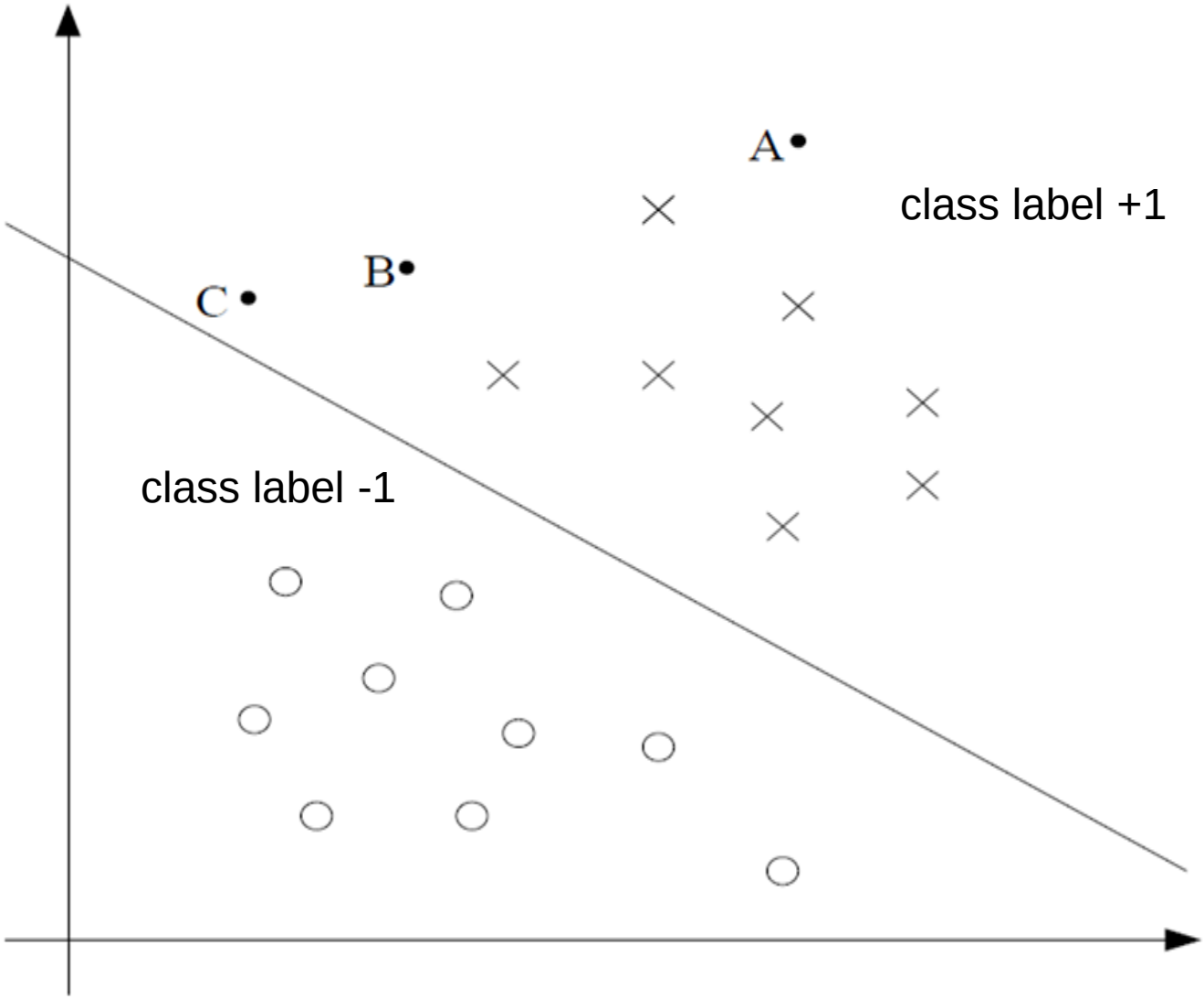
$$\{\mathbf{x}(1), d(1)\}, \{\mathbf{x}(2), d(2)\}, \cdots, \{\mathbf{x}(N), d(N)\}$$

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \cdots, x_n(i)]^T$$

d(i) is the class label of sample **x**(i). It is assumed that d(i) takes the value of +1 or -1.

The objective of pattern classification is to find a decision surface in the form of hyperplane to separate the data of the two classes.



Point A is far from the separating hyperplane. If we are to make a prediction for A, we are quite confident that its label is +1. Point C is very close to the hyperplane, and while we would predict +1, it seems likely that just a small change to the separating plane could easily cause our prediction to be -1. Hence, we're much more confident about our prediction at A than at C. More broadly, we see that if a point is far from the separating hyperplane, then we may be significantly more confident in our predictions. It would be nice if we manage to find a separating plane that allows us to make all correct and confident (meaning far from the seperating hyperplane).

The equation of the hyperplane is given as follows:

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

Where **x** is the input, **w** is the adjustable weight vector, and b is a bias.

We may thus write:

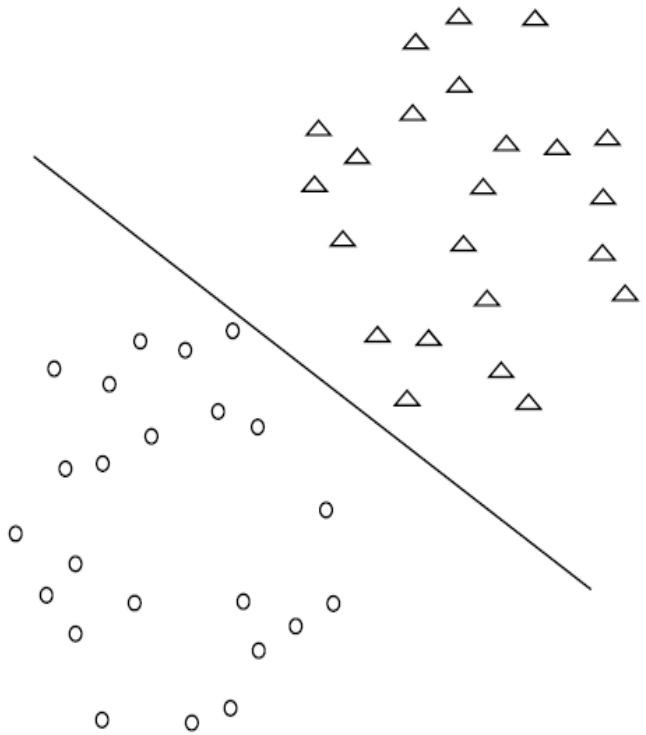
$$\mathbf{w}^T\mathbf{x} + b \geq 0 \qquad \text{for d} = +1$$

