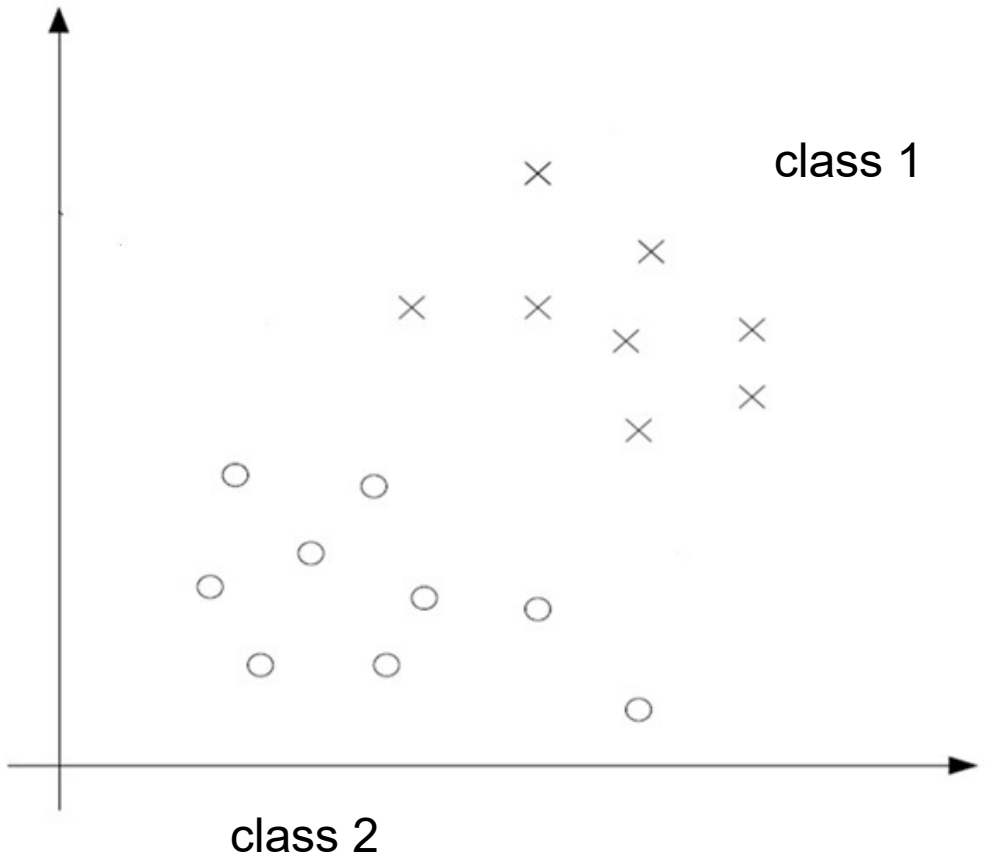
6. Support Vector Machines

6.1 Introduction

In previous part, radial basis function (RBF) neural network is introduced. In this lecture, we study another type of feedforward neural network, known as support vector machines (SVM). Like RBF neural networks, SVM can be used for pattern classification and regression.

Kernel support vector machine has a very similar architecture to the RBF neural networks, but their learning principles are very different. Kernel SVM is an extension of linear SVM. Next, we first introduce the principle of linear support vector machines (SVM), and then generalize the principle to kernel SVM.

To explain how linear SVM works, it is perhaps easiest to start with a pattern classification problem, where samples from different classes are separable (meaning we can draw a hyperplane to separate the samples in the space):



6.2 Optimal Hyperplane for Linearly Separable Patterns

Consider N training samples:

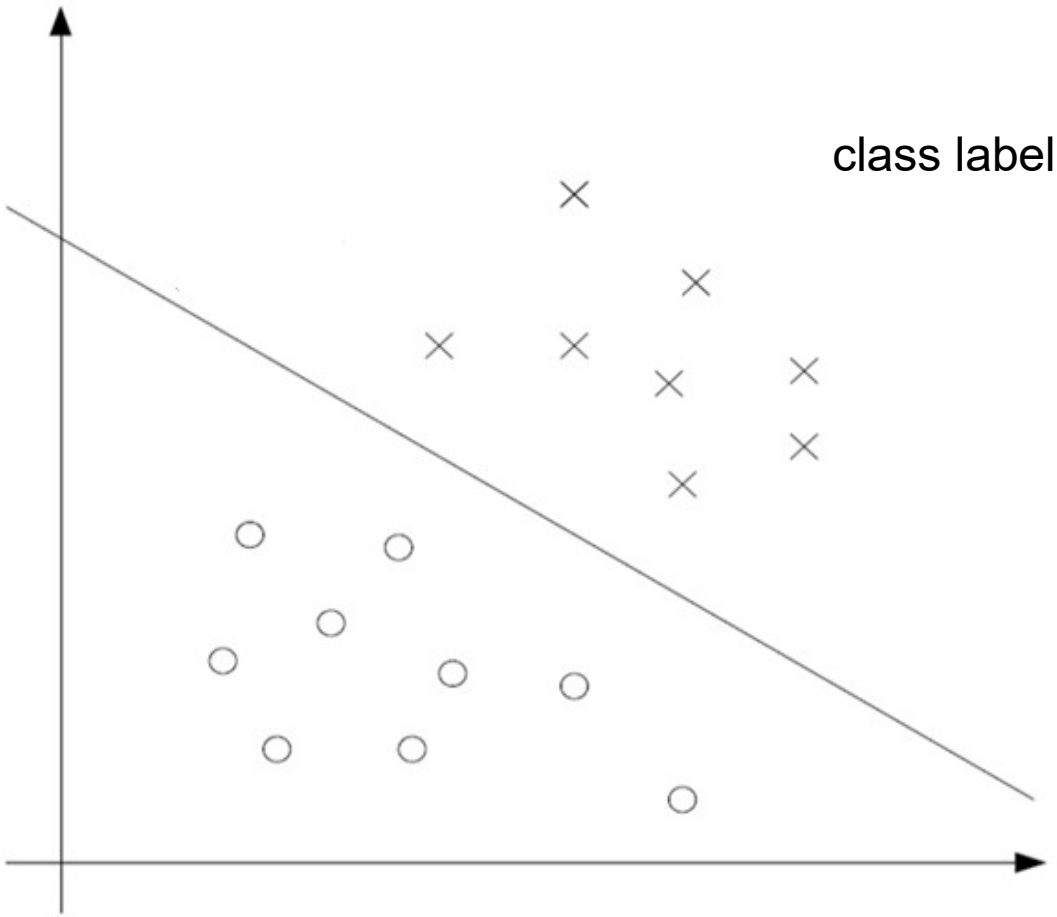
$$\{\mathbf{x}(1), d(1)\}, \{\mathbf{x}(2), d(2)\}, \cdots, \{\mathbf{x}(N), d(N)\}$$

where

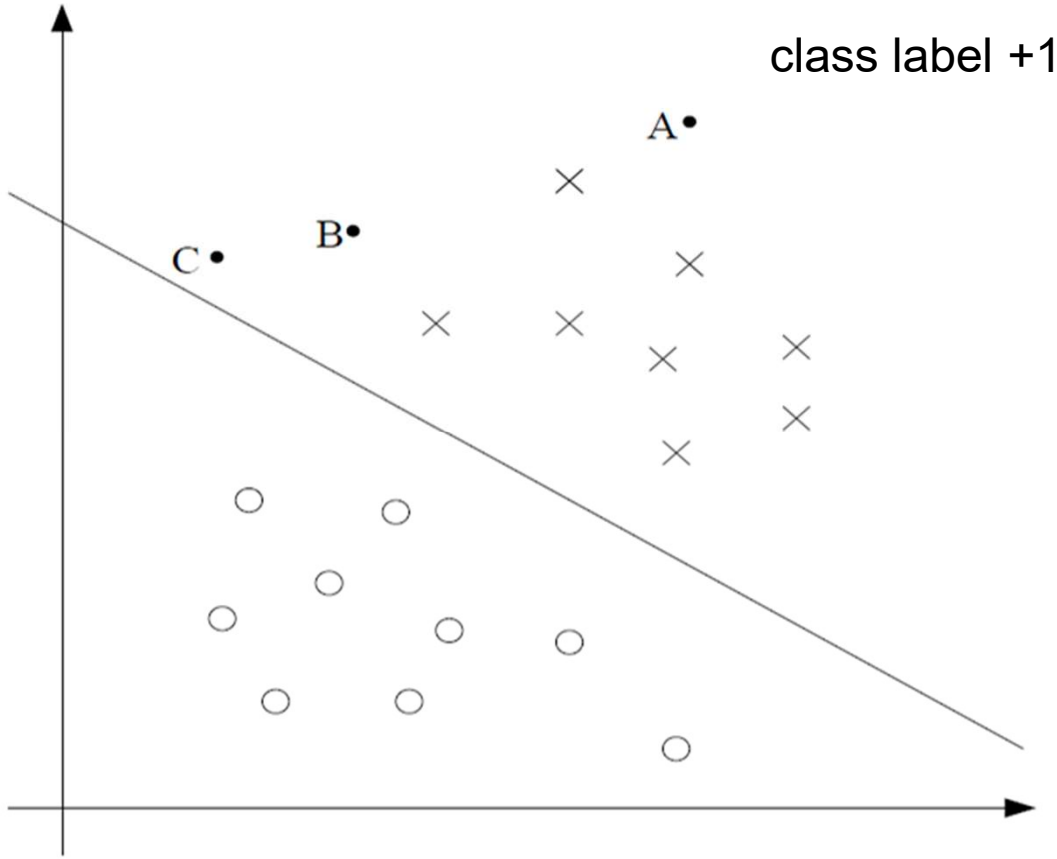
$$\mathbf{x}(i) = [x_1(i), x_2(i), \cdots, x_n(i)]^T$$

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 belonging to different classes shown below.


class label +1
class label -1

Assuming that we have three test data points A, B, C. We now need to predict the class labels for the three data point.

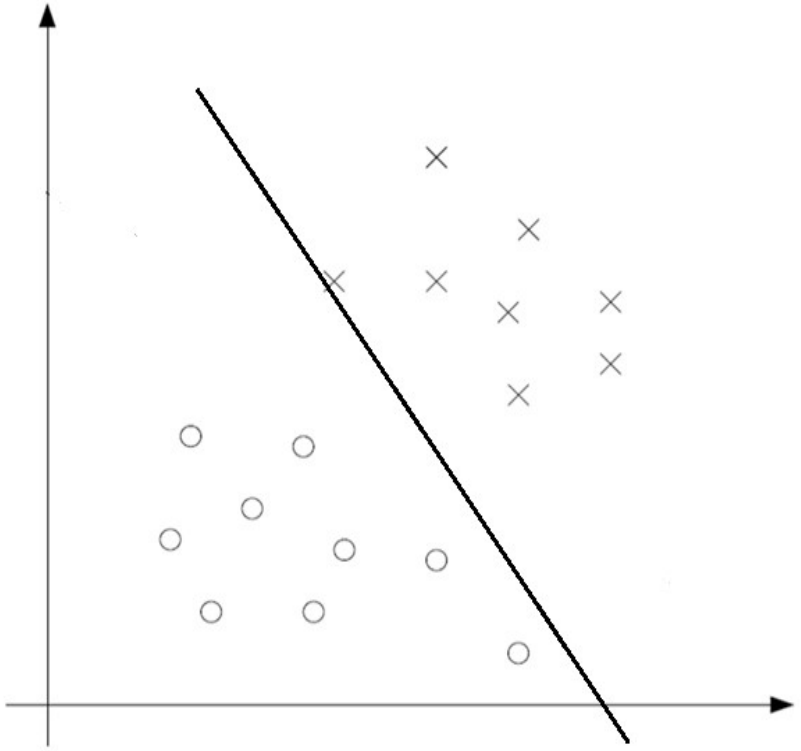


Point A is far from the separating hyperplane. If we are to make a prediction for A, we are quite confident that its label is +1.

