Support Vector Machine

8

To see better, go higher.

One of the key development in recent artificial intelligence (AI) research is the SVM, which has attracted a large amount of research and has produced good results in many applications. Because of the so-called "kernel trick", it has made SVM one of the most effective and efficient machine learning tools in the literature. In this chapter, we attempt to do an anatomy of SVM so that readers have a good understanding of its mechanism.

SVM is basically the combination of both a linear classifier and a k nearest neighbors classifier (K-NN). Therefore, to understand SVM, we will first introduce the linear classifier and the K-NN classifier.

We will only focus on two-class classification problem in this chapter. Because, any classification problem can be converted into a one-vs-all classification, which is a two-class classification problem.

8.1 Linear Classifier

The Bayesian methods in Chap. 7 are model based, and they can give good decision if the models are accurate. However, since data distributions are usually unknown, the models can only be estimated accurately if a large number of training samples are available. This is especially true when the number of features is large, which is common for multimedia data.

An alternative approach is to assume that there exists a functional form decision boundary between each pair of classes, and the parameters of the decision boundary or discriminant function can be estimated using available training samples. A linear classifier is one of those approaches.

D. Zhang, *Fundamentals of Image Data Mining*, Texts in Computer Science, https://doi.org/10.1007/978-3-030-17989-2_8

Suppose the data is represented as a n dimensional feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, then a linear discriminant function is formulated as

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \tag{8.1}$$

where x_i are the variables and w_i are the coefficients or weights. Assume $x_0 = 1$, $f(\mathbf{x})$ can be written as

$$f(\mathbf{x}) = \sum_{j=0}^{n} w_j x_j \tag{8.2}$$

Geometrically, $f(\mathbf{x}) = 0$ is a hyperplane in n-dimensional space and $f(\mathbf{x}) = 0$ is the decision boundary between two classes. A sample data with feature vector $\mathbf{x}$ is classified into one of the classes using the following criterion:

$$\mathbf{x} \in \begin{cases} class\,1 & if\, f(x) > 0 \\ class\,2 & if\, f(x) < 0 \end{cases}$$

8.1.1 A Theoretical Solution

The next is to find out the parameters or the set of weights of $f(\mathbf{x})$: $w_0, w_1, w_2, \ldots, w_n$, which will minimize the number of misclassified samples in a given training set.

A classical way to find the weights is to solve a set of linear equations given a set of training samples. For an n-dimensional data, $n + 1$ linear equations or samples to solve the $n + 1$ weights are needed. Suppose $d_i = 1$ (or $d_i > 0$) represents class 1 and $d_i = -1$ (or $d_i < 0$) represents class 2, and the $n + 1$ training samples are given as $(\mathbf{x}_i, d_i)$, where

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in}), \quad i = 1, 2, \ldots, n+1$$

Then, by substituting (8.2) with each of the training data $\mathbf{x}_i$, the set of weights w_i ($i = 0, 1, \ldots, n$) can be solved using the following $n + 1$ linear equations:

$$\sum_{j=0}^{n} w_j x_{ij} = d_i, \quad i = 1, 2, \ldots, n+1 \tag{8.3}$$

Equation (8.3) can be written in matrix form

$$\begin{bmatrix} 1, & x_{11}, & x_{12}, \ldots, x_{1n} \\ 1, & x_{21}, & x_{22}, \ldots, x_{2n} \\ & & \ldots..\ldots \\ 1, & x_{n1}, & x_{n2}, \ldots, x_{nn} \\ 1, & x_{n+1,1}, & x_{n+1,2}, \ldots, x_{n+1,n} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \ldots \\ w_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \ldots \\ d_n \\ d_{n+1} \end{bmatrix} \tag{8.4}$$

which in turn can be written as

$$\mathbf{X}\mathbf{w}^t = \mathbf{d}^T$$

where $\mathbf{w}^T$ and $\mathbf{d}^T$ are the transposes of $\mathbf{w}$ and $\mathbf{d}$, the solution of $\mathbf{w}$ is then given as (8.5)

$$\mathbf{w}^T = \mathbf{X}^{-1}\mathbf{d}^T \tag{8.5}$$

8.1.2 An Optimal Solution

The above theoretical solution is just based on $n + 1$ or part of the training samples, therefore, it is not optimal. An optimal solution is usually given by minimizing the squared errors of $f(\mathbf{x})$ on the entire training data set of N data ($\mathbf{x}_i$, d_i) and

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in}), \quad i = 1, 2, \ldots, N$$

That is, to minimize the following total squared error:

$$\begin{aligned} E &= \sum_{i=0}^{N-1} (f(\mathbf{x}_i) - d_i)^2 \\ &= \sum_{i=0}^{N-1} \left(\sum_{j=0}^{n} w_j x_{ij} - d_i \right)^2 \end{aligned} \tag{8.6}$$