$$\mathbf{w}^T\mathbf{x} + b < 0 \qquad \text{for d} = -1$$

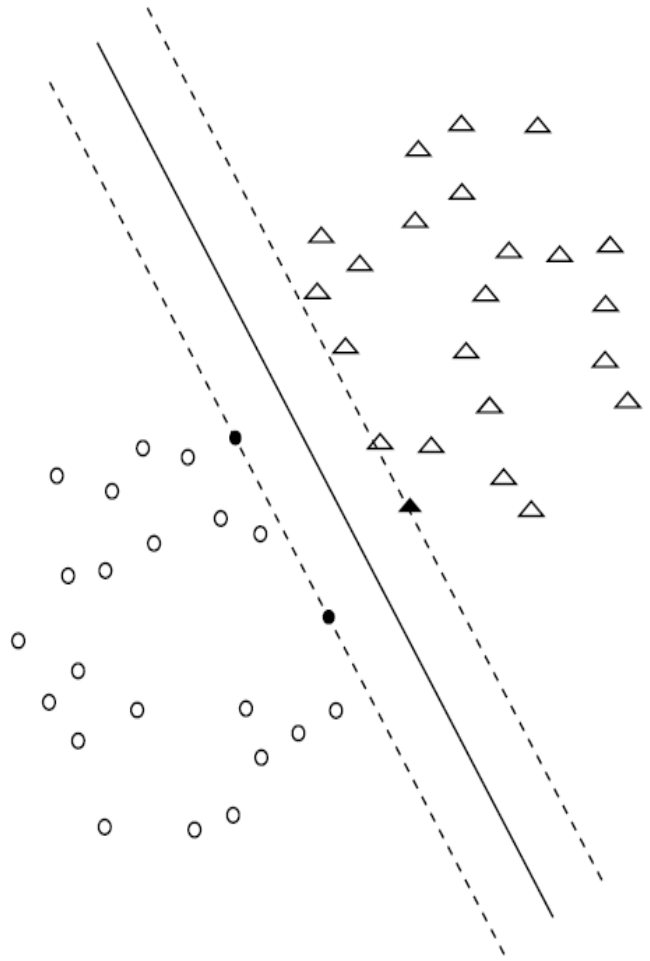
For a given weight vector **w** and bias *b*, the separation between the hyperplane and the closest data point is called the **margin of separation**.

The goal of a support vector machine is to find the particular hyperplane for which the separation is maximized. Under this condition, the decision surface is referred to as the **optimal hyperplane**.

The following diagram illustrates the geometric construction of such an optimal hyperplane for a two-dimensional input space.



(a)



(b)

The discriminant function

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

gives an algebraic measure of the distance from $\mathbf{x}$ to the hyperplane. Perhaps the easiest way to see this is to express $\mathbf{x}$ as:

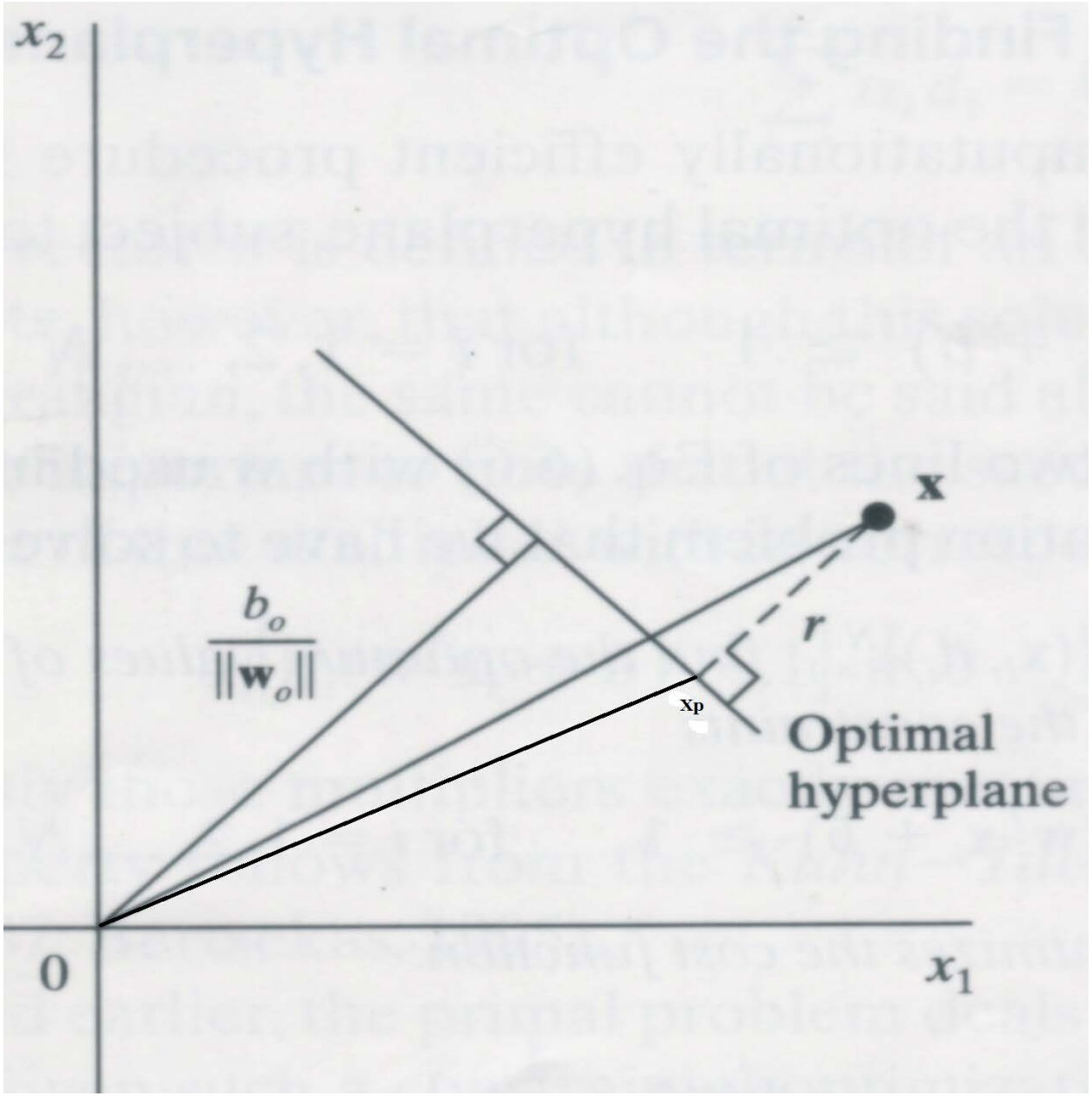
$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Where $\mathbf{x}_p$ is the projection of $\mathbf{x}$ onto the hyperplance, and r is the distance. r is positive if $\mathbf{x}$ is on the positive side of the optimal hyperplane, and negative if $\mathbf{x}$ is on the negative side. By definition, $g(\boldsymbol{x}_p) = 0$. Then we have:

$$g(\mathbf{x}) = r\|\mathbf{w}\|$$

or

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$


x_2
x_1
0
$\frac{b_o}{\|\mathbf{w}_o\|}$
x
r
Xp
Optimal hyperplane

In particular, the distance from the origin (**x**=0) to the optimal hyperplane is given by $b/\|\boldsymbol{w}\|$. If $b > 0$, the origin is on the positive side, and if $b < 0$, the origin is on the negative side. If $b = 0$, the optimal hyperplane passes through the origin.

If the data are linearly separable, we can always scale **w** and b so that

$$\mathbf{w}^T\mathbf{x} + b \geq 1 \qquad \text{for d} = +1$$

$$\mathbf{w}^T\mathbf{x} + b \leq -1 \qquad \text{for d} = -1$$

The particular data points for which the above is satisfied with equality sign are called **support vectors**. These vectors play a prominent role in the operation of this class of learning machines. In conceptual terms, the support vectors are those data points that lie closest to the decision surface and therefore the most difficult to classify.

Thus, the algebraic distance from support vector $\mathbf{x}^{(s)}$ to the optimal hyperplane is:

$$r = \frac{g(\mathbf{x}^{(s)})}{\|\mathbf{w}\|} = \begin{cases} \dfrac{1}{\|\mathbf{w}\|} & \text{for } d^{(s)}=+1 \\ \dfrac{-1}{\|\mathbf{w}\|} & \text{for } d^{(s)}=-1 \end{cases}$$

Where the plus sign indicates that $\mathbf{x}^{(s)}$ lies on the positive side of the optimal hyperplane, and the minus sign indicates that $\mathbf{x}^{(s)}$ lies on the negative side of the optimal hyperplane. The margin of separation between the two classes is given by:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Thus, maximizing the margin of separation between classes is equivalent to minimizing the Euclidean norm of the weight vector **w**.

Our goal is to develop a computationally efficient procedure to find the optimal hyperplane, subject to the constraint:

$$d \times (\mathbf{w}^T \mathbf{x} + b) \geq 1$$

Thus, the constrained optimization problem that we have to solve may now be stated as:

Given the training samples {**x**(i),d(i)}, i=1,2,…,N, find the optimal values of the weight vector **w** and bias b such that they satisfy the constraints

$$d(i) \times [\mathbf{w}^T \mathbf{x}(i) + b] \geq 1 \qquad \text{for i} = 1,2,\ldots,\text{N}$$

and the weight vector minimizes the following cost function:

$$J(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

This constrained optimization problem has the following characteristics:

(1) The cost function $J(\mathbf{w})$ is a convex function of $\mathbf{w}$;
(2) The constraints are linear in $\mathbf{w}$.

Accordingly, we may solve the constrained optimization problem using the method of Lagrange multipliers. First, we construct the Lagrange function:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha(i)[d(i)(\mathbf{w}^T\mathbf{x}(i) + b) - 1]$$

Where the auxiliary nonnegative variables α(i) are called Lagrange multipliers. The solution to the constrained optimization problem is determined by the saddle points of the Lagrange function $J(\boldsymbol{w}, b, \alpha)$, which has to be minimized with respect to $\mathbf{w}$ and b. It also has to be maximized with respect to α(i).

Differentiating $J(\mathbf{w}, b, \alpha)$ with respect to $\mathbf{w}$ and b and setting the results to zero, we get the following two conditions of optimality:

Condition 1: $$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0$$

Condition 2: $$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0$$

From Conditions 1 and 2, we have:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha(i) d(i) \mathbf{x}(i)$$

$$\sum_{i=1}^{N} \alpha(i) d(i) = 0$$

Following Karush-Kuhn-Tucker (KKT) optimization theory, at the optimal solution, we have:

$$\alpha(i)[d(i)(\mathbf{w}^T\mathbf{x}(i)+b)-1]=0$$

$$\alpha(i)\geq 0 \qquad \text{for i} = 1,2,\ldots,\text{N}$$

Which means that α(i) will be nonzero (actually it is positive) only for the points who satisfy the equality in the constraint.

