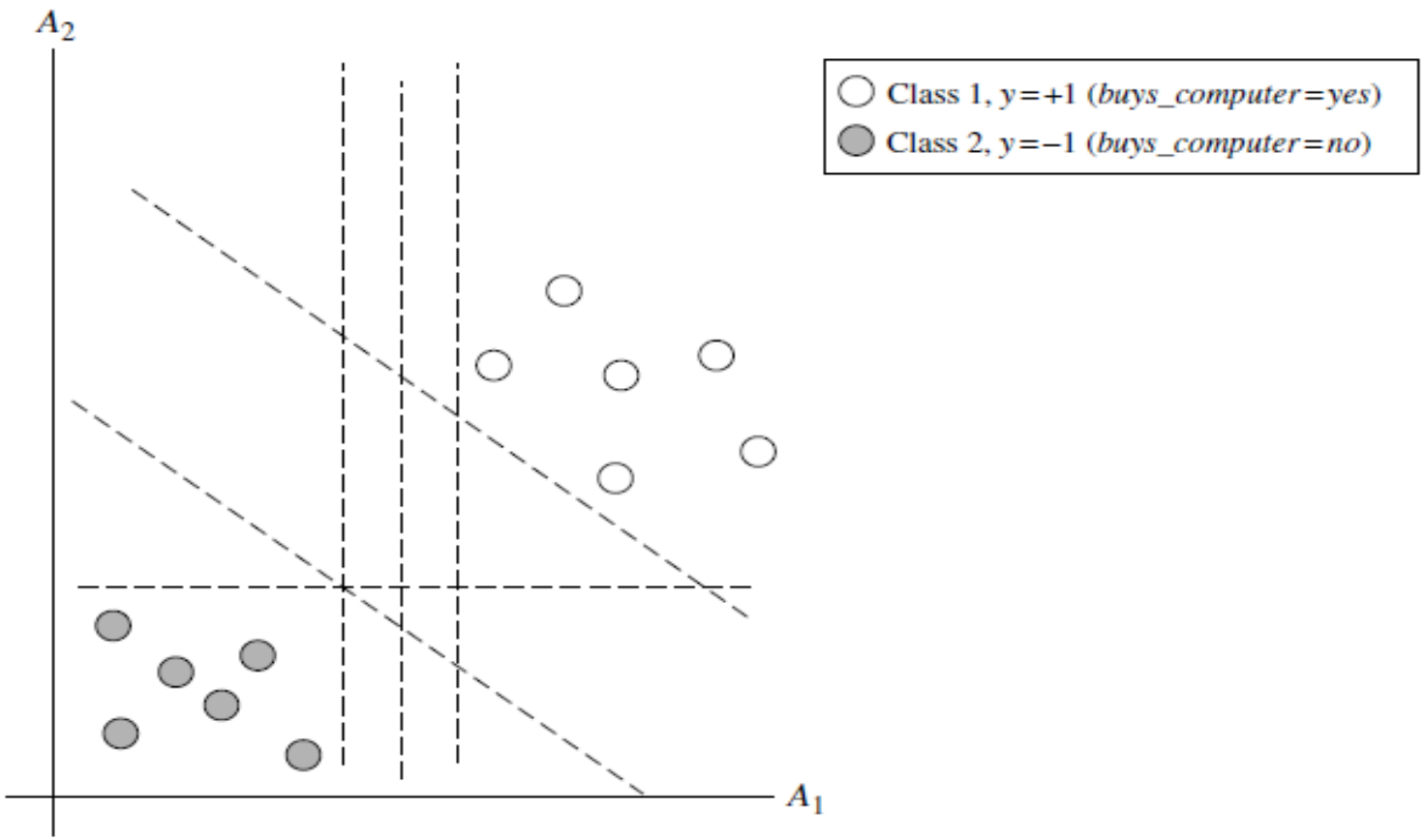
Support Vector Machine

SVM

- It is an algorithm that uses a nonlinear mapping to transform the original training data into a higher dimension.
- Within this new dimension, it searches for the linear optimal separating hyperplane (a "decision boundary" separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane.
- The SVM finds this hyperplane using *support vectors* ("essential" training tuples) and *margins* (defined by the support vectors).

SVM—History and Applications

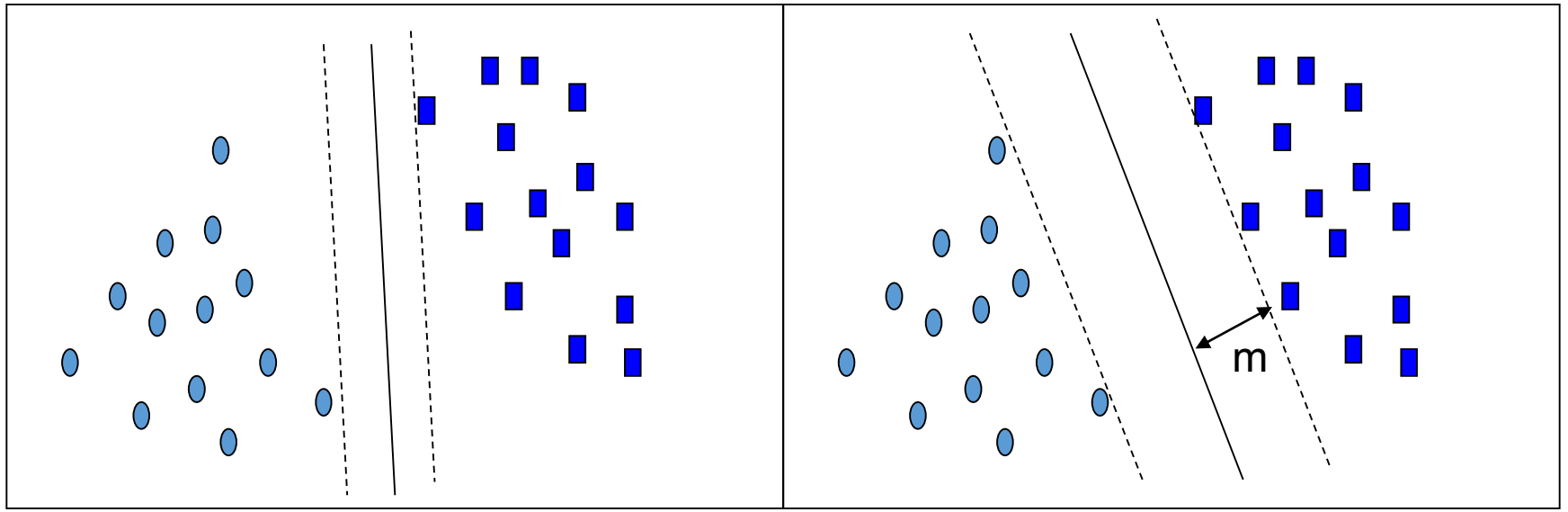
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests


A_2
A_1
Class 1, $y=+1$ (*buys_computer=yes*)
Class 2, $y=-1$ (*buys_computer=no*)

Linear SVM

- Consider the case where data are linearly separable.
- Let the data set D be given as (X_1,y_1), (X_2,y_2)……..$(X_{|D|}, y_{|D|})$, where X_i is the set of training tuples with associated class labels, y_i. Each y_i can take one of two values, either +1 or -1.
- From the graph in previous slide we can say that 2D data are linearly separable because a straight line can be drawn to separate all tuples of class +1 from all the tuples of class -1.
- There are an infinite number of separating lines that could be drawn. But we want to find the **"best"** one, that is, one that will have the minimum classification error on previously unseen tuples.
- How can we find this best line??

SVM—When Data Is Linearly Separable



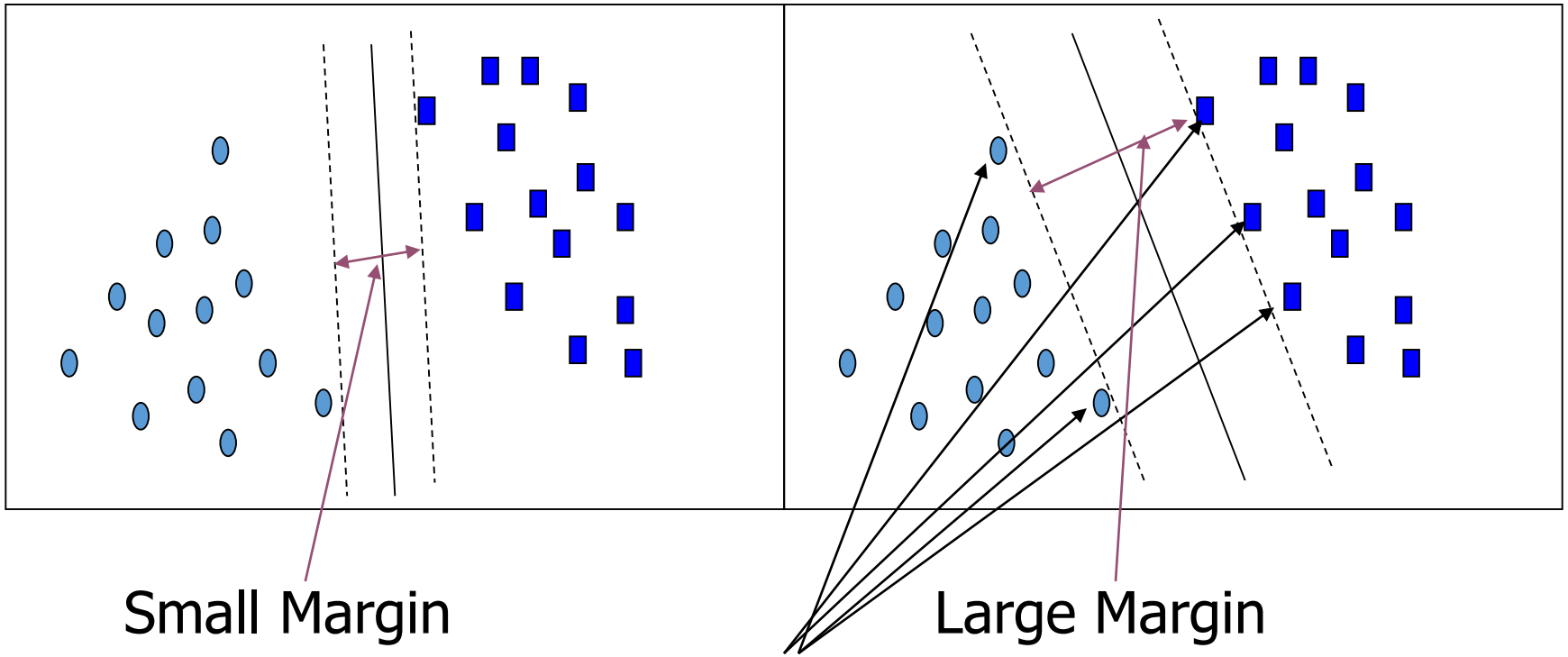
Let data D be $(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_{|D|}, y_{|D|})$, where $\mathbf{X}_i$ is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

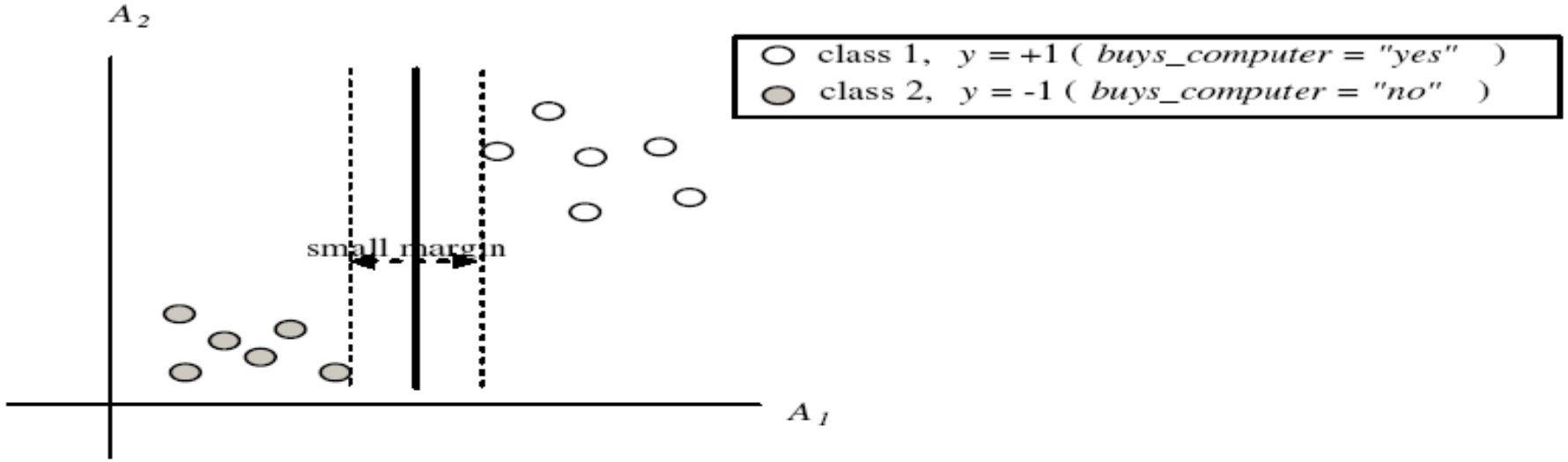
SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