By taking the partial derivative of E on w_k (k = 0, 1, 2, …, n) and letting the partial derivative to be 0: $\frac{\partial E}{\partial w_k} = 0$, the following $n + 1$ linear equations are obtained:

$$\sum_{i=0}^{N-1} x_{ik} \left(\sum_{j=0}^{n} w_j x_{ij} - d_i \right) = 0, \quad k = 0, 1, 2, \ldots, n \tag{8.7}$$

which is equivalent to the following:

$$\sum_{j=0}^{n} w_j \left(\sum_{i=0}^{N-1} x_{ik} x_{ij} \right) = \sum_{i=0}^{N-1} x_{ik} d_i, \quad k = 0, 1, 2, \ldots, n \tag{8.8}$$

By solving the above $n + 1$ linear equations using the same method as solving (8.3), a set of weights (w_0, w_1, w_2 …. w_n) is obtained which results is an optimal hyperplane. It can be observed that compared with (8.3), in an optimal solution algorithm, x_{ij} is replaced with $\sum_{i=0}^{N-1} x_{ik} x_{ij}$, while d_i is replaced with $\sum_{i=0}^{N-1} x_{ik} d_i$.

8.1.3 A Suboptimal Solution

Although the solution from (8.8) results in a more optimal decision boundary than that from (8.5), the solution of (8.8) would involve processing very large matrices which is computationally expensive and undesirable. This is because multimedia data usually has very high-dimensional features. An alternative approach is to use an iterative optimization algorithm to find a suboptimal solution to (8.2). Common practice is to use an error-driven weight-adaption technique which is basically a trial-and-error technique. The iterative optimization procedure is given in the following:

1. Initialize the weights w_0, w_1, w_2 … w_n with some small random values
2. Take the next training sample $\{\mathbf{x}, d\} = \{(x_1, x_2, \ldots, x_n), d\}$, $d = 1$ or -1
3. Compute $f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
4. If $f(\mathbf{x}) \neq d$ (a misclassification), update $w_0 \leftarrow w_0 + cdk$ and $w_j \leftarrow w_j + cdx_j$, $j = 1, 2, \ldots, m$; where k and c are both positive constants
5. Repeat Steps 2–4 on each of the remaining training samples, until all the samples are correctly classified or the weights stop to change.

To demonstrate that the weights w_i or the hyperplane $f(\mathbf{x})$ are moving in the right directions, let f_{new} and f_{old} be the updated value and old value of $f(\mathbf{x})$ respectively. Because k, c and the feature value x_j are all positive, after the update in step 4, all the weights w_j become larger if $d = 1$ and all the weights w_j become smaller if $d = -1$. That means, if there is a misclassification, the decision function is updated according to the following rules:

$$\begin{cases} f_{new} > f_{old} & \text{if } d = 1 \\ f_{new} < f_{old} & \text{if } d = -1 \end{cases}$$

Therefore, in either case, the new hyperplane $f(\mathbf{x})$ is moving in the right direction with the updated weights, until the misclassified sample is located at the correct side of the hyperplane.

A linear classifier can only classify data which are linearly separable. However, this idea can be extended to build a nonlinear classifier. For example, we can convert a two-dimensional feature vector (x, y) in xy space to a five-dimensional feature vector $(u_1, u_2, u_3, u_4, u_5)$ in a higher dimensional space, where $u_1, = \sqrt{2}x$, $u_2 = \sqrt{2}y$, $u_3 = x^2$, $u_4 = \sqrt{2}xy$, $u_5 = y^2$. This would be the polynomial kernel $(1 + x + y)^2$. This is the key idea behind the kernel method which will be discussed in Sect. 8.3.

8.2 K-Nearest Neighbors Classification

K-nearest neighbors or K-NN is a simple algorithm that stores all available cases and classifies new cases based on a decision function (e.g., a distance measure).

Given a training dataset D and a distance measure *dist*:

- $(\mathbf{x}_i, y_i)$, $i = 1, 2, \ldots, N$
- $\mathbf{x}_i$ is a training data in R^n
- y_i is the corresponding class of the data $\mathbf{x}_i$, and $y_i \in \{c_j, j = 1, 2, \ldots, M\}$
- $dist\ (\mathbf{x} - \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|$.

A new observation data $\mathbf{x}$ is classified into one of the classes y_j using the following algorithm:

1. Input the new data $\mathbf{x}$
2. Compute the distance of $\mathbf{x}$ to all the training samples $\mathbf{x}_i$ in the dataset: $dist(\mathbf{x} - \mathbf{x}_i)$
3. Sort $dist(\mathbf{x} - \mathbf{x}_i)$ $(i = 1, 2, \ldots, N)$ in ascending order and rank all the $\mathbf{x}_i$ accordingly: $\mathbf{x}_{r1}, \mathbf{x}_{r2}, \ldots, \mathbf{x}_{rk}, \ldots, \mathbf{x}_{rN}$
4. For the nearest neighbor (NN) classification, classify $\mathbf{x}$ to y_{r1}

    a. For a K-NN classification, classify $\mathbf{x}$ to the majority class y_{rp} among the top k ranked data: $\{\mathbf{x}_{r1}, \mathbf{x}_{r2}, \ldots, \mathbf{x}_{rk}\}$.