We first expand $J(\boldsymbol{w},b,\alpha)$:

$$J(\mathbf{w},b,\alpha)=\frac{1}{2}\mathbf{w}^T\mathbf{w}-\sum_{i=1}^{N}\alpha(i)d(i)\mathbf{w}^T\mathbf{x}(i)-b\sum_{i=1}^{N}\alpha(i)d(i)+\sum_{i=1}^{N}\alpha(i)$$

The third term in the above expansion is zero by the optimality condition 2. Furthermore, by the optimality condition 1, we have:

$$\mathbf{w}^T\mathbf{w}=\sum_{i=1}^{N}\alpha(i)d(i)\mathbf{w}^T\mathbf{x}(i)=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha(i)\alpha(j)d(i)d(j)\mathbf{x}^T(i)\mathbf{x}(j)$$

Substituting the above into the expansion of $J(\boldsymbol{w}, b, \alpha)$ yields:

$$J(\mathbf{w}, b, \alpha) = \sum_{i=1}^{N} \alpha(i) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha(i)\alpha(j) d(i) d(j) \mathbf{x}^T(i)\mathbf{x}(j)$$

Thus, we may solve the dual problem, whose solution is equal to the solution of the original problem (often referred to as primal problem). The dual problem is stated as follows:

$$Q(\alpha) = \sum_{i=1}^{N} \alpha(i) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha(i)\alpha(j) d(i) d(j) \mathbf{x}^T(i)\mathbf{x}(j)$$

Subject to the conditions:

(1) $\sum_{i=1}^{N} \alpha(i) d(i) = 0$

(2) $\alpha(i) \geq 0$

Having determined the optimum Lagrange multipliers, we may compute the optimal weight vector **w** and b:

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha(i) d(i) \mathbf{x}(i)$$

$$b^* = 1 - \mathbf{w}^* \mathbf{x}^{(s)} \qquad \text{for } d^{(s)} = 1$$

8.3 Optimal Hyperplane for Nonseparable Patterns

The discussion thus far has focused on linearly separable patterns. In this section, we consider the more difficult cases of nonseparable patterns. Given such a set of training data, it is impossible to construct a separating hyperplane without encountering classification errors. Nevertheless, we would like to find an optimal hyperplane that minimises the classification error.

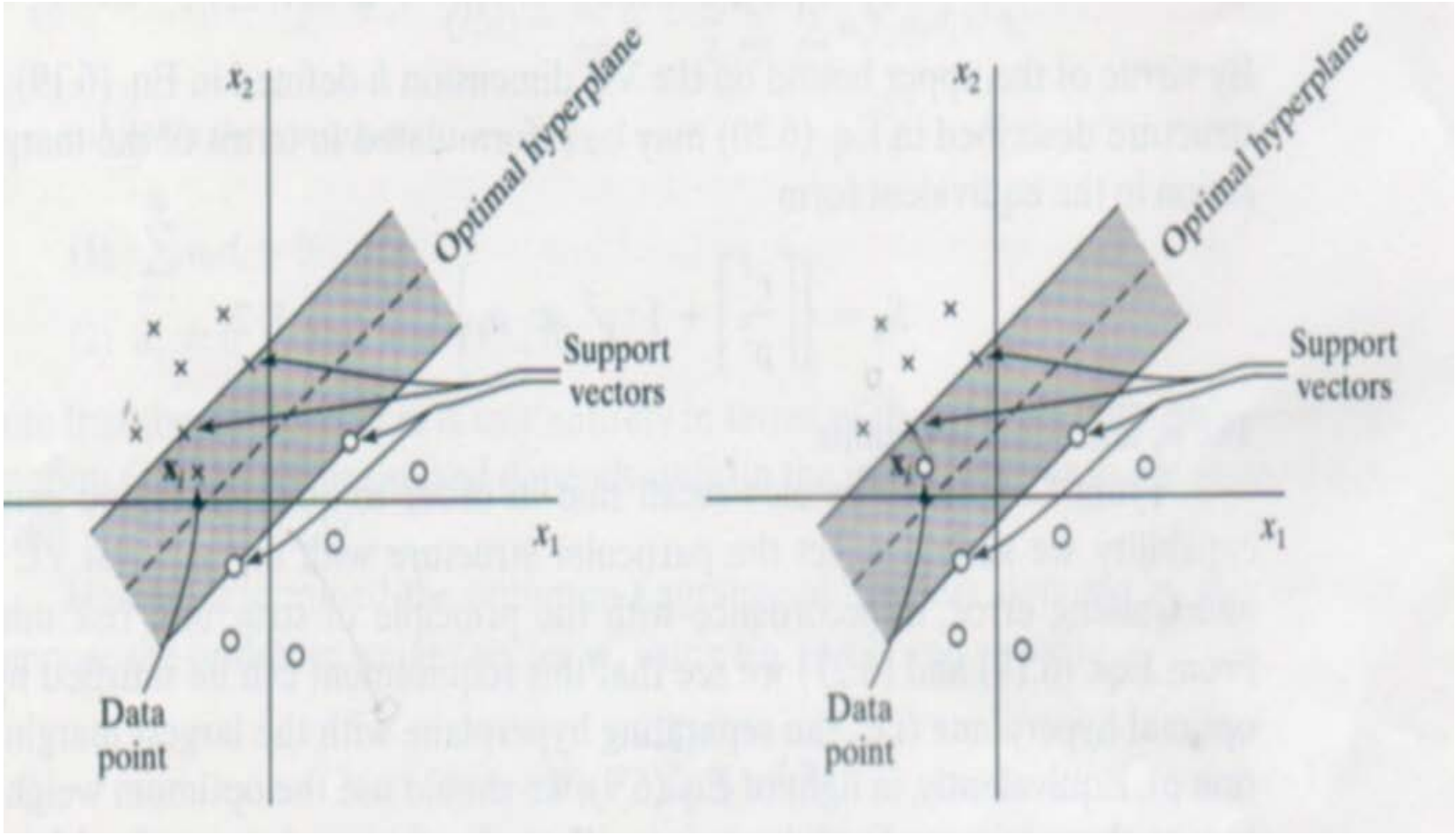
The margin of separation between classes is said to be soft if a data point {**x**(i),d(i)} violates the following conditions:

$$d(i) \times [\mathbf{w}^T \mathbf{x}(i) + b] \geq 1 \qquad \text{i} = 1,2,\ldots,\text{N}$$

The violation can arise in one of two ways:

(a) The data point {**x**(i),d(i)} falls inside the region of separation but on the correct side of the hyperplane.