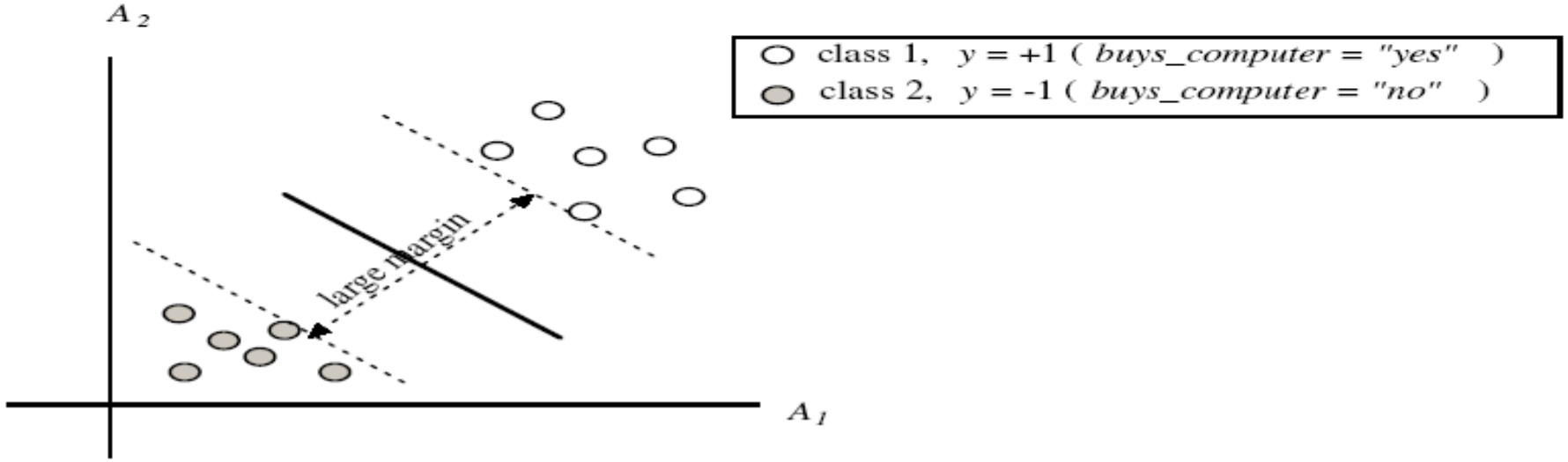
- If our data were 3-D, we would want to find the best separating plane. So generalizing to n dimensions, we want to find the best hyperplane. Hyperplane will be used to refer to the decision boundary that we are seeking, regardless of the number of input attributes. So how to find this hyperplane??
- SVM approaches this problem by searching for the **maximum marginal hyperplane**. Consider the figure of next slide, which shows two possible separating hyperplanes and their associated margins.
- Hyperplane with the larger margin is to be considered to be more accurate at classifying future data tuples than the hyperplane with the smaller margin. That is why SVM searches for the hyperplane with the largest margin, *maximum maginal hyperplane* (MMH). The associated margin gives the largest separation between classes.

SVM—General Philosophy



SVM—Margins and Support Vectors





- Getting to an informal definition of margin, we can say that the shortest distance from a hyperplane to one side of its margin is equal to the shortest distance from the hyperplane to the other side of its margin, where "sides" of the margin are parallel to the hyperplane.
- When dealing with the MMH, this distance is, in fact, the shortest distance from the MMH to the closest training tuple of either class.

SVM—Linearly Separable

- A separating hyperplane can be written as

  $$\mathbf{W} \bullet \mathbf{X} + b = 0$$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)

- For 2-D it can be written as

  $$w_0 + w_1 x_1 + w_2 x_2 = 0$$

- The hyperplane defining the sides of the margin:

  $$H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \text{ and}$$

  $$H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1 \text{ for } y_i = -1$$

- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors.**

- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

Instead of optimizing only the objective function (which is called unconstrained optimization), we need to optimize the Lagrangian of the problem, which considers the constraints at the same time. The need to consider constraints is obvious because they restrict the feasible solutions. Since our inequality constraints are expressed using "≥", the **Lagrangian** is formed by the constraints multiplied by positive Lagrange multipliers and subtracted from the objective function, i.e.,

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{n} \alpha_i [y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \tag{41}$$

where $\alpha_i \geq 0$ are the **Lagrange multipliers**.