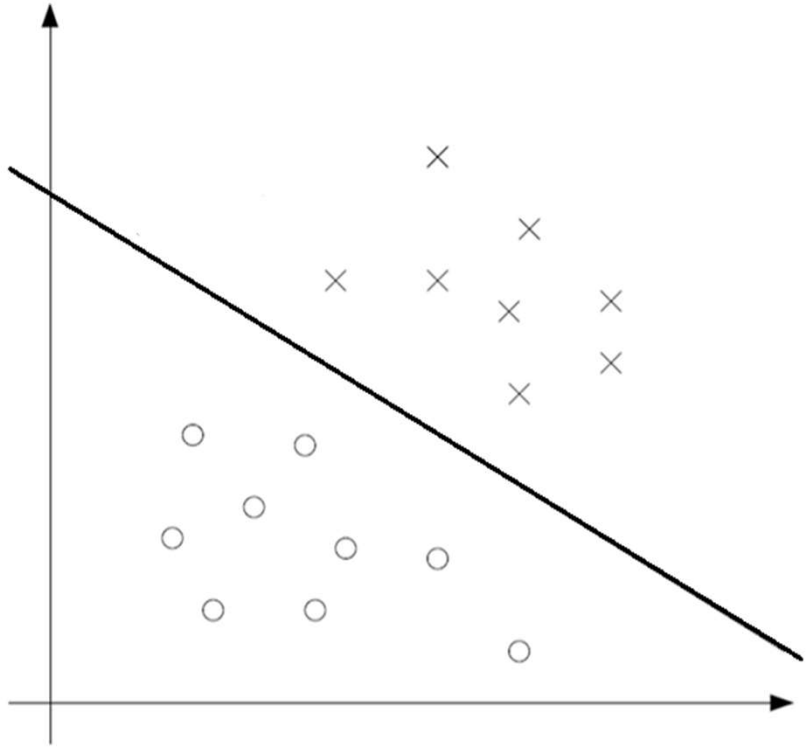
Point C is very close to the hyperplane, and while we would predict +1, it seems likely that just a small change to the separating plane could easily cause our prediction to be -1. Hence, we are much more confident about our prediction at A than at C.

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 separating hyperplane).

The next figure shows two different hyperplanes that both can classify the training samples correctly:



(a)



(b)

Based on the intuitive analysis, certainly the hyperplane in (b) is preferred because the samples are farther from separating hyperplane in (b) than in (a).

Mathematically, 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 the bias.

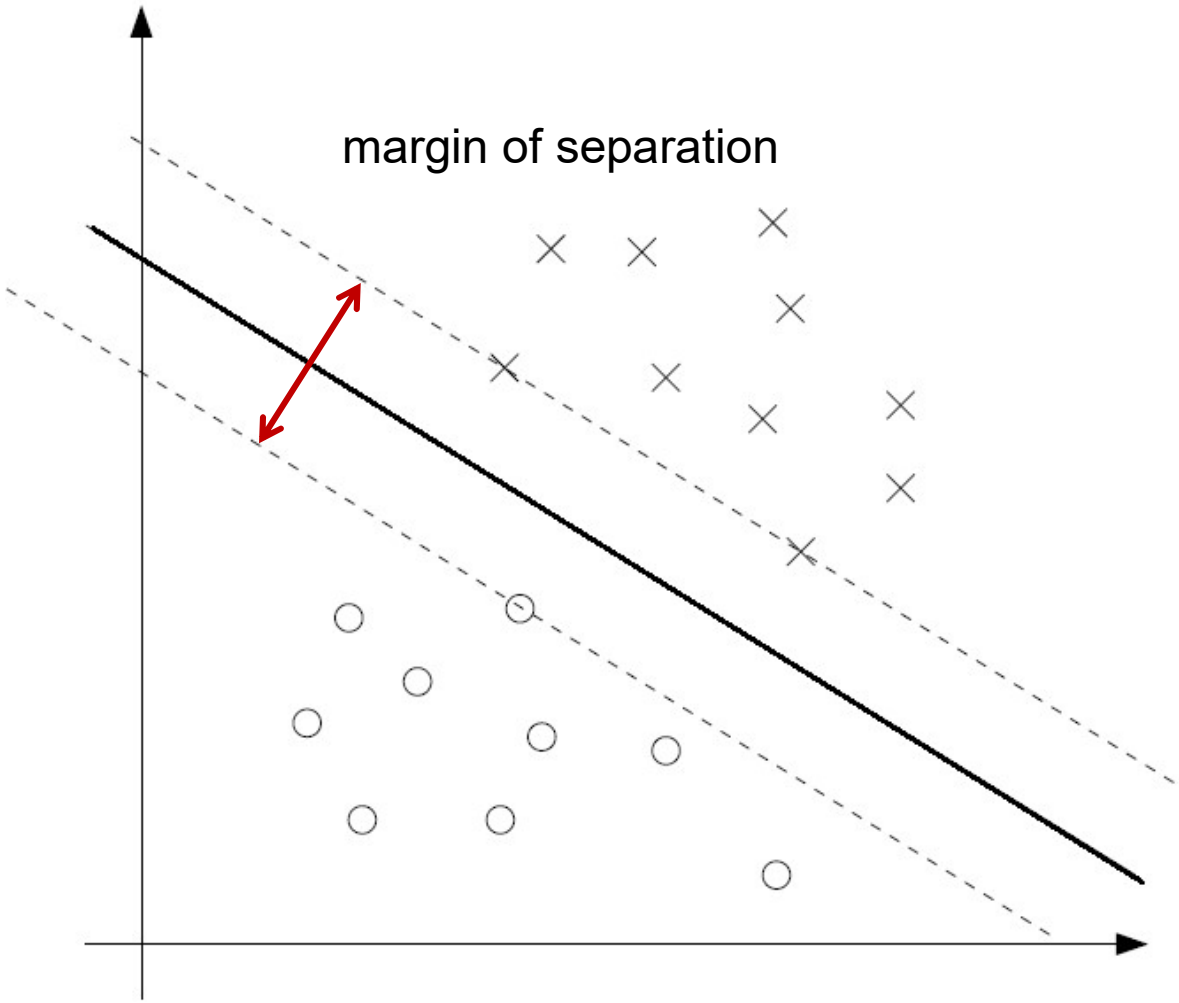
(a) For $\mathrm{d} = +1$

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

(b) For $\mathrm{d} = -1$

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

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 (SVM) is to find the particular hyperplane for which the margin of separation is maximized. Under this condition, the decision surface is referred to as the **optimal hyperplane**.

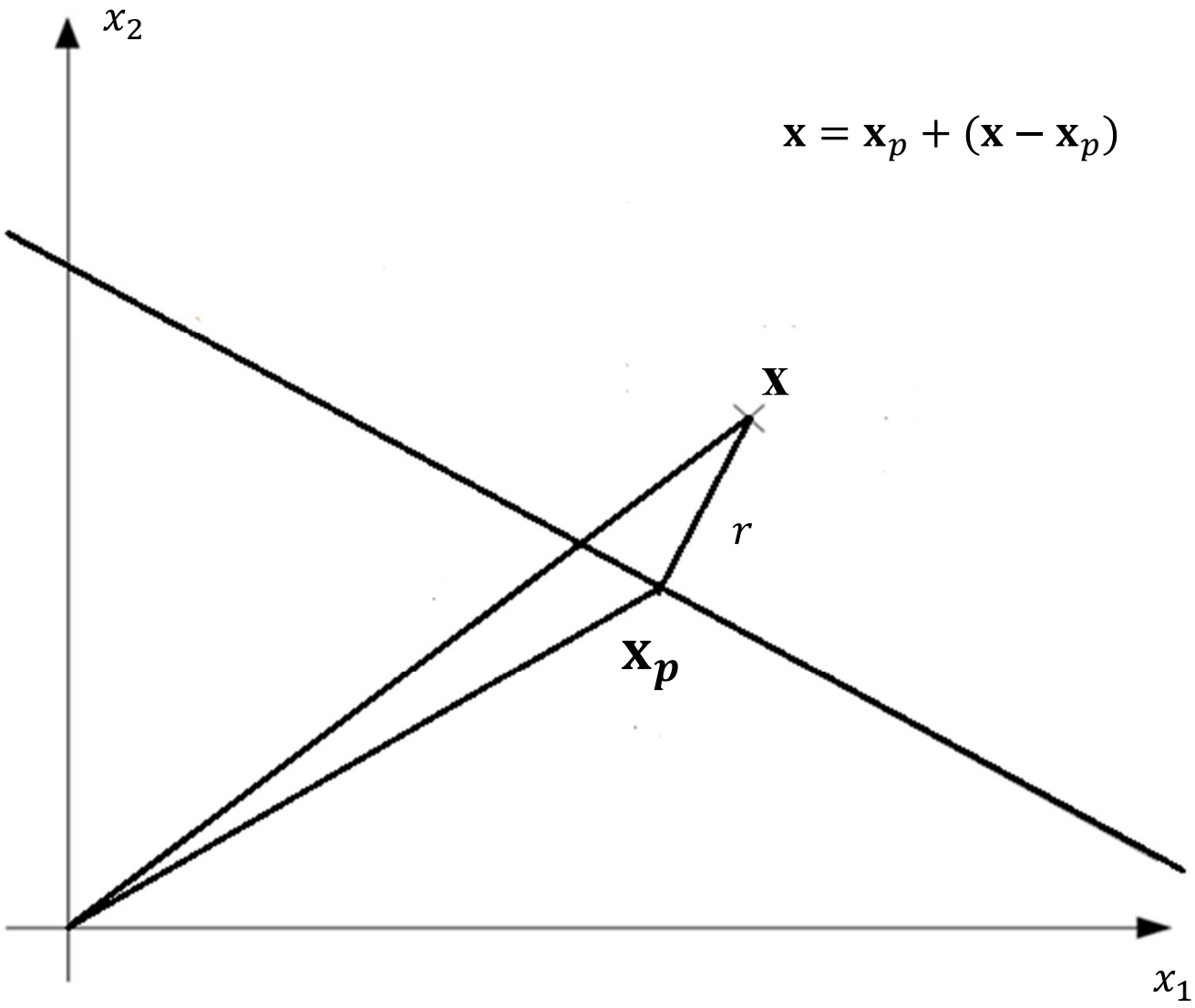
The discriminant function

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

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 hyperplane, and r is the distance from $\mathbf{x}$ to the hyperplane as illustrated below.


x_2
$\mathbf{x} = \mathbf{x}_p + (\mathbf{x} - \mathbf{x}_p)$
$\mathbf{x}$
r
$\mathbf{x}_p$
x_1

r is positive if $\mathbf{x}$ is on the positive side of the optimal hyperplane, and is negative if $\mathbf{x}$ is on the negative side.

$$
\begin{aligned}
g(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\
&= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b \\
&= \mathbf{w}^T\mathbf{x}_p + b + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} \\
&= g(\mathbf{x}_p) + r\|\mathbf{w}\|
\end{aligned}
$$

Since $\mathbf{x}_p$ is on the separating hyperplane, we have:

$$g(\mathbf{x}_p) = 0$$

Then we have:

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

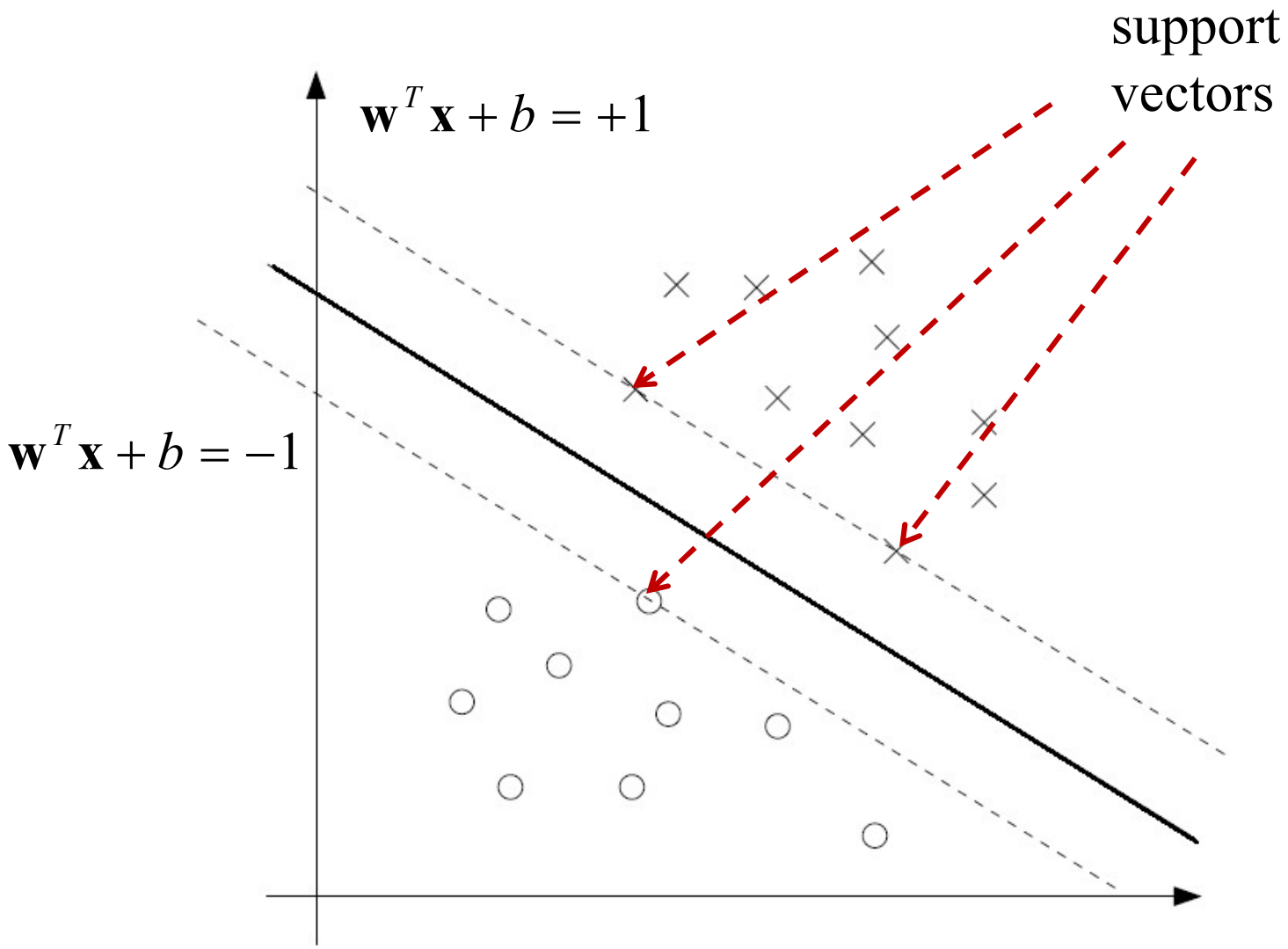
If the data are linearly separable, we can always scale **w** and b so that

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

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

The particular data points for which the above is satisfied with equality sign are called **support vectors**. The next figure shows the position of the support vectors.

The support 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.


support
vectors
$\mathbf{w}^T\mathbf{x} + b = +1$
$\mathbf{w}^T\mathbf{x} + b = -1$

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:

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

Maximizing the margin of separation ρ is equivalent to minimizing the Euclidean norm of the weight vector **w**.

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

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

Thus, the constrained optimization problem that we have to solve may 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 (please see the Appendix for the Lagrange multipliers method):

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

where

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

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 $J(\mathbf{w}, b, \alpha)$, which has to be minimized with respect to **w** and b.

Differentiating $J(\mathbf{w}, b, \alpha)$ with respect to **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:
$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0$$

Since
$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = -\sum_{i=1}^{N} \alpha(i) d(i)$$

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

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

From Conditions 1 and 2, we obtain :

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

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

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

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

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

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

We 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 condition 2. Furthermore, by the optimality condition 1, we have:

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

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:

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

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

The dual (also primal) problem is a quadratic programming (QP) problem, we can use available QP software to solve it to get α(i).

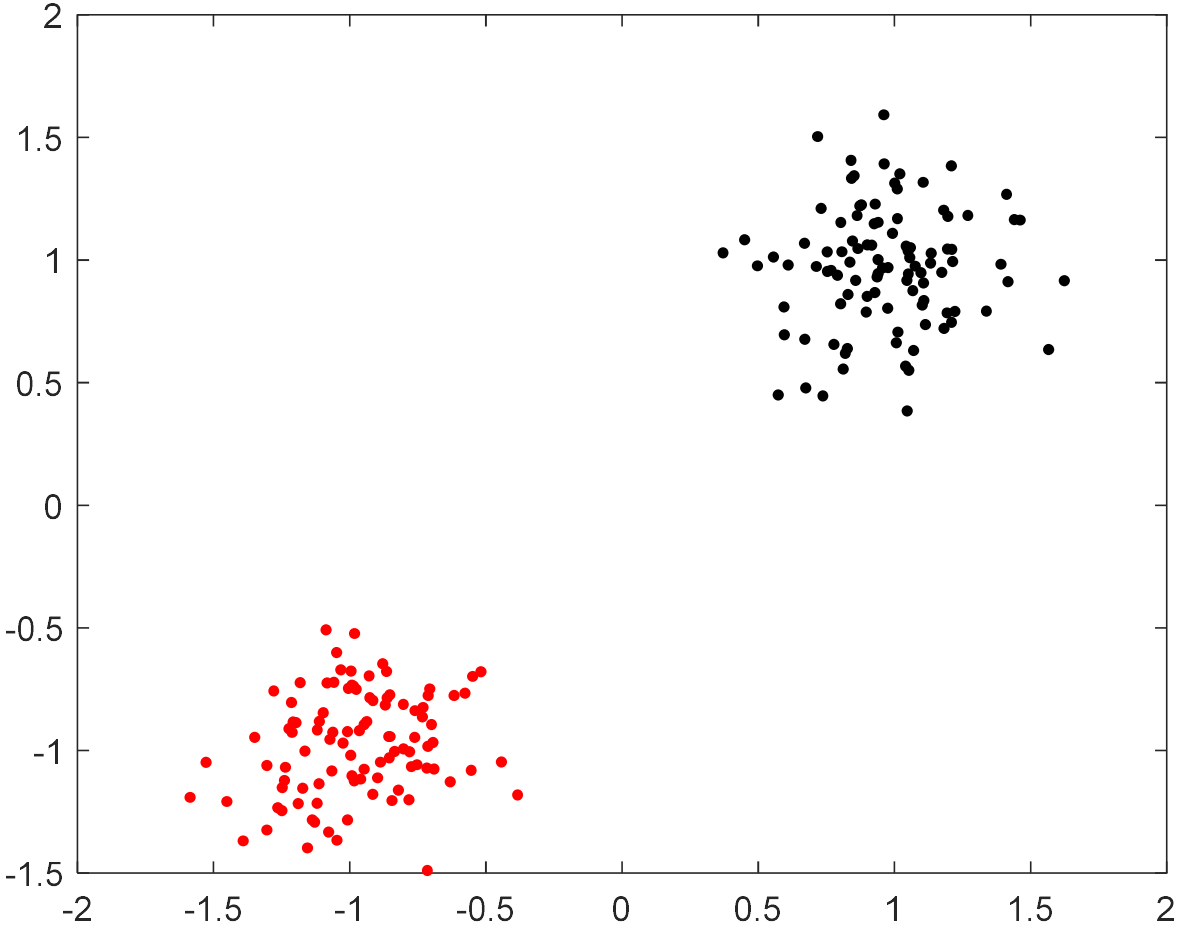
After having determined the optimum Lagrange multipliers α(i), we may compute the optimal weight vector **w** and b:

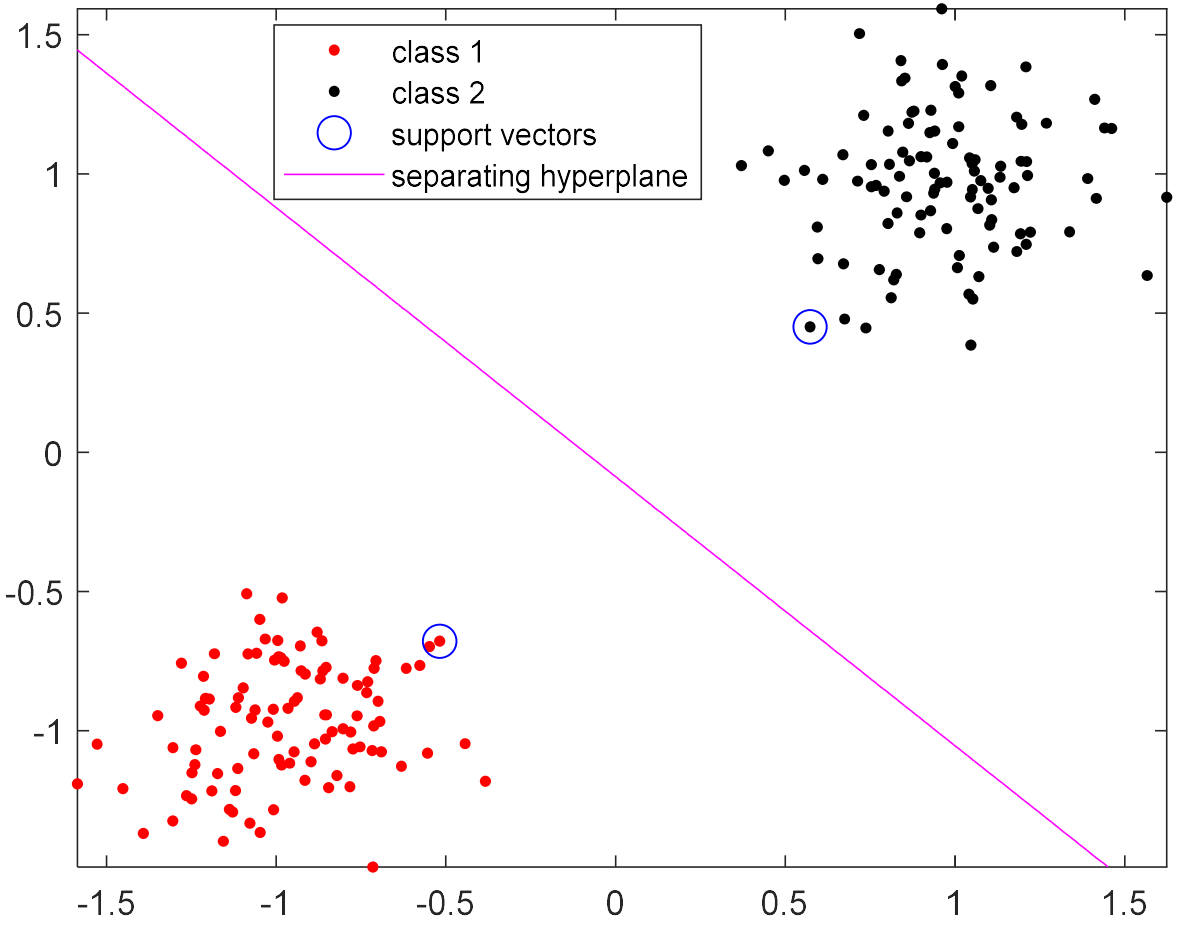
$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha(i) d(i) \mathbf{x}(i)$$