(b) The data point {**x**(i),d(i)} falls on the wrong side of the hyperplane.



(a) correct side

(b) wrong side

To treat the nonseparable data points, we introduce a new set of nonnegative scalar variables ζ(i), i=1,2,…,N, into the definition of the separating hyperplane as shown below:

$$d(i) \times [\mathbf{w}^T \mathbf{x}(i) + b] \geq 1 - \xi(i) \qquad \text{i} = 1,2,\ldots,\text{N}$$

The ζ(i) are called slack variables, which measure the deviation of a data point from the ideal condition of pattern separability. For $0 \leq \zeta(i) \leq 1$, the data point falls inside the region of separation but on the correct side of the hyperplane. For ζ(i)>1, it falls on the wrong side of the separating hyperplane.

Our goal is to find a separating hyperplane that minimises the classification error averaged over the training data. We may do this by minimizing the following function:

$$J(\mathbf{w},\xi)=\frac{1}{2}\mathbf{w}^T\mathbf{w}+C\sum_{i=1}^{N}\xi(i)$$

Parameter C has to be selected by user. Often it is determined experimentally via the standard use of a training/testing set.

Using Lagrange multipliers method, we may again formulate the dual problem for nonseparable patterns as follows:

$$Q(\alpha)=\sum_{i=1}^{N}\alpha(i)-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha(i)\alpha(j)d(i)d(j)\mathbf{x}^T(i)\mathbf{x}(j)$$

Subject to the conditions:

(1) $$\sum_{i=1}^{N}\alpha(i)d(i)=0$$

(2) $$0\le\alpha(i)\le C$$

The optimal weight vector **w** and b are given by:

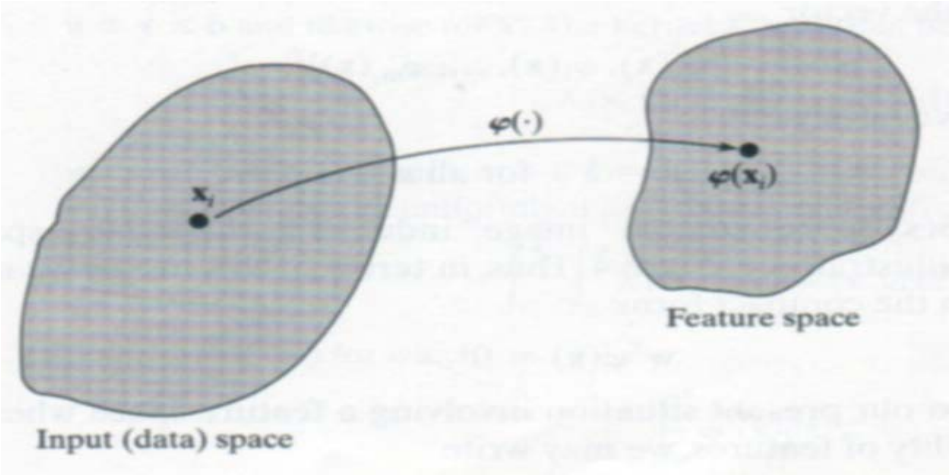
$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha(i) d(i) \mathbf{x}(i)$$

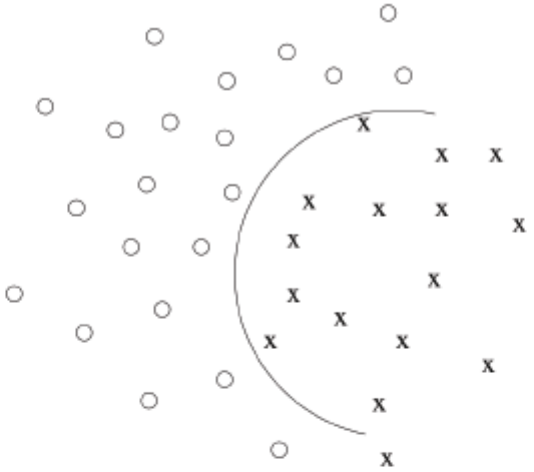
$$b^* = 1 - \mathbf{w}^* \mathbf{x}^{(s)} \qquad \text{for } d^{(s)} = 1$$

8.4 Support Vector Machines for Nonlinear Pattern Classification

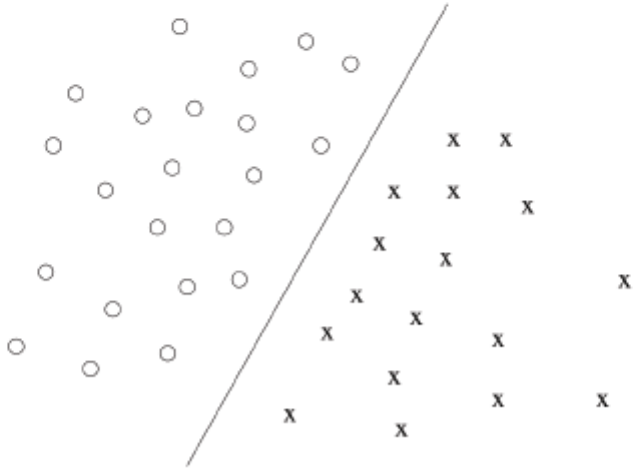
In the above, linear SVM is presented. If the data is linearly inseparable, how to construct a classifier to classify the data? This can be solved as follows:

1. Nonlinear mapping of an input vector into a high-dimensional feature space;
2. Construction of an optimal hyperplane for separating the data in the high-dimensional feature space.





Non-linear separator in the original $\mathbf{x}$-space



Linear separator in the feature ϕ-space

The first step operation is in accordance with Cover's theorem on the separability of patterns. Consider an input space made up of nonlinearly separable patterns. Cover's theorem states that such a multi-dimensional space may be transformed into a new feature space where the patterns are linearly separable with high probability, provided two conditions are satisfied:

(a) The transformation is nonlinear.
(b) The dimensionality of the feature space is high enough.

The second step operation exploits the ideas of building an optimal separating hyperplane in accordance with the previously discussed linear SVM, but with a fundamental difference: the hyperplane is now defined as a linear function of vectors drawn from the feature space rather than the original input space.