Kuhn-Tucker Condition

$$\begin{aligned} &\text{Minimize} : \ f(\mathbf{x}) \\ &\text{Subject to} : \ g_i(\mathbf{x}) \le b_i, \quad i = 1, 2, \ldots, n \end{aligned} \tag{42}$$

where f is the objective function and g_i is a constraint function (which is different from y_i in (40) as y_i is not a function but a class label of 1 or -1). The Lagrangian of (42) is,

$$L_P = f(\mathbf{x}) + \sum_{i=1}^{n} \alpha_i [g_i(\mathbf{x}) - b_i)] \tag{43}$$

An optimal solution to the problem in (42) must satisfy the following **necessary** (but **not sufficient**) conditions:

$$\frac{\partial L_P}{\partial x_j} = 0, \quad j = 1, 2, \ldots, r \tag{44}$$

$$g_i(\mathbf{x}) - b_i \le 0, \quad i = 1, 2, \ldots, n \tag{45}$$

$$\alpha_i \ge 0, \quad i = 1, 2, \ldots, n \tag{46}$$

$$\alpha_i (b_i - g_i(\mathbf{x}_i)) = 0, \quad i = 1, 2, \ldots, n \tag{47}$$

(45) is simply the original set of constraints in (42). The condition (47) is called the **complementarity condition**, which implies that at the solution point,

$$\begin{aligned} &\text{If } \alpha_i > 0 \quad &&\text{then} \quad g_i(\mathbf{x}) = b_i. \\ &\text{If } g_i(\mathbf{x}) > b_i \quad &&\text{then} \quad \alpha_i = 0. \end{aligned}$$

These mean that for active constraints, $\alpha_i > 0$, whereas for inactive constraints $\alpha_i = 0$. As we will see later, they give some very desirable properties to SVM.

SVM

Kuhn-Tucker Conditions

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{n} \alpha_i [y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \qquad (41)$$

Let us come back to our problem. For the minimization problem (40), the Kuhn–Tucker conditions are (48)–(52):

$$\frac{\partial L_P}{\partial w_j} = w_j - \sum_{i=1}^{n} y_i \alpha_i x_{ij} = 0, \quad j = 1, 2, ..., r \qquad (48)$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n} y_i \alpha_i = 0 \qquad (49)$$

$$y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 \geq 0, \quad i = 1, 2, ..., n \qquad (50)$$

$$\alpha_i \geq 0, \quad i = 1, 2, ..., n \qquad (51)$$

$$\alpha_i (y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1) = 0, \quad i = 1, 2, ..., n \qquad (52)$$

Inequality (50) is the original set of constraints. We also note that although there is a Lagrange multiplier α_i for each training data point, the complementarity condition (52) shows that only those data points on the margin hyperplanes (i.e., H_+ and H_-) can have $\alpha_i > 0$ since for them $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 = 0$. These data points are called **support vectors**, which give the name to the algorithm, *support vector machines*. All the other data points have $\alpha_i = 0$.

In general, Kuhn–Tucker conditions are necessary for an optimal solution, but not sufficient. However, for our minimization problem with a convex objective function and a set of linear constraints, the Kuhn–Tucker conditions are both **necessary** and **sufficient** for an optimal solution.

Solving the optimization problem is still a difficult task due to the inequality constraints. However, the Lagrangian treatment of the convex optimization problem leads to an alternative **dual** formulation of the problem, which is easier to solve than the original problem, which is called the **primal** problem (L_P is called the **primal Lagrangian**).

Transforming from the primal to its corresponding dual can be done by setting to zero the partial derivatives of the Lagrangian (41) with respect to the **primal variables** (i.e., **w** and b), and substituting the resulting relations back into the Lagrangian. This is to simply substitute (48), which is

$$w_j = \sum_{i=1}^{n} y_i \alpha_i x_{ij}, \quad j = 1, 2, \ldots, r \tag{53}$$

and (49), which is

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{n} \alpha_i [y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1]$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \tag{54}$$

into the original Lagrangian (41) to eliminate the primal variables, which gives us the dual objective function (denoted by L_D),

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \tag{55}$$

L_D contains only **dual variables** and must be maximized under the simpler constraints, (48) and (49), and $\alpha_i \geq 0$. Note that (48) is not needed as it has already been substituted into the objective function L_D. Hence, the **dual** of the primal Equation (40) is

Linear SVM – Non-separable case

The geometric interpretation is shown in Fig. 3.18, which has two error data points $\mathbf{x}_a$ and $\mathbf{x}_b$ (circled) in wrong regions.

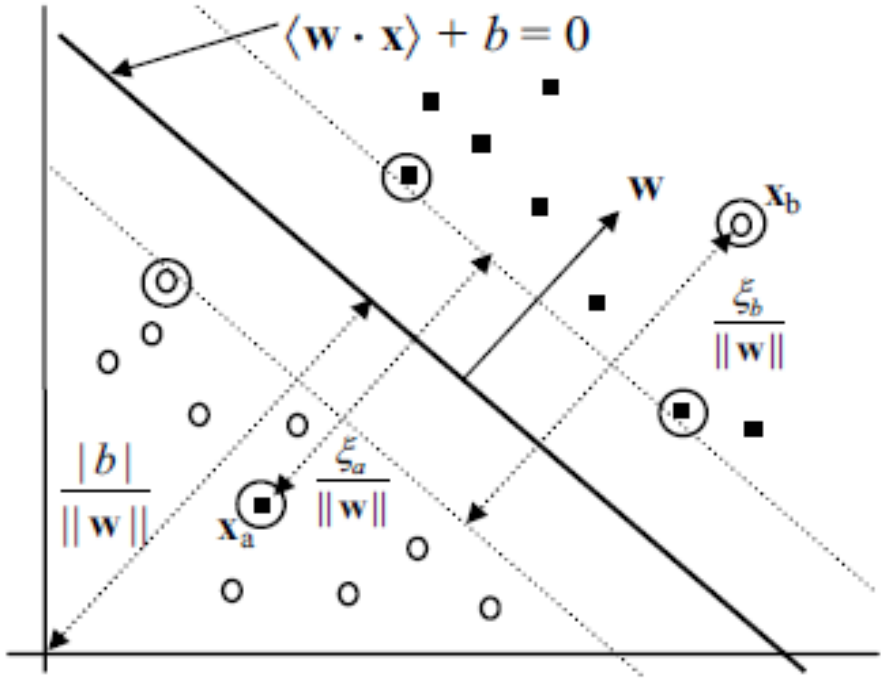


Fig. 3.18. The non-separable case: $\mathbf{x}_a$ and $\mathbf{x}_b$ are error data points

Recall that the primal for the linear separable case was:

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} \tag{59}$$

$$\text{Subject to}: y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, ..., n.$$

To allow errors in data, we can relax the margin constraints by introducing **slack** variables, ξ_i (≥ 0) as follows:

$$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1 - \xi_i \quad \text{for } y_i = 1$$
$$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 + \xi_i \quad \text{for } y_i = -1.$$

Thus we have the new constraints:

$$\text{Subject to:} \quad y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \; i = 1, 2, \ldots, n,$$
$$\xi_i \geq 0, \; i = 1, 2, \ldots, n.$$

We also need to penalize the errors in the objective function. A natural way is to assign an extra cost for errors to change the objective function to

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\left(\sum_{i=1}^{n} \xi_i\right)^k \tag{60}$$

where $C \geq 0$ is a user specified parameter. The resulting optimization problem is still a convex programming problem. $k = 1$ is commonly used, which has the advantage that neither ξ_i nor its Lagrangian multipliers appear in the dual formulation. We only discuss the $k = 1$ case below.