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

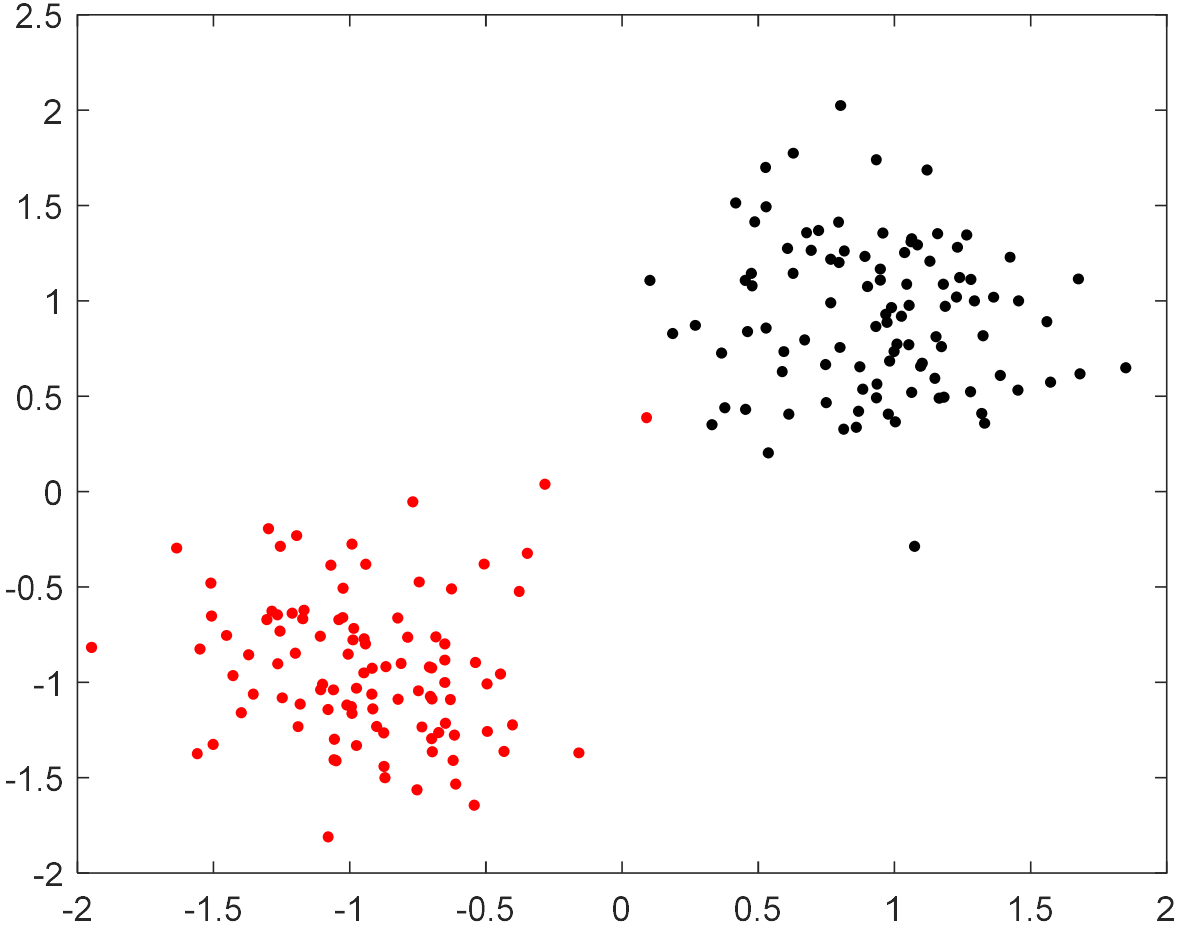
Actually, most of the α(i) are zeros. The samples **x**(i) corresponding to nonzero α(i) are the support vectors. In other words, the optimal hyperplane is determined by the support vectors only.

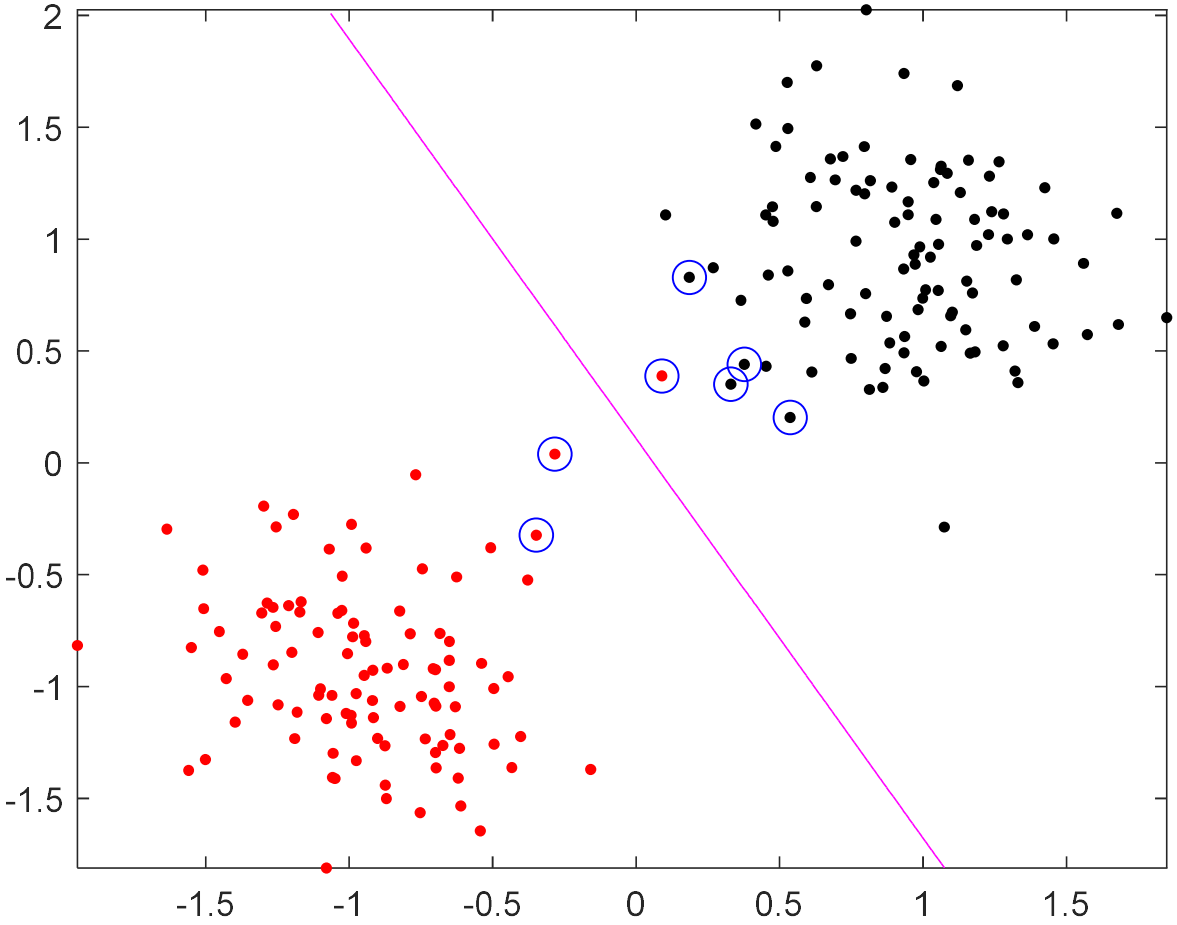
Example 1 of linearly separable samples




class 1
class 2
support vectors
separating hyperplane
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.5
0
0.5
1
1.5

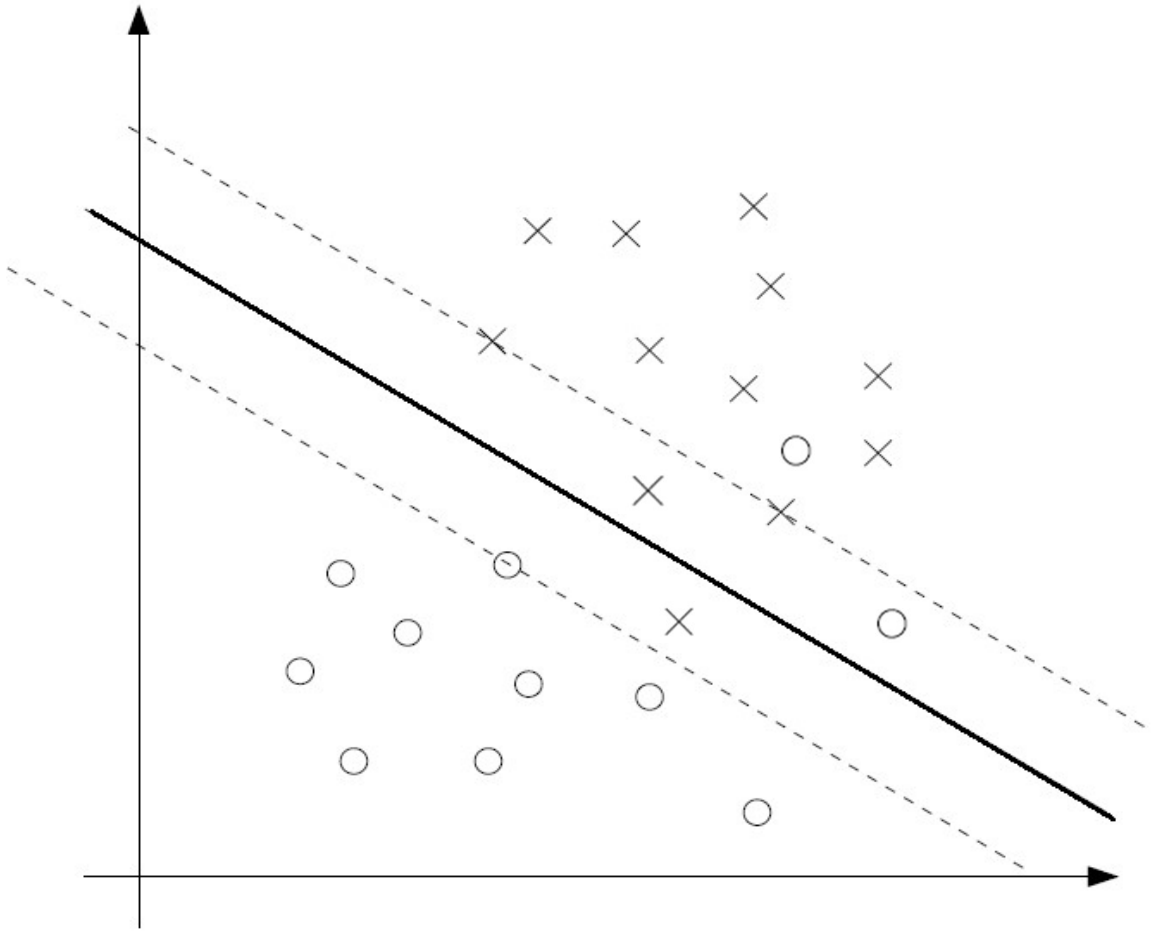
Example 2 of linearly separable




2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5

6.3 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 as shown below.



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.

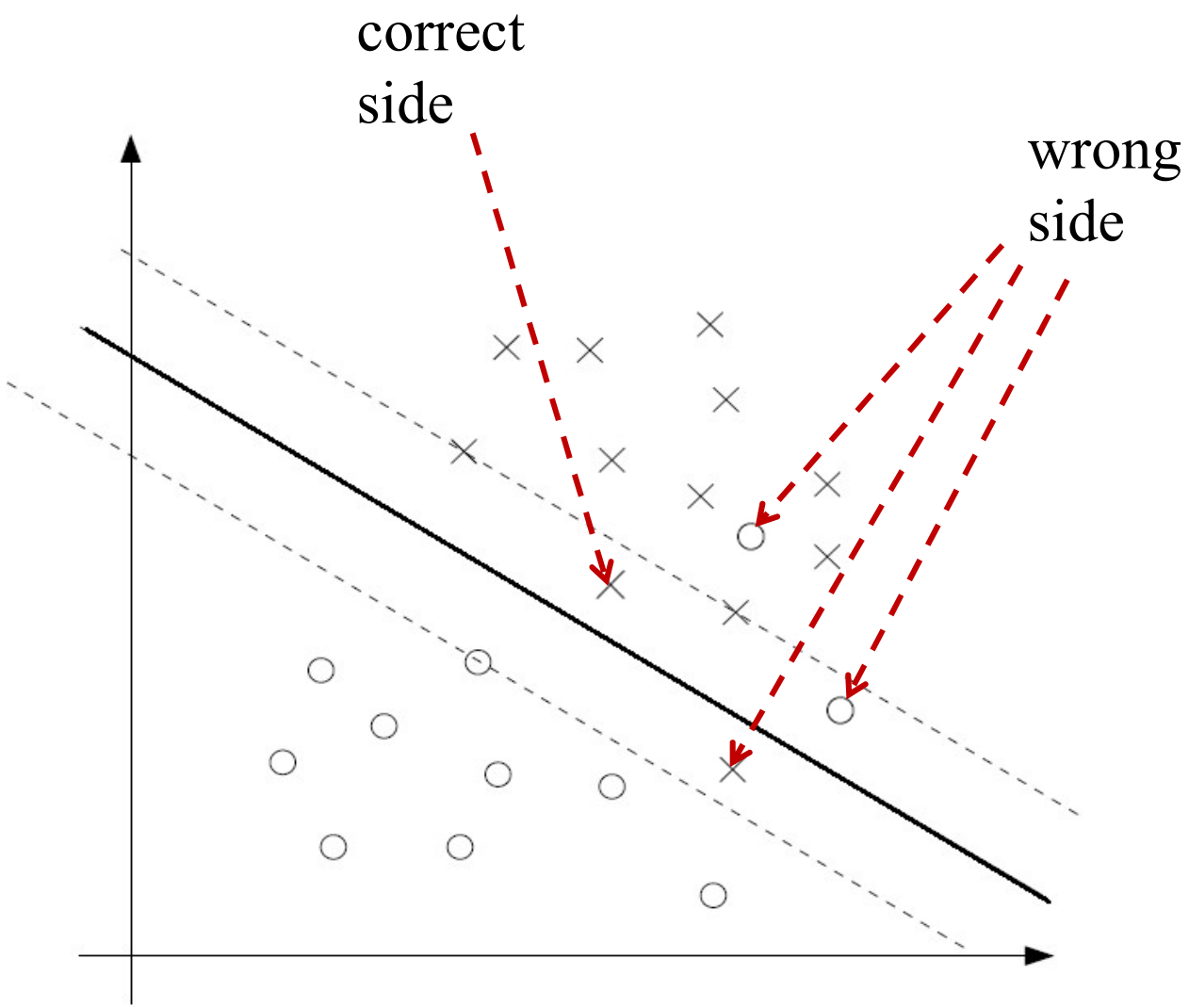
The margin of separation between classes is said to be soft if a data point {**x**(i),d(i)}, i=1,2,…N, violates the following conditions:

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

The violation can arise in the following two ways:

(a) The data point **x**(i) falls inside the region of separation but on the correct side of the hyperplane;

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


correct
side
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:

$$d(i) \times [\mathbf{w}^T \mathbf{x}(i) + b] \geq 1 - \xi(i)$$

where ζ(i) are called slack variables, which measure the deviation of a data point from the ideal condition of pattern separability.

(a) For $0 \leq \zeta(i) \leq 1$, the data point falls inside the region of separation but on the correct side of the hyperplane.

(b) For $\zeta(i) > 1$, 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 use of a validation set.

By 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:

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

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

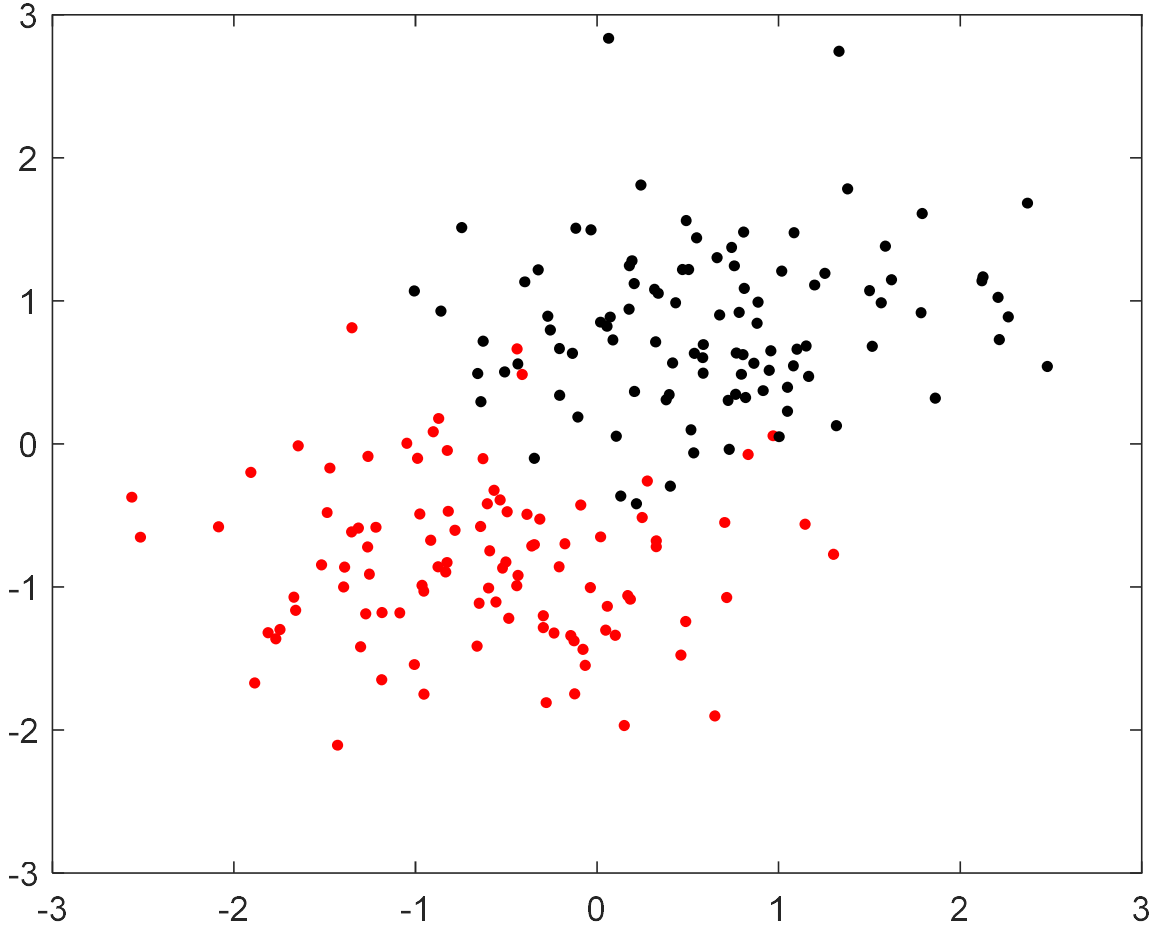
Again, we can solve the QP problem to get α(i). Then the optimal weight vector **w** and b are obtained as :

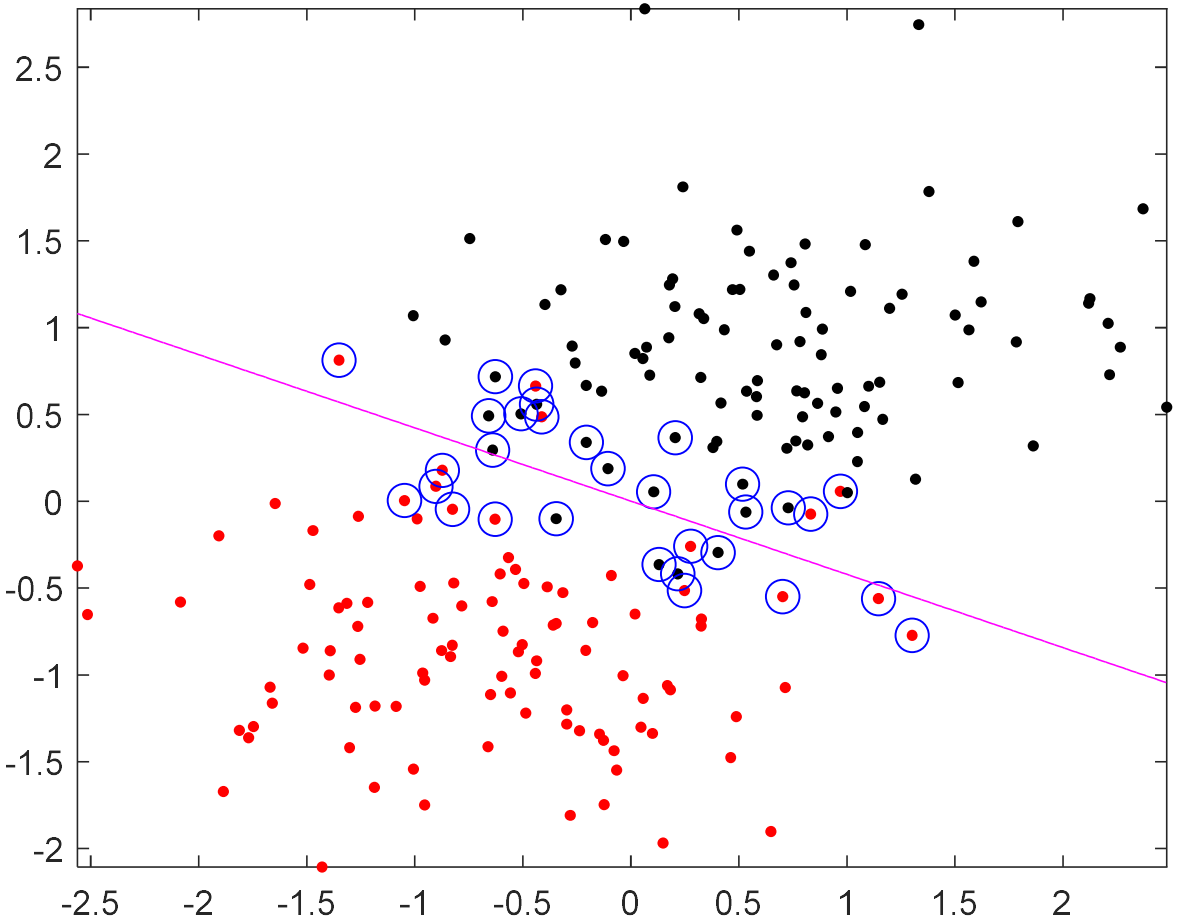
The optimal weight vector **w** and b are given by:

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

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

Example of linearly nonseparable




2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

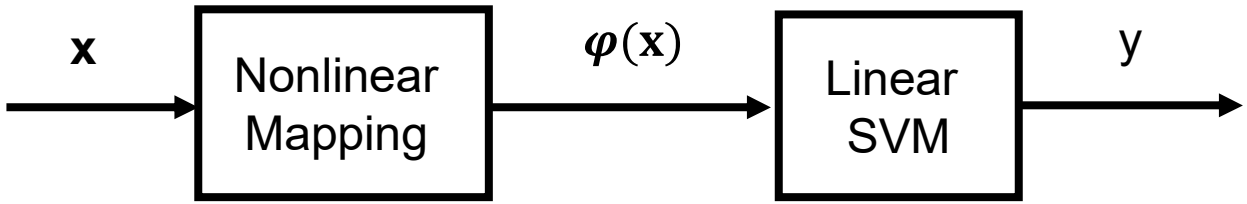
6.4 Kernel Support Vector Machines

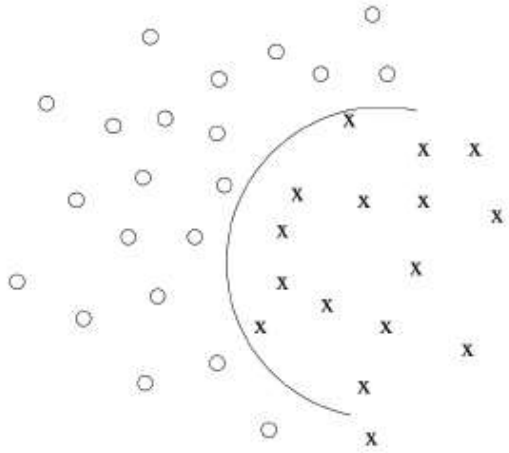
In the above, linear SVM is presented. If the data is linearly inseparable, how to construct a classifier to classify the data?

This can be solved as follows:

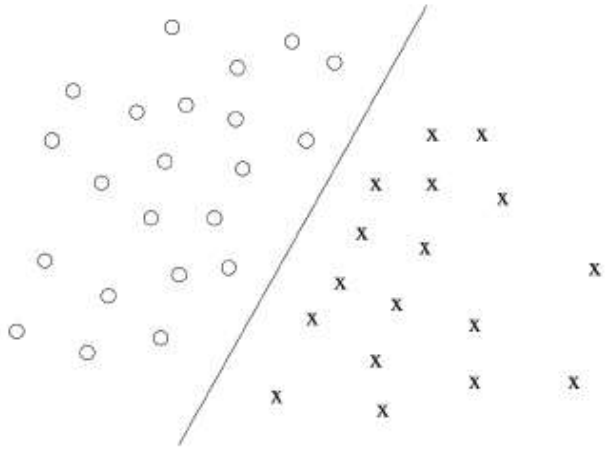
Step 1: Nonlinear mapping of an input vector into a high-dimensional feature space;

Step 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 in the feature space rather than the original input space.