Let $\varphi_j(\mathbf{x})$, $j=1,2,\ldots,m$, denotes a set of nonlinear transformations from the input space to the feature space, where m is the dimension of the feature space. It is assumed that $\varphi_j(\mathbf{x})$ is defined a *priori* for all j. Given such a set of nonlinear transformations, we may define a hyperplane acting as the decision surface as follows:

$$\sum_{j=1}^{m} w_j \varphi_j(\mathbf{x}) + b = 0$$

where w_j , $j=1,2,\ldots,m$, denotes a set of weights linearly connecting the feature space to the output space, and b is the bias. The above hyperplane in the feature space can also be written as:

$$\sum_{j=0}^{m} w_j \varphi_j(\mathbf{x}) = 0$$

where w_0 is the bias b, and $\varphi_0(\mathbf{x}) = 1$ for any $\mathbf{x}$.

Define the vector:

$$\boldsymbol{\varphi}(\mathbf{x}) = \left[\varphi_0(\mathbf{x}) \quad \varphi_1(\mathbf{x}) \quad \cdots \quad \varphi_m(\mathbf{x})\right]^T$$

The decision surface is:

$$\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}) = 0$$

Adapting the optimality condition of linear SVM to the present situation involving a feature space where we now seek "linear" separability of features, we may write:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha(i) d(i) \boldsymbol{\varphi}[\mathbf{x}(i)]$$

Thus, the decision surface in the feature space is:

$$\sum_{i=1}^{N} \alpha(i) d(i) \boldsymbol{\varphi}^T[\mathbf{x}(i)] \boldsymbol{\varphi}(\mathbf{x}) = 0$$

The term $\boldsymbol{\varphi}^T[\mathbf{x}(i)]\boldsymbol{\varphi}(\mathbf{x})$ represents the inner product of two vectors induced in the feature space by the input vector **x** and the input pattern **x**(i). We introduce the inner-product kernel denoted by $K[\mathbf{x}, \mathbf{x}(\mathrm{i})]$ and defined by:

$$K[\mathbf{x}, \mathbf{x}(i)] = \boldsymbol{\varphi}^T(\mathbf{x})\boldsymbol{\varphi}[\mathbf{x}(i)]$$

$$= \sum_{j=0}^{m} \varphi_j(\mathbf{x})\varphi_j[\mathbf{x}(i)]$$

From this definition, we immediately see that the inner product kernel is a symmetric function of its arguments:

$$K[\mathbf{x}, \mathbf{x}(i)] = K[\mathbf{x}(i), \mathbf{x}]$$

Most importantly, we may use the inner-product kernel to construct the optimal hyperplane in the feature space without having to consider the feature space itself in explicit form:

$$\sum_{i=1}^{N} \alpha(i) d(i) K[\mathbf{x}(i), \mathbf{x}] = 0$$

Mercer's Theorem

Let $K(\mathbf{x}, \mathbf{x}')$ be a continuous symmetric kernel that is defined in the closed interval $\mathbf{a} \leq \mathbf{x} \leq \boldsymbol{b}$ and likewise for $\mathbf{x}'$. The kernel $K(\mathbf{x}, \mathbf{x}')$ can be expanded in the series:

$$K[\mathbf{x}, \mathbf{x}'] = \sum_{i=1}^{\infty} \lambda_i \varphi_i(\mathbf{x}) \varphi_i(\mathbf{x}')$$

With positive coefficients $\lambda_i > 0$. For this expansion to be valid and for it to converge absolutely and uniformly, it is necessary and sufficient that the following condition holds

$$\int_{\mathbf{b}}^{\mathbf{a}} \int_{\mathbf{b}}^{\mathbf{a}} K(\mathbf{x}, \mathbf{x}') \psi(\mathbf{x}) \psi(\mathbf{x}') d\mathbf{x} d\mathbf{x}' \geq 0$$

for all $\psi(.)$ for which

$$\int_{\mathbf{b}}^{\mathbf{a}} \psi^2(\mathbf{x}) d\mathbf{x} < \infty$$

The functions $\varphi_i(\mathbf{x})$ are called eigenfunctions of the expansion and the numbers λ_i are called eigenvalues. The fact that all of the eigenvalues are positive means that the kernel $K(\mathbf{x}, \mathbf{x}')$ is positive definite.

In light of Mercer's theorem, we may now make the following observations:

(a) For $\lambda_i \neq 1$, the i-th image $\sqrt{\lambda_i}\varphi_i(\mathbf{x})$ induced in the feature space by the input vector $\mathbf{x}$ is an eigenfunction of the expansion.

(b) In theory, the dimensionality of the feature space (i.e. the number of eigen-values/functions) can be infinitely large.

Mercer's theorem only says whether or not a candidate kernel is actually an inter-product kernel in some space and therefore admissible for use in support vector machine. However, it says nothing about how to construct the function $\varphi_i(\mathbf{x})$.

Design of a kernel support vector machine

The expansion of the inner-product kernel $K(\mathbf{x}, \mathbf{x}')$ permits us to construct a decision surface that is nonlinear in the input space, but its image in the feature space is linear. With this expansion at hand, we may now state the dual form for the constrained optimization of a support vector machine as follows:

Find the Lagrange multipliers that maximize the following objective function:

$$Q(\alpha) = \sum_{i=1}^{N} \alpha(i) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha(i)\alpha(j)d(i)d(j)K(\mathbf{x}_i, \mathbf{x}_j)$$

Subject to the conditions:

(1) $$\sum_{i=1}^{N} \alpha(i)d(i) = 0$$

(2) $$0 \leq \alpha(i) \leq C$$

The dual problem of kernel SVM is of the same form as that of linear SVM for nonseparable patterns, except for the fact that the inner product $\mathbf{x}^T(i)\mathbf{x}(j)$ used in linear SVM is replaced by the inner-product kernel $K[\mathbf{x}(i),\mathbf{x}(j)]$.