The new optimization problem becomes:

$$\begin{aligned}
&\text{Minimize}: \quad \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C \sum_{i=1}^{n} \xi_i \\
&\text{Subject to}: \; y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., n \\
&\qquad\qquad \xi_i \geq 0, \quad i = 1, 2, ..., n.
\end{aligned} \tag{61}$$

This formulation is called the **soft-margin SVM**. The primal Lagrangian (denoted by L_P) of this formulation is as follows

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{n} \mu_i \xi_i \tag{62}$$

where α_i, $\mu_i \geq 0$ are the **Lagrange multipliers**. The **Kuhn–Tucker conditions** for optimality are the following:

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{n} \mu_i \xi_i \qquad (62)$$

where α_i, $\mu_i \geq 0$ are the **Lagrange multipliers**. The **Kuhn–Tucker conditions** for optimality are the following:

$$\frac{\partial L_P}{\partial w_j} = w_j - \sum_{i=1}^{n} y_i \alpha_i x_{ij} = 0, \quad j = 1, 2, ..., r \qquad (63)$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n} y_i \alpha_i = 0 \qquad (64)$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0, \quad i = 1, 2, ..., n \qquad (65)$$

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i \geq 0, \quad i = 1, 2, ..., n \qquad (66)$$

$$\xi_i \geq 0, \quad i = 1, 2, ..., n \qquad (67)$$

$$\alpha_i \geq 0, \quad i = 1, 2, ..., n \qquad (68)$$

$$\mu_i \geq 0, \quad i = 1, 2, ..., n \qquad (69)$$

$$\alpha_i (y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i) = 0, \quad i = 1, 2, ..., n \qquad (70)$$

$$\mu_i \xi_i = 0, \quad i = 1, 2, ..., n \qquad (71)$$

As the linear separable case, we then transform the primal to its dual by setting to zero the partial derivatives of the Lagrangian (62) with respect to the **primal variables** (i.e., **w**, b and ξ_i), and substituting the resulting relations back into the Lagrangian. That is, we substitute Equations (63), (64) and (65) into the primal Lagrangian (62). From Equation (65), $C - \alpha_i - \mu_i = 0$, we can deduce that $\alpha_i \leq C$ because $\mu_i \geq 0$. Thus, the dual of (61) is

$$\text{Maximize: } L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \tag{72}$$

$$\text{Subject to: } \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n.$$

Interestingly, ξ_i and its Lagrange multipliers μ_i are not in the dual and the objective function is identical to that for the separable case. The only difference is the constraint $\alpha_i \leq C$ (inferred from $C - \alpha_i - \mu_i = 0$ and $\mu_i \geq 0$).

The dual problem (72) can also be solved numerically, and the resulting α_i values are then used to compute $\mathbf{w}$ and b. $\mathbf{w}$ is computed using Equation (63) and b is computed using the Kuhn–Tucker complementarity conditions (70) and (71). Since we do not have values for ξ_i, we need to get around it. From Equations (65), (70) and (71), we observe that if $0 < \alpha_i < C$ then both $\xi_i = 0$ and $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i) = 0$. Thus, we can use any training data point for which $0 < \alpha_i < C$ and Equation (70) (with $\xi_i = 0$) to compute b:

$$b = \frac{1}{y_i} - \sum_{i=1}^{n} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \tag{73}$$

Again, due to numerical errors, we can compute all possible b's and then take their average as the final b value.

Note that Equations (65), (70) and (71) in fact tell us more:

$$\begin{array}{lcl} \alpha_i = 0 & \Rightarrow & y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 \text{ and } \xi_i = 0 \\ 0 < \alpha_i < C & \Rightarrow & y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1 \text{ and } \xi_i = 0 \\ \alpha_i = C & \Rightarrow & y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \leq 1 \text{ and } \xi_i \geq 0 \end{array} \tag{74}$$

Similar to support vectors for the separable case, (74) shows one of the most important properties of SVM: the solution is sparse in α_i. Most training data points are outside the margin area and their α_i's in the solution are 0. Only those data points that are on the margin (i.e., $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1$, which are support vectors in the separable case), inside the margin (i.e., $\alpha_i = C$ and $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) < 1$), or errors are non-zero. Without this sparsity property, SVM would not be practical for large data sets.

The final decision boundary is (we note that many α_i's are 0)

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i=1}^{n} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b = 0. \quad (75)$$