Let $\varphi_j(\mathbf{x})$, $j=1,2,...,m$, 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 , $j=1,2,...,m$, 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:

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

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

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

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

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:

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

Let's see an example.

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

which can also be written as:

$$K[\mathbf{x},\mathbf{z}] = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right)$$

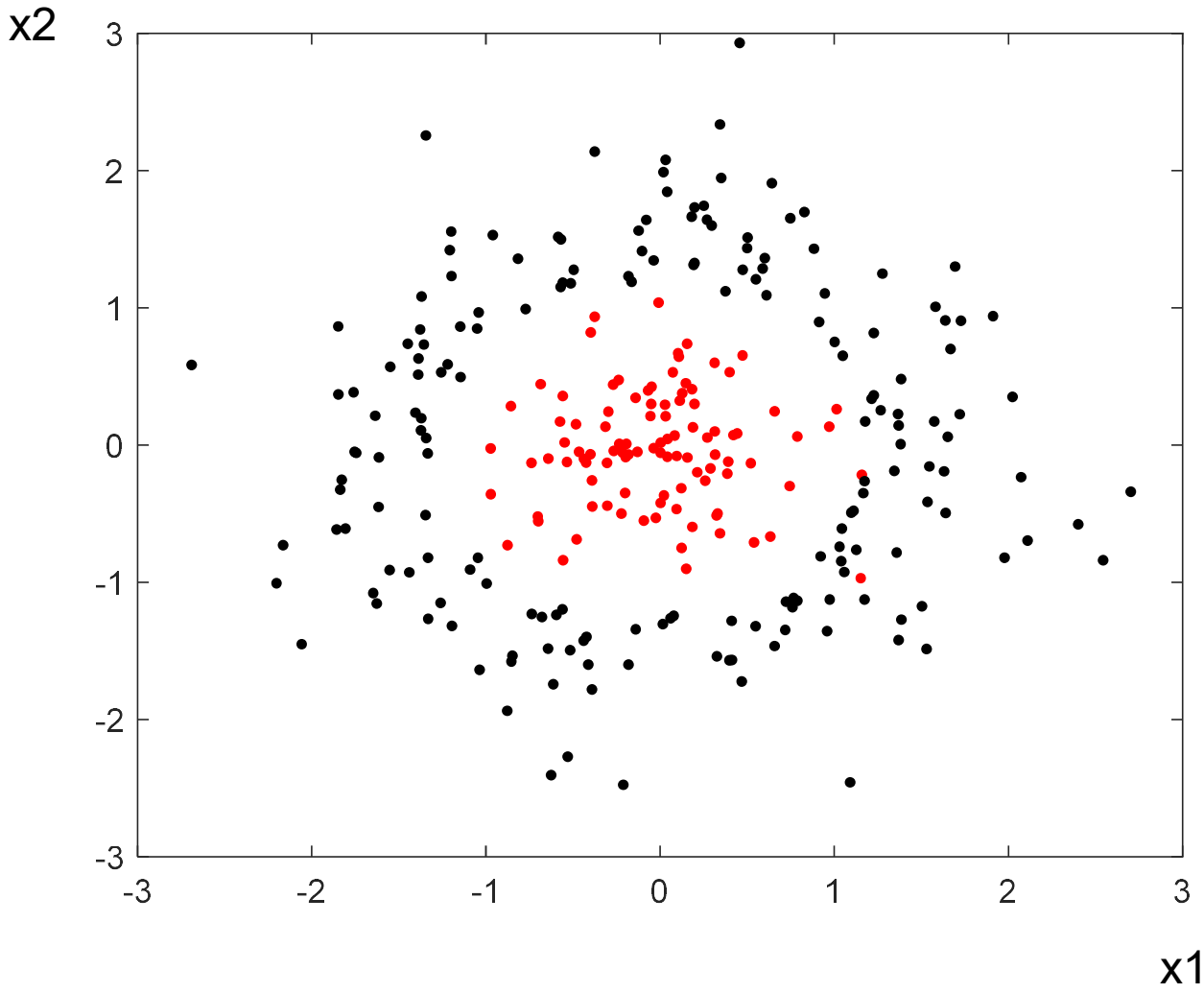
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i z_i x_j z_j$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} (x_i x_j)(z_i z_j)$$

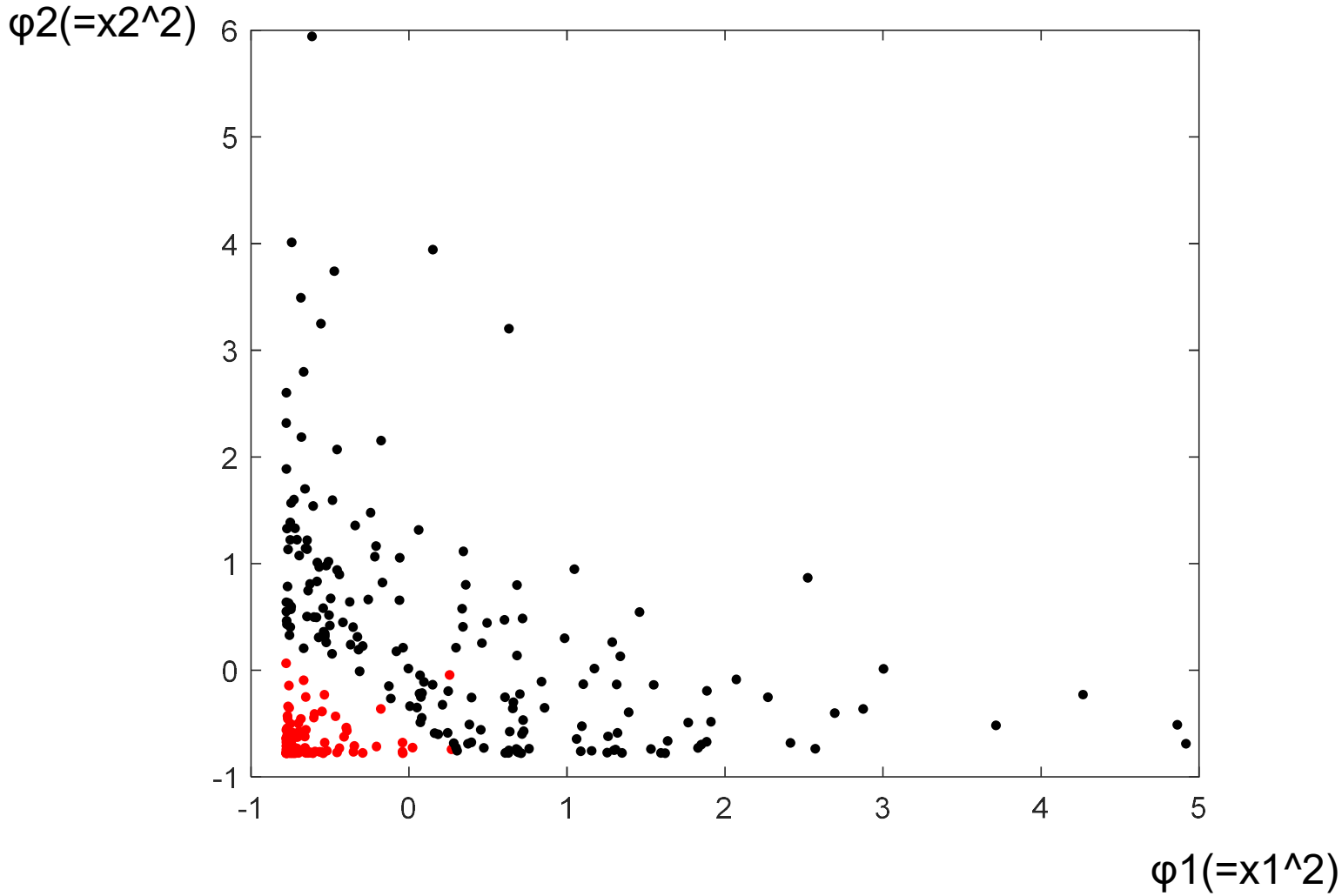
If $\varphi(\mathbf{x}) = \begin{bmatrix} x_1^2 & x_1x_2 & x_2x_1 & x_2^2 \end{bmatrix}^T$

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

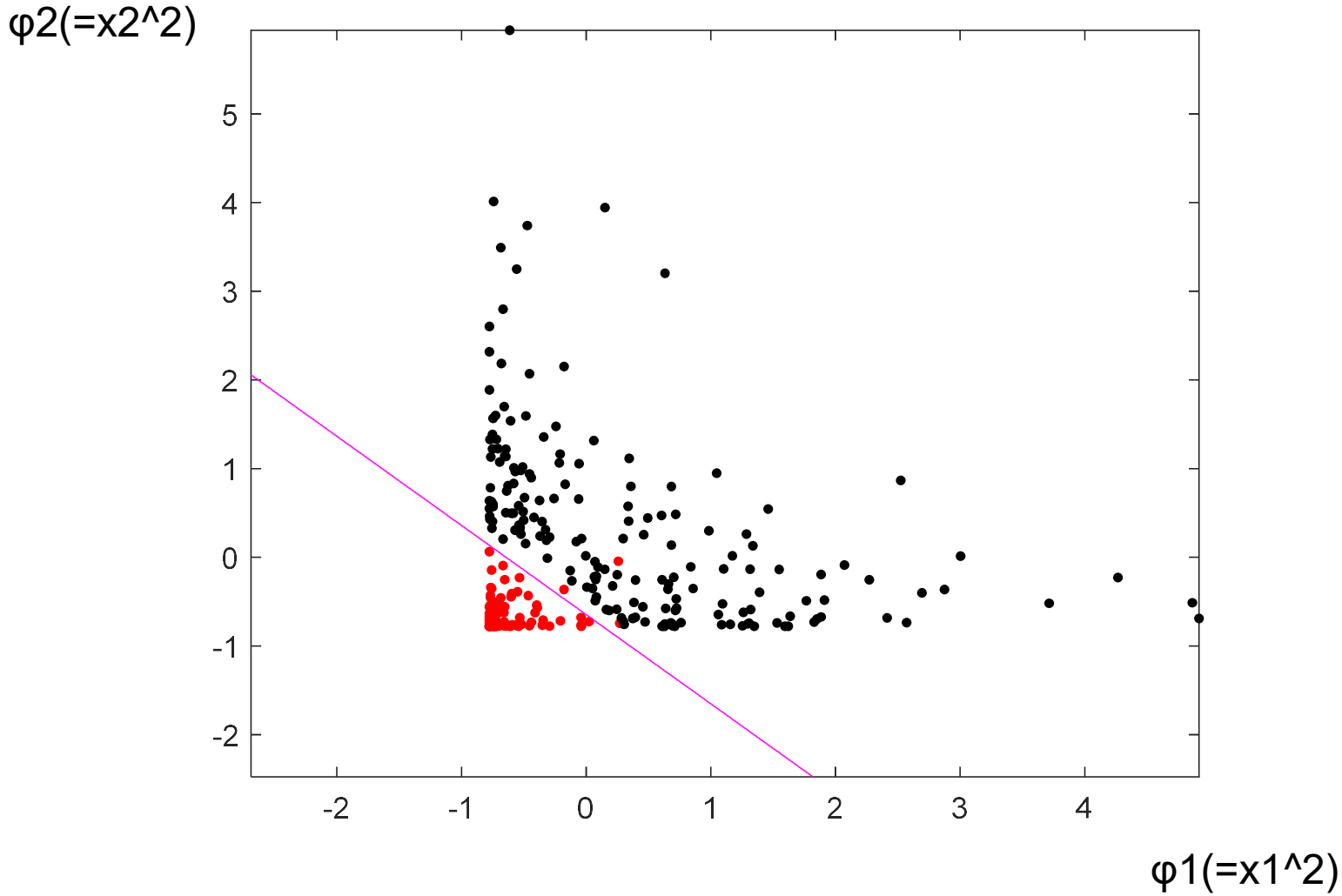
Samples in input space



Samples in feature space (φ space)



Linearly separable in feature space (φ space)



Now we introduce kernel matrix **K**, whose element at row i and column j is defined as**:**

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

Obviously, the kernel matrix **K** is symmetric. In addition, for any vector **z**, we have:

$$\begin{aligned}
z^T \mathbf{K} z &= \sum_i \sum_j z_i k(i, j) z_j \\
&= \sum_i \sum_j z_i \boldsymbol{\varphi}[\mathbf{x}(i)]^T \boldsymbol{\varphi}[\mathbf{x}(j)] z_j \\
&= \sum_i \sum_j z_i \sum_k \varphi_k[\mathbf{x}(i)] \varphi_k[\mathbf{x}(j)] z_j \\
&= \sum_k \sum_i \sum_j z_i \varphi_k[\mathbf{x}(i)] \varphi_k[\mathbf{x}(j)] z_j \\
&= \sum_k \left(\sum_i z_i \varphi_k[\mathbf{x}(i)] \right)^2 \geq 0
\end{aligned}$$

The above result means the kernel matrix **K** is positive semi-definite. To judge whether a *K* is a valid kernel, we have the following Mercer's Theorem:

Mercer's Theorem

*K is a valid kernel, it is necessary and sufficient that for any samples x(1),x(2),…,x(N), the corresponding kernel matrix **K** is symmetric and positive semi-definite.*

For the example kernel discussed above:

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

$\boldsymbol{\varphi}(\mathbf{x})$ has a computation complexity of $O(n^2)$, while K has a complexity of $O(n)$, which is linear in the dimension of the input space. The computational complexity reduction by using kernel is significant if the input space is of very high dimensional.