Having found the optimum values of the Lagrange multipliers, denoted by α(i), we may compute the decision value of a given input vector **x**:

$$y = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x})$$

$$= \sum_{i=1}^{N} \alpha(i) d(i) K[\mathbf{x}(i), \mathbf{x}]$$

In practice, we do not compute the weight vector **w** explicitly because the feature space is hidden, and $\boldsymbol{\varphi}[\mathbf{x}(i)]$ is unavailable.

$$\mathbf{w} = \sum_{i=1}^{N} \alpha(i) d(i) \boldsymbol{\varphi}[\mathbf{x}(i)]$$

Kernel Functions in SVM

The requirement on kernel $K(\mathbf{x}, \mathbf{x}')$ is to satisfy Mercer's theorem. Within this requirement, there is some freedom in how it is chosen. Two inner-product kernels are often used:

(i) Polynomial kernel

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^p$$

where p is a positive integer, and is specified *a priori* by the user.

(ii) Gaussian kernel

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

where σ is the width of the Gaussian kernel, and is specified *a priori* by the user.

The following points are noteworthy:

(i) The inner-product kernels of polynomial and Gaussian functions always satisfy Mercer's theorem.

(ii) For the two types of inner product, the dimensionality of the feature space is determined by the number of support vectors extracted from the training data by the solution to the constrained optimization problem.

(iii) The underlying theory of a support vector machine avoids the need for heuristics often used in the design of conventional radial basis function neural networks.

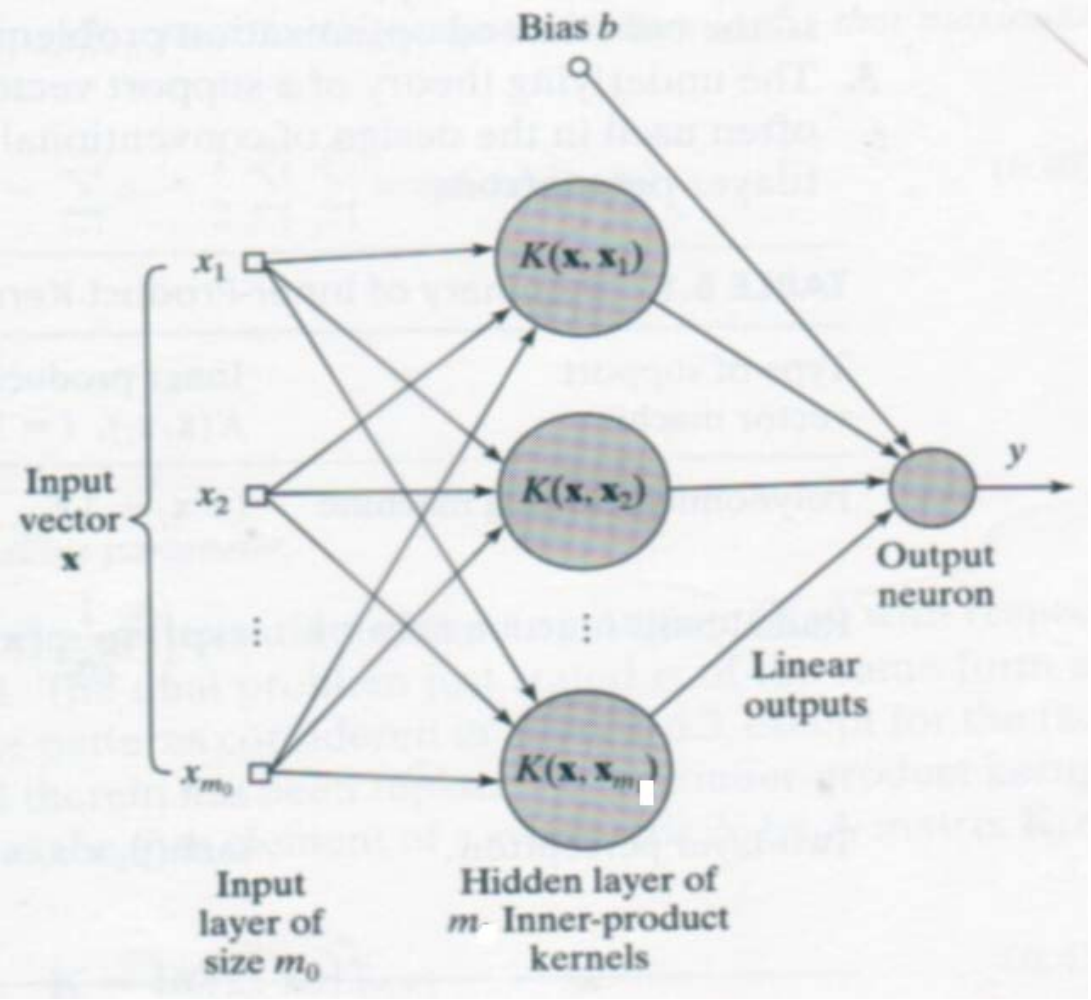
(iv) In Gaussian kernel SVM, the number of the neurons and their centre vectors are determined automatically by the number of support vectors and their values.

The support vector machine differs from the conventional approach to the design of a feed-forward neural network in a fundamental way. In the conventional approach, model complexity is controlled by keeping the number of hidden layer neurons small. On the other hand, the support vector machine offers a solution to the design of a learning machine by controlling model complexity independent of dimensionality as summarized below:

(i) Conceptual problem. Dimensionality of the feature (hidden) space is purposely made very large to enable the construction of a decision surface in the form of a hyperplane in that space. For good generalization performance, the model complexity is controlled by imposing certain constraints on the construction of the separating hyperplane, which results in the extraction of a fraction of the training data as support vectors.

(ii) Computational problem. Numerical optimization in a high-dimensional space suffers from high dimensionality. This computational problem is avoided by using the notion of an inner-product kernel (defined in accordance with Mercer's theorem) and solving the dual form of the constrained in the input space. This is often called "kernel trick".