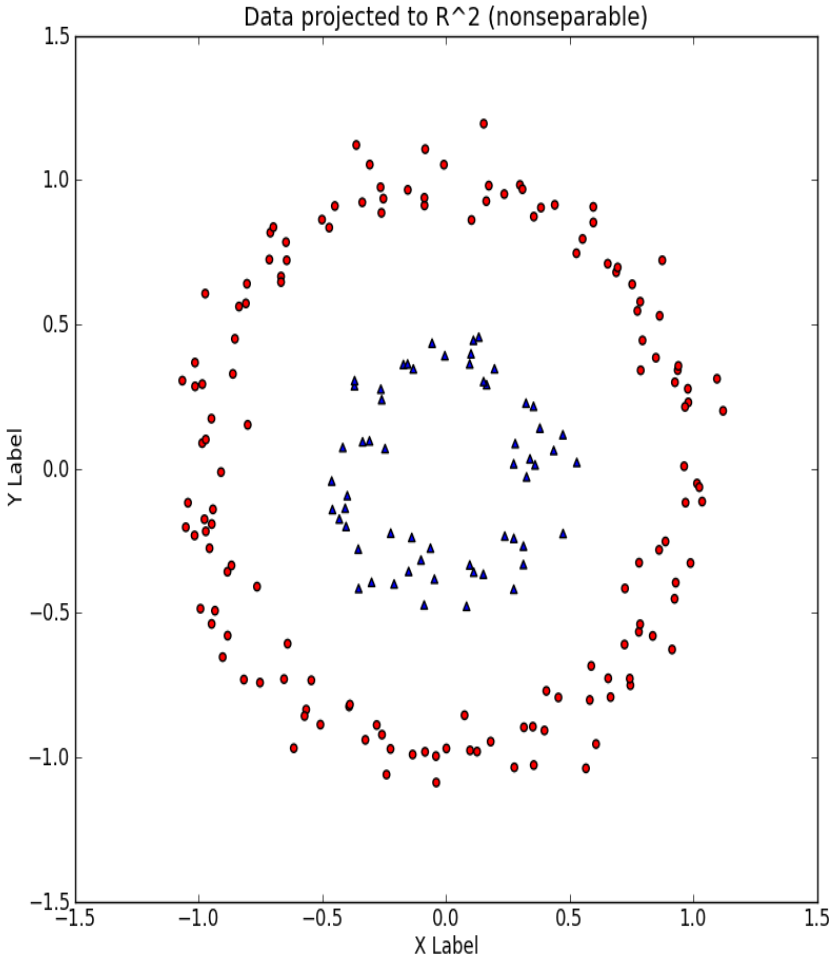
The decision rule for classification (testing) is the same as the separable case, i.e., $sign(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b)$. We notice that for both Equations (75) and (73), $\mathbf{w}$ does not need to be explicitly computed. This is crucial for using kernel functions to handle nonlinear decision boundaries.

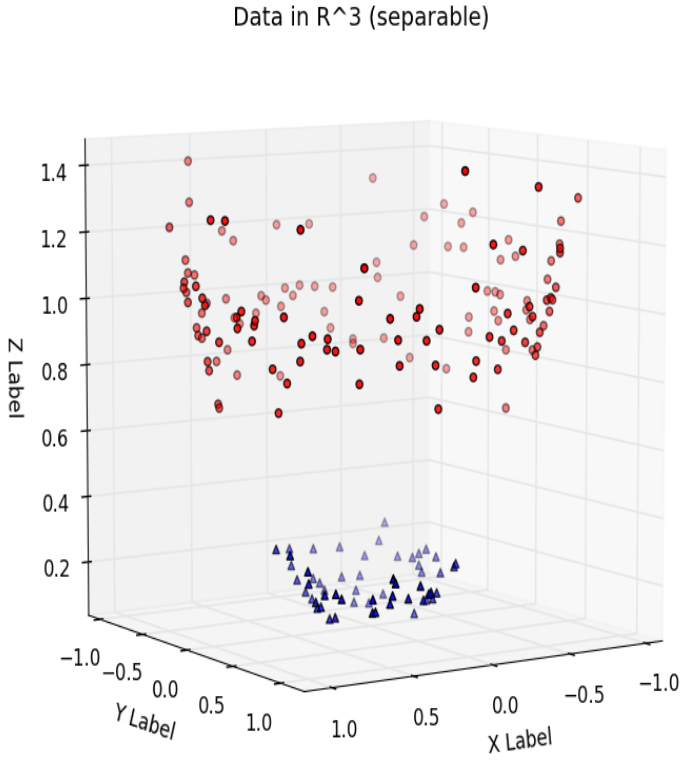
Finally, we still have the problem of determining the parameter C. The value of C is usually chosen by trying a range of values on the training set to build multiple classifiers and then to test them on a validation set before selecting the one that gives the best classification result on the validation set. Cross-validation is commonly used as well.

Why Is SVM Effective on High Dimensional Data?

- The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data.
- The **support vectors** are the essential or critical training examples —they lie closest to the decision boundary (MMH). If all other training examples are removed and the training is repeated, the same separating hyperplane would be found.
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality.
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high.

Non Linear SVM





SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space.

Example 16: Suppose our input space is 2-dimensional, and we choose the following transformation (mapping):

$$(x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2) \tag{81}$$

The training example $((2, 3), -1)$ in the input space is transformed to the following training example in the feature space:

$$((4, 9, 8.5), -1).$$ ■

- Search for a linear separating hyperplane in the new space.

Thus, the basic idea is to map the data in the input space X to a feature space F via a nonlinear mapping ϕ,

$$\begin{aligned} \phi &: X \to F \\ \mathbf{x} &\mapsto \phi(\mathbf{x}). \end{aligned} \tag{76}$$

After the mapping, the original training data set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ becomes:

$$\{(\phi(\mathbf{x}_1), y_1), (\phi(\mathbf{x}_2), y_2), \ldots, (\phi(\mathbf{x}_n), y_n)\}. \tag{77}$$

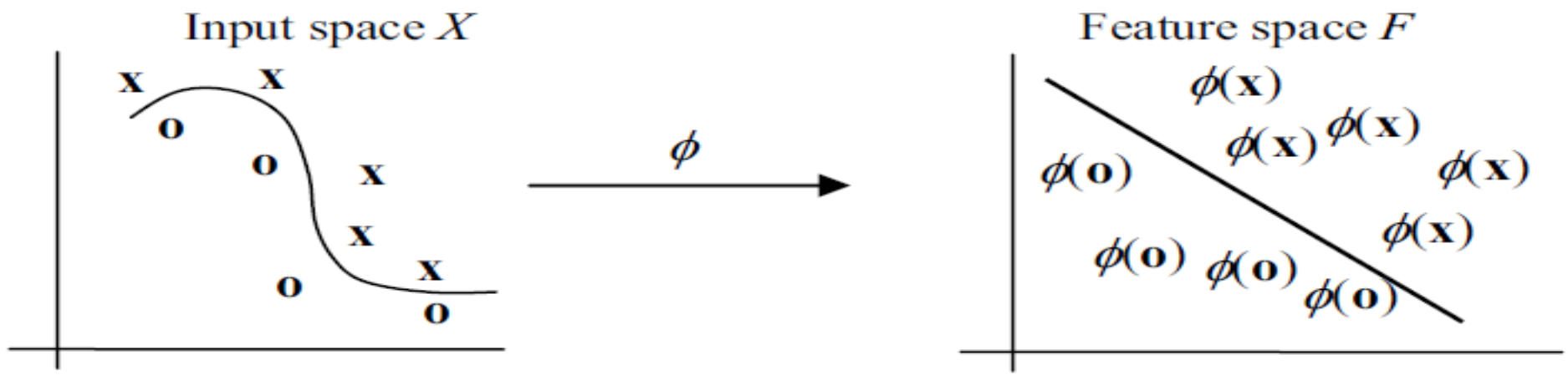


Fig. 3.19. Transformation from the input space to the feature space

Consider the following mapping ϕ for an example $\mathbf{x} = \{x_1, \ldots, x_D\}$