Kernel Functions in SVM

The requirement on kernel $K(\mathbf{x}, \mathbf{z})$ 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

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^p$$

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

(ii) Gaussian kernel

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

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

Design of a kernel support vector machine

The expansion of the inner-product kernel permits us to construct a decision surface that is nonlinear in the input space, but is linear in the feature space. We may 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 $K[\mathbf{x}(i),\mathbf{x}(j)]$.

After having found the optimum values of the Lagrange multipliers, denoted by $\alpha(i)$, we may compute the decision value of a given input vector $\mathbf{x}$:

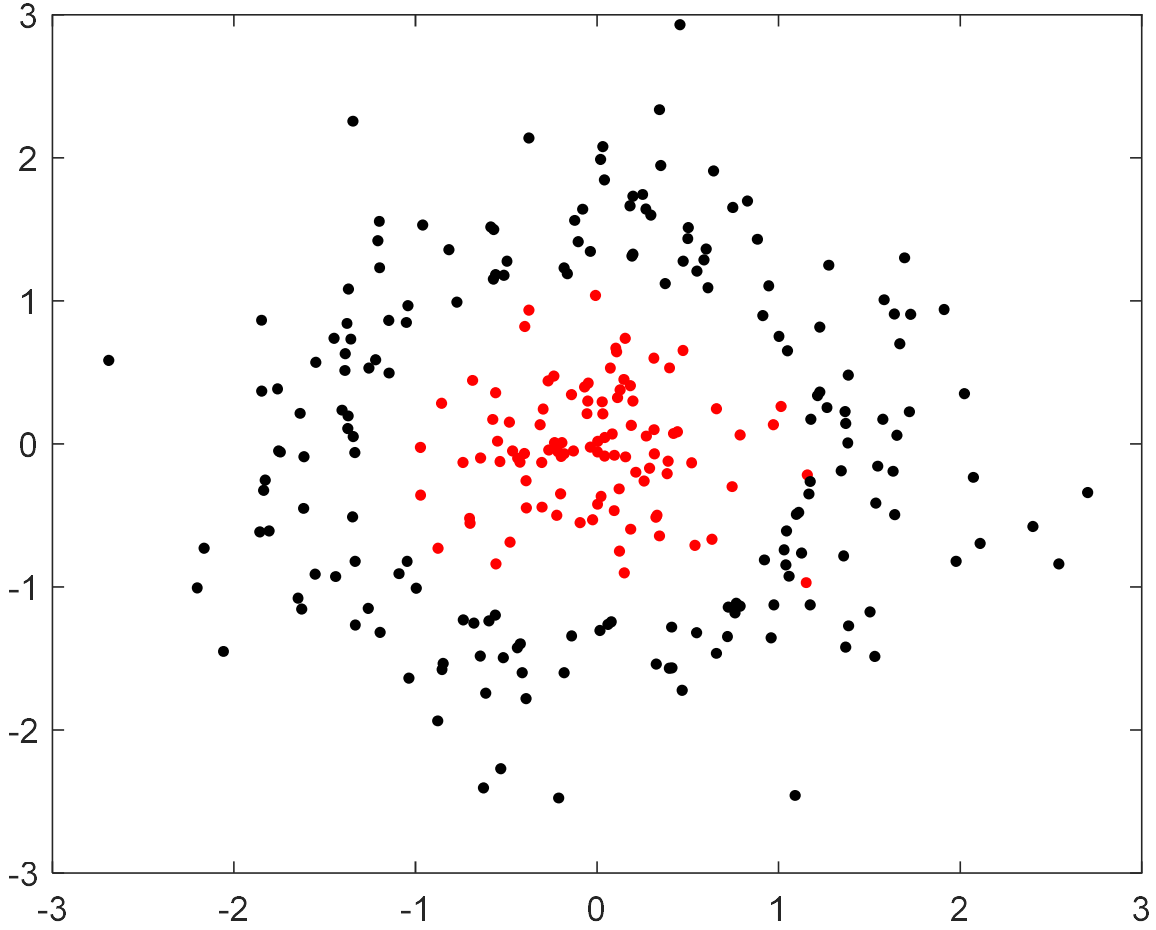
$$y = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x})$$

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

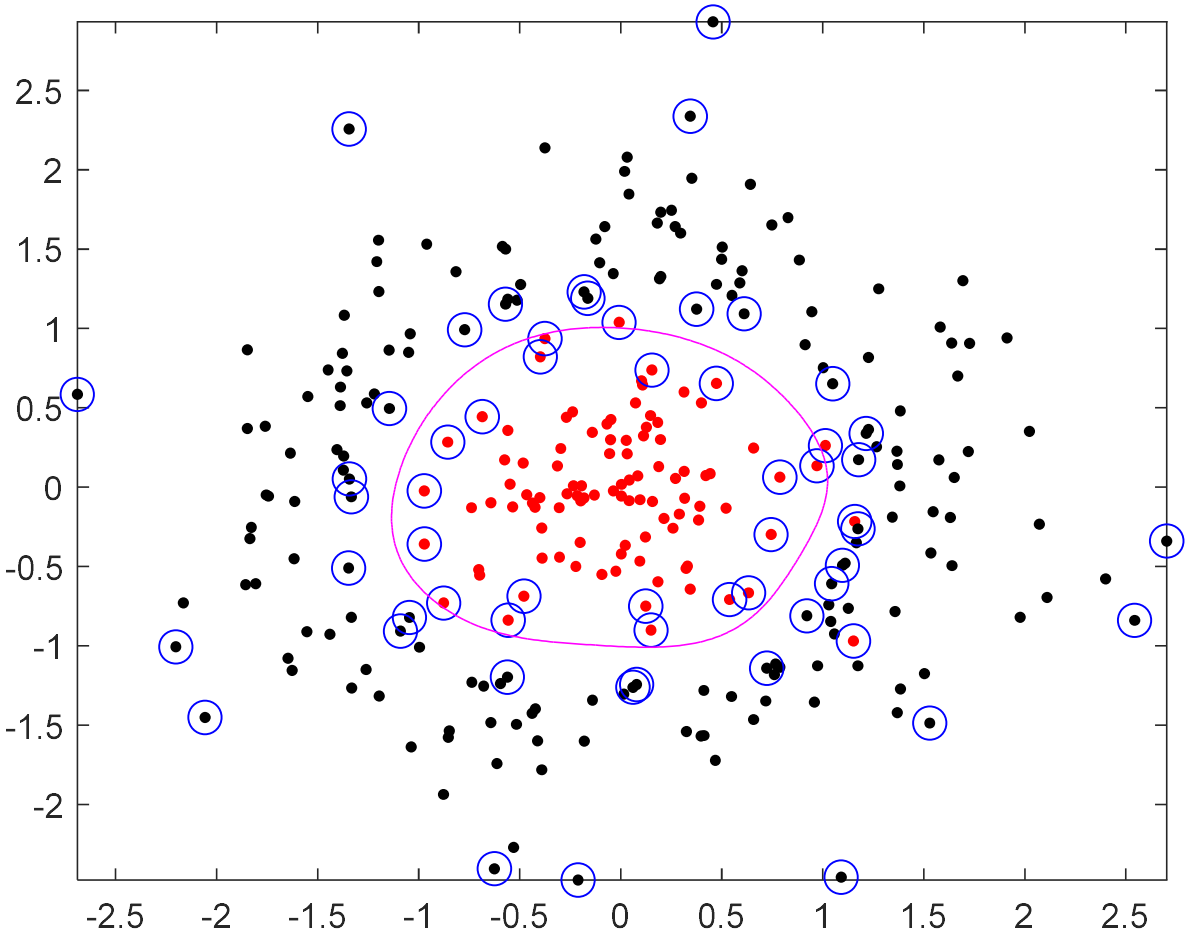
We do not compute the weight vector $\mathbf{w}$ explicitly because the feature space is hidden, and $\boldsymbol{\varphi}[\mathbf{x}(i)]$ is unavailable:

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

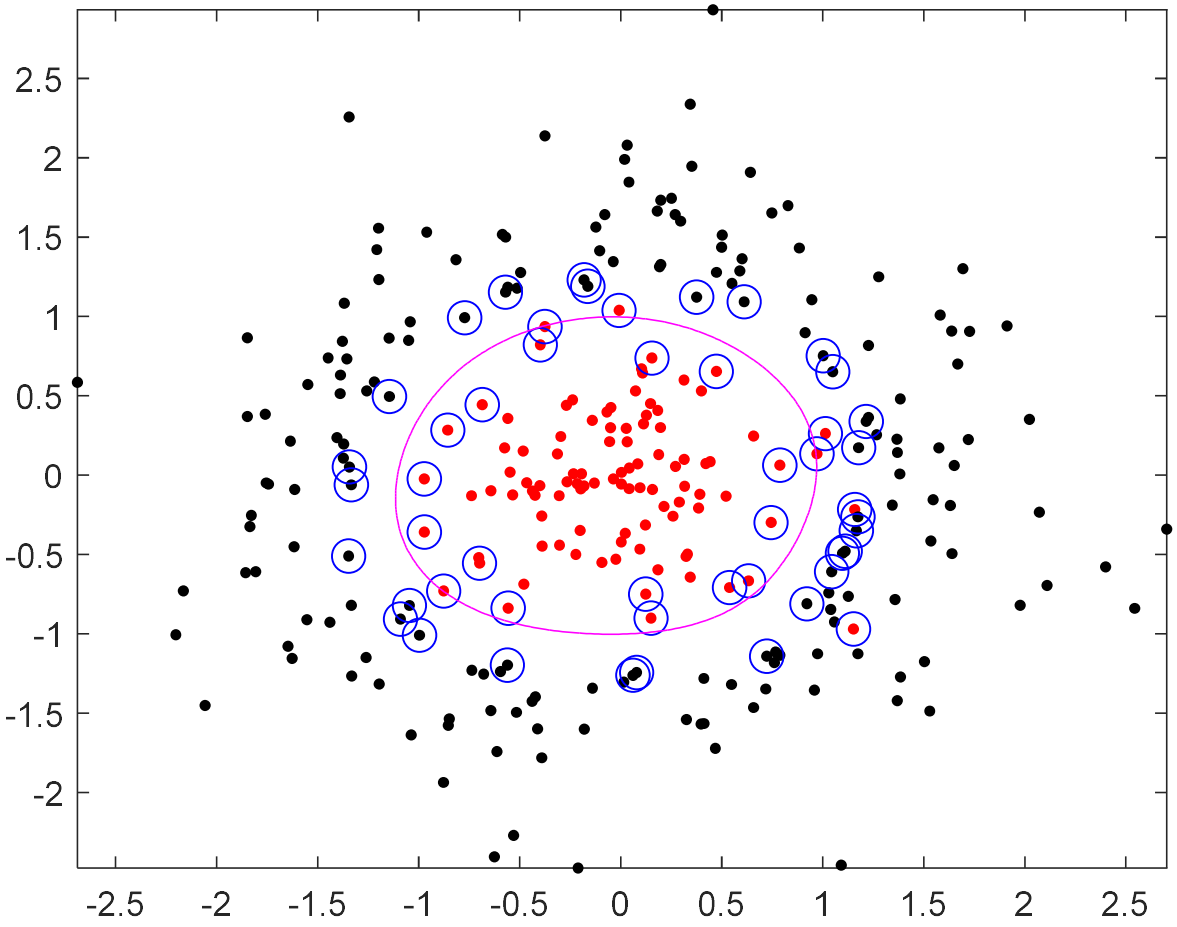
Example 1 of kernel SVM



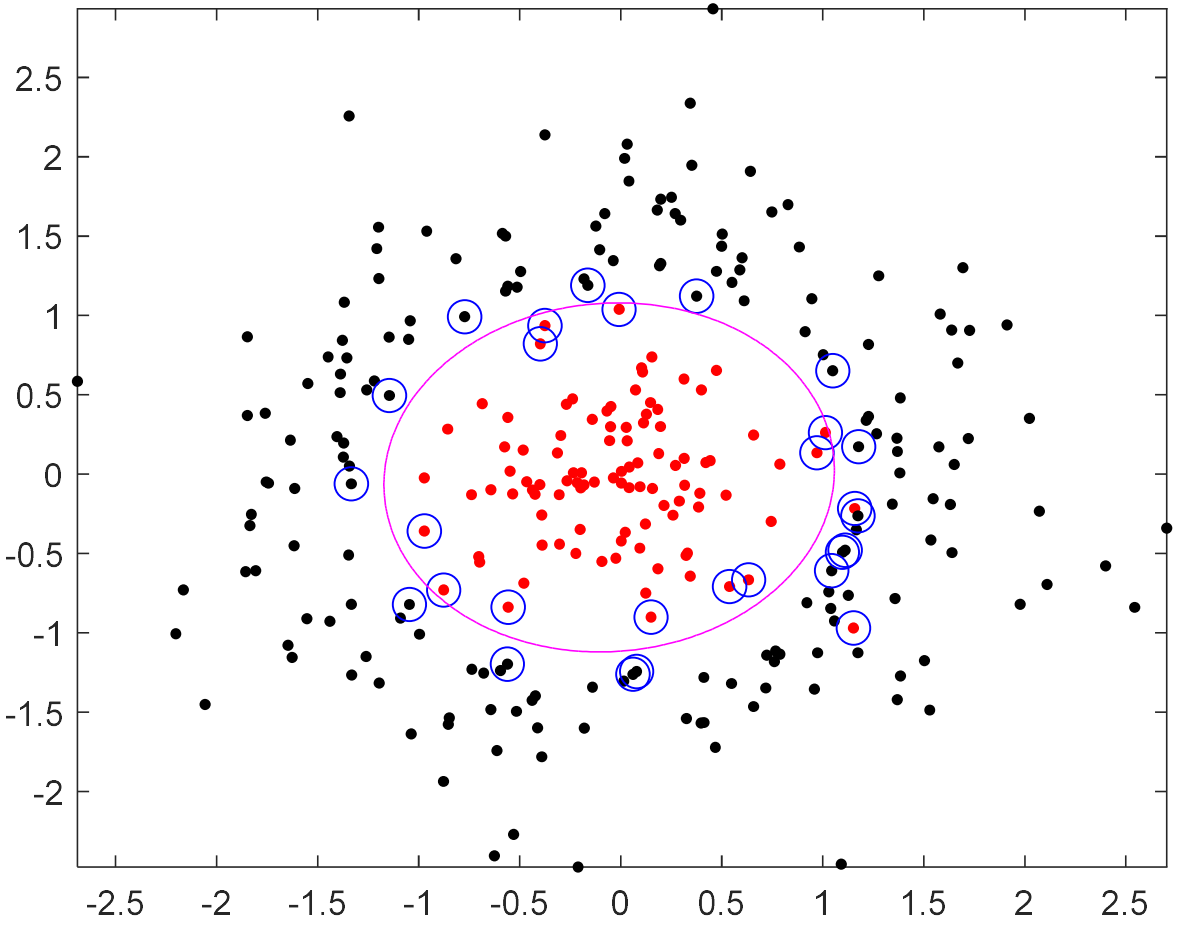
Example 1 of kernel SVM (σ=1)



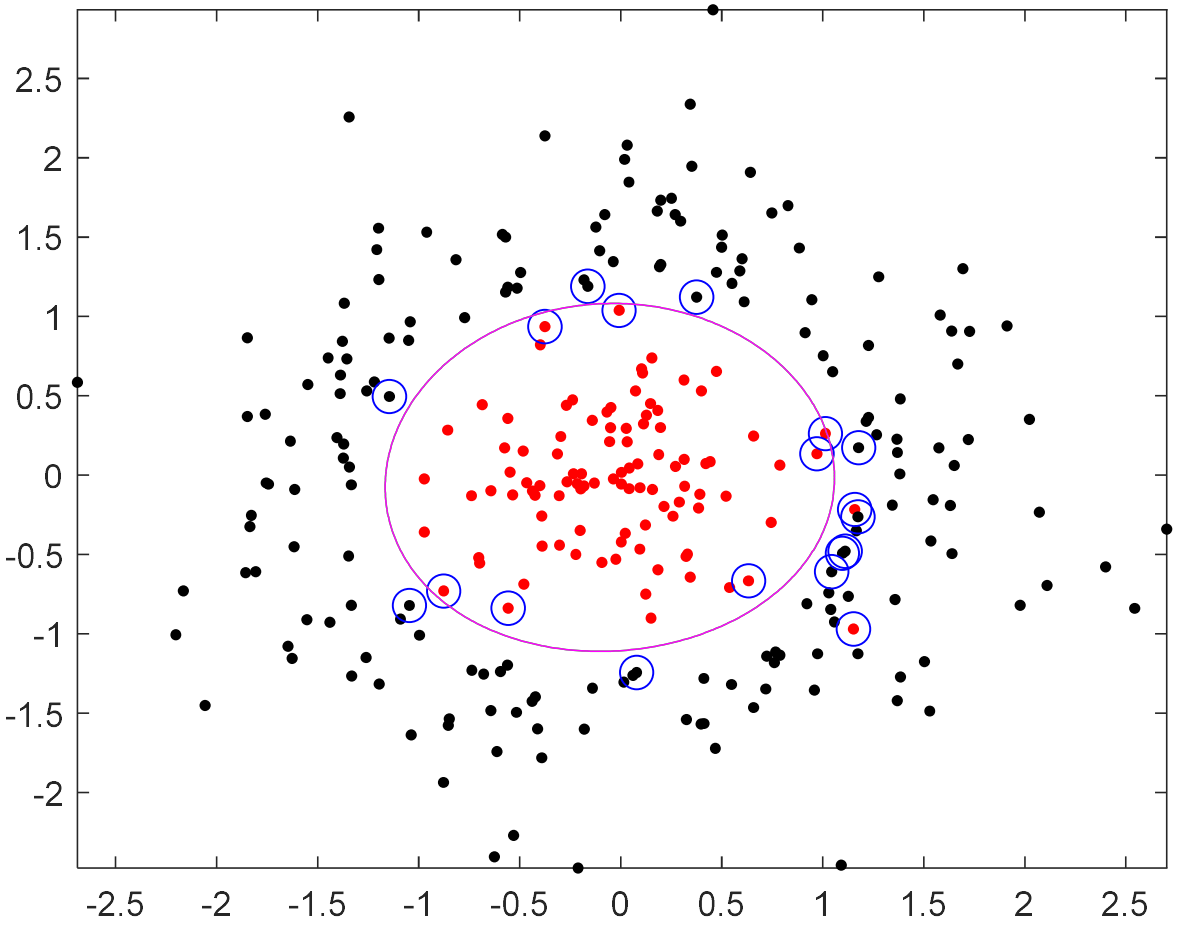
Gaussian kernel (σ=0.6767)



Polynomial kernel (p=2)



Polynomial kernel (p=3)



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.

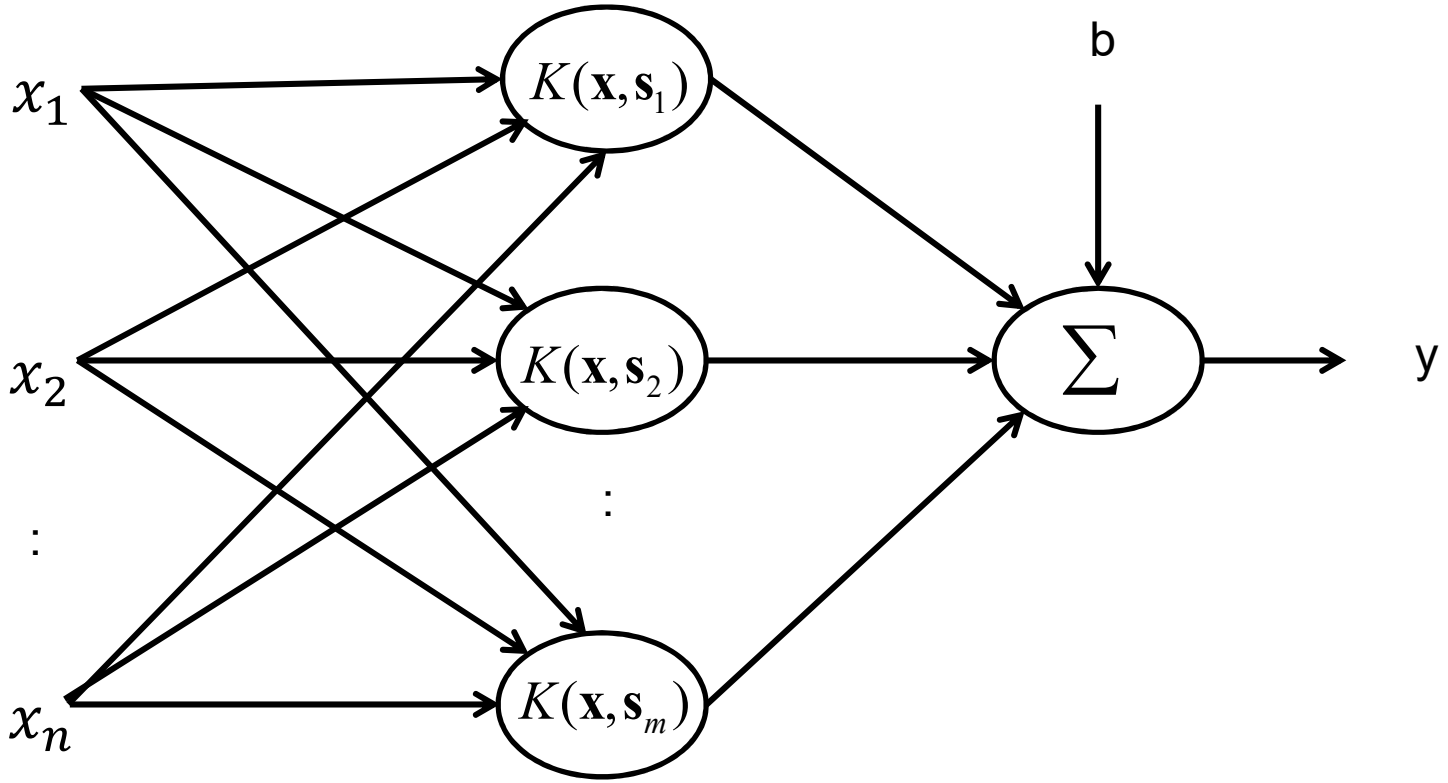
(iv) In Gaussian kernel SVM, the number of the neurons and their centre vectors are determined automatically by the number of support vectors and their values.

The support vector machine differs from the conventional approach 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



$\mathbf{s}_i$ are the support vectors

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:

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

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:

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

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

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

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