Architecture of a support vector machine



An illustrative Example of Kernel SVM Design

To illustrate the procedure for the design of a support vector machine, we revisit the XOR (eXclusive OR) problem, which is summarized below:

Index of data	input vector **x**	desired output d
1	$[1\ 1]^T$	-1
2	$[-1\ 1]^T$	1
3	$[-1\ -1]^T$	-1
4	$[1\ -1]^T$	1

To proceed, assuming the polynomial kernel is used:

$$K[\mathbf{x}, \mathbf{x}(i)] = [1 + \mathbf{x}^T \mathbf{x}(i)]^2$$

The outputs of the kernel function for the 4 input vectors are:

$$\mathbf{K} = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$$

Thus, the objective function for the dual problem is

$$Q(\alpha) = \alpha(1) + \alpha(2) + \alpha(3) + \alpha(4) - \frac{1}{2}[9\alpha^2(1) - 2\alpha(1)\alpha(2) - 2\alpha(1)\alpha(3) + 2\alpha(1)\alpha(4)$$
$$+9\alpha^2(2) + 2\alpha(2)\alpha(3) - 2\alpha(2)\alpha(4) + 9\alpha^2(3) - 2\alpha(3)\alpha(4) + 9\alpha^2(4)]$$

Optimizing Q(α) with respect to the Lagrange multipliers yields the following set of simultaneous equations:

$$9\alpha(1) - \alpha(2) - \alpha(3) + \alpha(4) = 1$$
$$-\alpha(1) + 9\alpha(2) + \alpha(3) - \alpha(4) = 1$$
$$-\alpha(1) + \alpha(2) + 9\alpha(3) - \alpha(4) = 1$$
$$\alpha(1) - \alpha(2) - \alpha(3) + 9\alpha(4) = 1$$

Thus, the optimal values of the Lagrange multipliers are:

$$\alpha(1) = \alpha(2) = \alpha(3) = \alpha(4) = \frac{1}{8}$$

The output of the kernel SVM for an input vector **x** is:

$$\begin{aligned} y &= \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) \\ &= \sum_{i=1}^{4} \alpha(i) d(i) K[\mathbf{x}(i), \mathbf{x}] \end{aligned}$$

For the 4 input vectors, the outputs are:

$$y(1) = \frac{1}{8} * (-1) * 9 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (-1) * 1 = -1$$

$$y(2) = \frac{1}{8} * (-1) * 1 + \frac{1}{8} * (+1) * 9 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (-1) * 1 = 1$$

$$y(3) = \frac{1}{8} * (-1) * 1 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (+1) * 9 + \frac{1}{8} * (-1) * 1 = 1$$

$$y(4) = \frac{1}{8} * (-1) * 1 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (+1) * 1 + \frac{1}{8} * (-1) * 9 = -1$$

Solving Dual Problem of SVM.

In SVM, the primal problem is converted into the dual problem, which is a quadratic programming (QP) problem whose objective function is solely dependent on a set of Lagrange multipliers $\alpha(i)$, $i=1,2,\ldots,N$.

Due to its immense size, the QP problem that arises from SVMs cannot be easily solved via standard QP techniques. Not long ago, the use of SVMs was still limited to a small group of researchers. One possible reason is that training algorithms for SVMs are slow, especially for large problems. Another explanation is that SVM training algorithms are complex, subtle, and difficult for an average engineer to implement.

To address the computation efficiency problem, sequential minimal optimization (SMO) algorithm was proposed, which is 1000 times faster than conventional methods. Many public available toolboxes such as LIBSVM implement SMO.

Public Available Software for SVM

mySVM
http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html
mySVM, by Stefan Rüping, is a C++ implementation of SVM classification and regression. Available as C++ source code and Windows binaries. Kernels: linear, polynomial, radial basis function, neural (tanh), anova.

JmySVM
http://www-ai.cs.uni-dortmund.de/SOFTWARE/YALE/index.html
JmySVM, a Java version of mySVM is part of the YaLE (Yet Another Learning Environment) learning environment.

mySVM/db
http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVMDB/index.html
mySVM/db is an efficient extension of mySVM which is designed to run directly inside a relational database using an internal JAVA engine. It was tested with an Oracle database, but with small modifications it should also run on any database offering a JDBC interface. It is especially useful for large datasets available as relational databases.

LIBSVM
http://www.csie.ntu.edu.tw/~cjlin/libsvm/
LIBSVM (Library for Support Vector Machines), is developed by Chang and Lin and contains *C*-classification, v-classification, ε-regression, and v-regression. Developed in C++ and Java, it supports also multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection. It has interfaces for Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW. Kernels: linear, polynomial, radial basis function, and neural (tanh).

looms
http://www.csie.ntu.edu.tw/~cjlin/looms/
looms, by Lee and Lin, is a very efficient leave-one-out model selection for SVM two-class classification. While LOO cross-validation is usually too time consuming to be performed for large datasets, looms implements numerical procedures that make LOO accessible. Given a range of parameters, looms automatically returns the parameter and model with the best LOO statistics. Available as C source code and Windows binaries.

BSVM
http://www.csie.ntu.edu.tw/~cjlin/bsvm/
BSVM, authored by of Hsu and Lin, provides two implementations of multi-class classification, together with SVM regression. Available as source code for UNIX/Linux and as binaries for Windows.

Hopfield neural network

1. Architecture
2. Operations performed in neurons
3. Weight calculation
4. Storage capacity

BAM neural network

1. Architecture---know how to determine number of neurons in each layer

- Can be used to implement mapping from Pattern i to Pattern j

- Can be used to store pattern by mapping Pattern i to Pattern i (i.e. itself)

2. Operations performed in neurons

3. Weight calculations

SOM neural network

1. Architecture
2. Characteristics
3. Training procedure
4. Setting of parameters
5. Properties

RBF neural network

1. Architecture
2. Operations performed in neurons
3. Weight estimation method
4. Neuron centre vector determination methods

Multilayer perceptron (MLP) neural network

1. Architecture

2. Working principle

3. Back-propagation algorithm

- Weight updating rule
- Local gradient of hidden layer neuron
- Local gradient of output layer neuron

Support vector machines

1. Primal optimization problem of linear SVM

2. Dual optimization problem of linear SVM

- For linear separable patterns, should be able to derive dual problem from primal problem

3. Kernel SVM