$$\phi : \mathbf{x} \rightarrow \{x_1^2, x_2^2, \ldots, x_D^2, , x_1x_2, x_1x_2, \ldots, x_1x_D, \ldots\ldots, x_{D-1}x_D\}$$

It's an example of a quadratic mapping

- Each new feature uses a pair of the original features

With the transformation, the optimization problem in (61) becomes

$$\text{Minimize}: \quad \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C \sum_{i=1}^{n} \xi_i \tag{78}$$

$$\text{Subject to}: \quad y_i(\langle \mathbf{w} \cdot \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., n$$

$$\xi_i \geq 0, \quad i = 1, 2, ..., n$$

Its corresponding dual is

$$\text{Maximize}: \quad L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle. \tag{79}$$

$$\text{Subject to}: \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., n.$$

The final decision rule for classification (testing) is

$$\sum_{i=1}^{n} y_i \alpha_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b \tag{80}$$

Problem: Mapping usually leads to the number of features blow up!

- Computing the mapping itself can be inefficient in such cases
- Moreover, *using* the mapped representation could be inefficient too
  - e.g., imagine computing the similarity between two examples: $\phi(\mathbf{x})^\top \phi(\mathbf{z})$

$$sign(\langle \mathbf{w} \cdot \mathbf{z} \rangle + b) = sign\left(\sum_{i \in sv} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{z} \rangle + b \right).$$

$$\sum_{i=1}^{n} y_i \alpha_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b$$

- Thankfully, Kernels help us avoid both these issues!
- The mapping doesn’t have to be explicitly computed.
- Computations with the mapped features remain efficient.

Thus, if we have a way to compute the dot product $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ in the feature space F using the input vectors $\mathbf{x}$ and $\mathbf{z}$ directly, then we would not need to know the feature vector $\phi(\mathbf{x})$ or even the mapping function ϕ itself. In SVM, this is done through the use of **kernel functions**, denoted by K,

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle, \tag{82}$$

which are exactly the functions for computing dot products in the transformed feature space using input vectors $\mathbf{x}$ and $\mathbf{z}$. An example of a kernel function is the **polynomial kernel**,

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d. \tag{83}$$

Example 17: Let us compute this kernel with degree $d = 2$ in a 2-dimensional space. Let $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

$$\begin{aligned}\langle \mathbf{x} \cdot \mathbf{z} \rangle^2 &= (x_1 z_1 + x_2 z_2)^2 \\ &= {x_1}^2 {z_1}^2 + 2 x_1 z_1 x_2 z_2 + {x_2}^2 {z_2}^2 \\ &= \langle ({x_1}^2, {x_2}^2, \sqrt{2} x_1 x_2) \cdot ({z_1}^2, {z_2}^2, \sqrt{2} z_1 z_2) \rangle \\ &= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle,\end{aligned} \tag{84}$$

where $\phi(\mathbf{x}) = ({x_1}^2, {x_2}^2, \sqrt{2} x_1 x_2)$, which shows that the kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^2$ is a dot product in the transformed feature space. The number of dimensions in the feature space is 3. Note that $\phi(\mathbf{x})$ is actually the mapping function used in Example 16. Incidentally, in general the number of dimensions in the feature space for the polynomial kernel function $K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d$ is $\binom{r+d-1}{d}$, which is a huge number even with a reasonable number (r) of attributes in the input space. Fortunately, by using the kernel function in (83), the huge number of dimensions in the feature space does not matter. ■

The derivation in (84) is only for illustration purposes. We do not need to find the mapping function. We can simply apply the kernel function directly. That is, we replace all the dot products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ in (79) and (80) with the kernel function $K(\mathbf{x}, \mathbf{z})$ (e.g., the polynomial kernel in (83)). This strategy of directly using a kernel function to replace dot products in the feature space is called the **kernel trick**. We would never need to explicitly know what ϕ is.

However, the question is, how do we know whether a function is a kernel without performing the derivation such as that in (84)? That is, how do we know that a kernel function is indeed a dot product in some feature space? This question is answered by a theorem called the **Mercer's theorem**, which we will not discuss here. See [118] for details.

18. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

It is clear that the idea of kernel generalizes the dot product in the input space. The dot product is also a kernel with the feature map being the identity

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle. \tag{85}$$

Commonly used kernels include

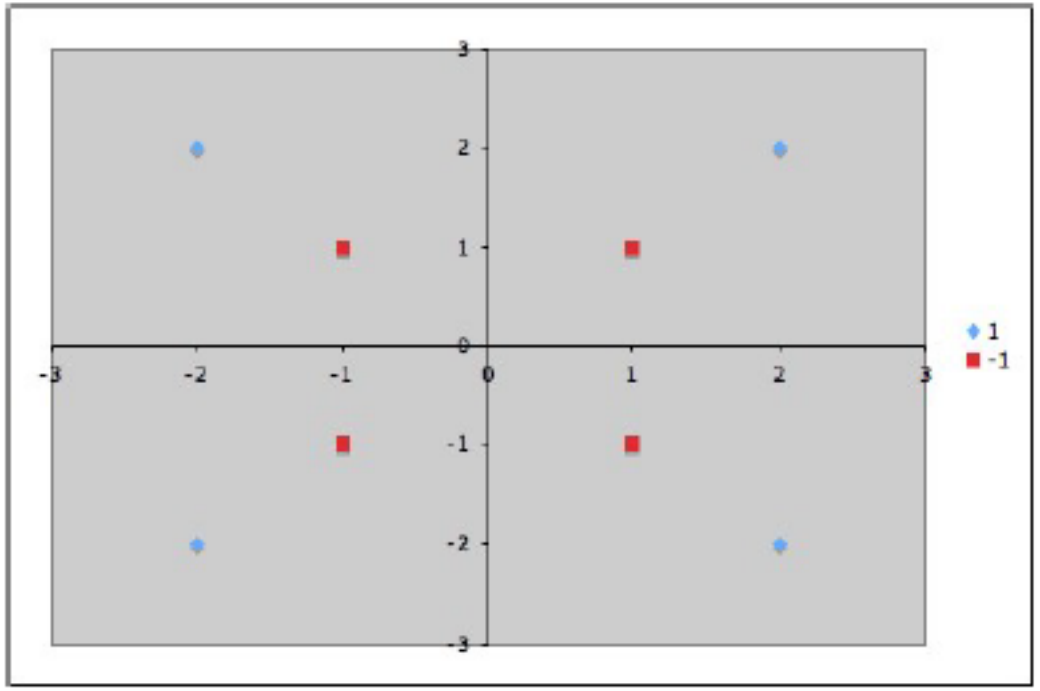
$$\text{Polynomial:} \quad K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x} \cdot \mathbf{z} \rangle + \theta)^d \tag{86}$$

$$\text{Gaussian RBF:} \quad K(\mathbf{x}, \mathbf{z}) = e^{-\|\mathbf{x}-\mathbf{z}\|^2/2\sigma} \tag{87}$$

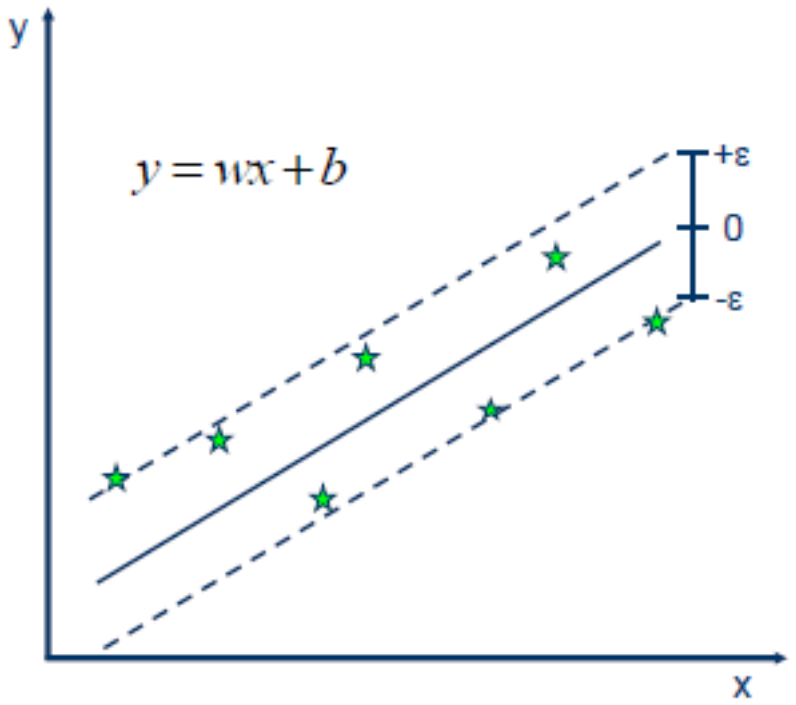
where $\theta \in \mathcal{R}$, $d \in N$, and $\sigma > 0$.

Example

- (1,1) and (2,2) are the support vectors.
- The kernel trick is polynomial kernel.



Support Vector Regression



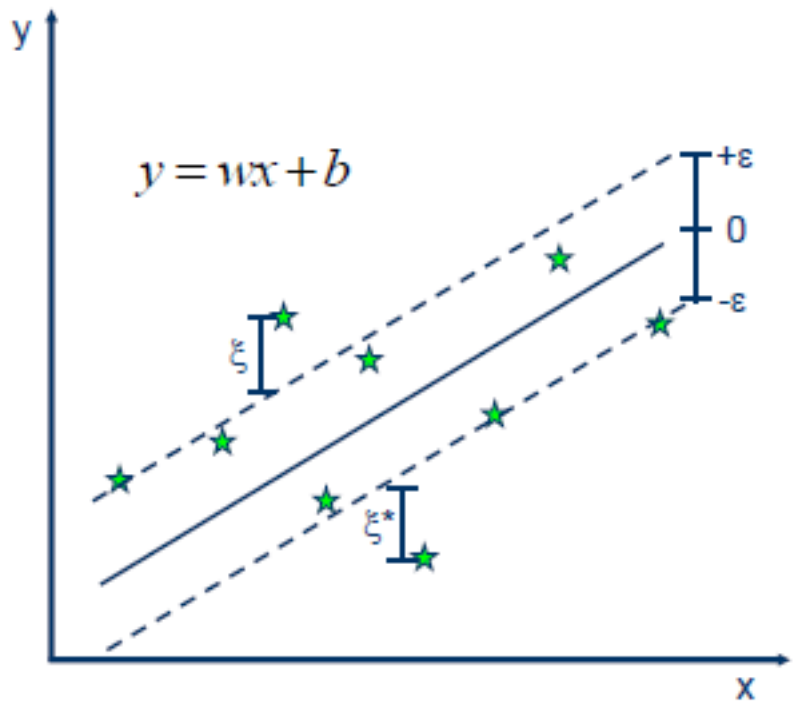
- Solution:

$$\min \frac{1}{2}\|w\|^2$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon$$

$$wx_i + b - y_i \le \varepsilon$$



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$

$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \ge 0$$

SVM also has some limitations

- It works only in real-valued space. For a categorical attribute, we need to convert its categorical values to numeric values. One way to do this is to create an extra binary attribute for each categorical value, and set the attribute value to 1 if the categorical value appears, and 0 otherwise.
- It allows only two classes, i.e., binary classification. For multiple class classification problems, several strategies can be applied, e.g., one against-rest.
- The hyperplane produced by SVM is hard to understand by users. It is difficult to picture where the hyperplane is in a high-dimensional space. The matter is made worse by kernels. Thus, SVM is commonly used in applications that do not required human understanding.