Although Euclidean (L_2) and city block distance (L_1) are a typical choice for the distance measure, any other distance can be used depending on the applications. The nearest neighbor (NN or 1-NN) results in too many classes, while K-NN gives more reliable classification results. This is because the values of k have a smoothing effect that makes the classifier more resistant to outliers. However, the performance of a K-NN classifier depends on the choice of k which is usually determined empirically.

Figure 8.1 demonstrates the comparison between an NN classifier and a K-NN classifier [1]. It can be seen from the two classification results, in the case of a NN classifier (after merging), outlier data points create small islands within a class (e.g., red point within the green class) and sharp corners on the class boundaries, those islands, and sharp corners likely lead to incorrect predictions; while the 5-NN

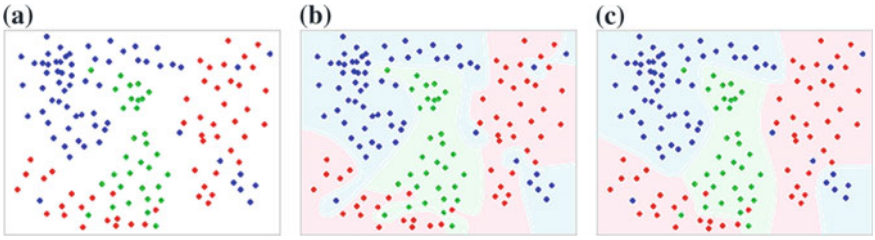


Fig. 8.1 Comparison between NN and K-NN. **a** The data to be classified; **b** classification result from a 1-NN classifier; **c** classification result from a 5-NN classifier

classifier smooths over these outliers, which lead to better classification on the data. However, the 5-NN classifier also causes misclassifications which are characterized by the blue dots in red region and red dots in green region. There can also be confusions by the tied votes among the five nearest neighbors (e.g., two neighbors are red, the next two neighbors are blue, and the last neighbor is green).

This kind of misclassification can be overcome to a certain extent by using the *weighted* K-NN. The idea is to give more weight to the neighbors with shorter distance to the test data than to the faraway neighbors. The commonly used weighted K-NN is the Gaussian weighted K-NN.

Unlike any other classifiers which are independent of the original training data once trained, a K-NN classifier is memoryless. If we analog a classifier to a connoisseur traveling around the world to judge (classify) different kinds of antiques for people. While other types of connoisseurs just need to take a toolkit summarizing the key characteristics of the antiques, a K-NN connoisseur will have to carry every kind of real antiques in his/her collection in order to make a new judgement. This may sound too cumbersome, however, one of the key advantages of a K-NN classifier is that it can classify data which are nonlinearly separable. This is the key idea behind the kernel-based support vector machine (SVM).

8.3 Support Vector Machine

In the previous two sections, we have introduced the linear classifier and K-NN classifier, both are key to understand SVM.

A linear classifier is simple and once trained, it is like a tool or a machine which can be used to tell if a data belongs to one of the classes. However, the disadvantages of a linear classifier include

- The solution is either not optimal or computationally expensive
- It cannot classify data which are nonlinearly separable

A K-NN classifier is also simple and can separate data nonlinearly. However, the disadvantages of a K-NN classifier include

- It is difficult to choose a k
- Dependence on training data.

Now that we have understood how the linear classifier and the K-NN classifier behave, we would like to build a classifier which takes advantage of both and overcomes their disadvantages. This is SVM.

An SVM is basically a binary linear classifier, however, with two prominent goals to achieve:

- To maximize the margin which separates the two classes (optimal)
- To use only a few training data (or support vectors) to determine the hyperplane which separates the two classes (efficient)

A kernel-based SVM adds another goal to

- Be able to classify data which are nonlinearly separable

As can be seen, once the three goals are achieve, we would truly build a machine which combines the advantages of both the linear classifier and the K-NN classifier, while overcomes their disadvantages.

To formulate SVM, we will start with the simple perceptron and the primal form of SVM. In the next, the dual form of SVM is introduced, and finally, the kernel-based SVM is described in details.

8.3.1 The Perceptron

A perceptron is a binary linear classifier which is one of the simplest classifiers. Given an unknown data, the perceptron simply generates a linear prediction. The training process is the same as the linear classifier introduced in Sect. 8.1. The only difference is that a perceptron can do online learning, which means it can process the training data one at a time instead of having to taking the entire training dataset. Although it is simple, the perceptron is the key to understand both SVM and Artificial Neural Network (ANN) later on.

Given a training dataset D:

- $D = \{(\mathbf{x}_i, y_i),\ i = 1, 2, \ldots, N\}$
- $\mathbf{x}_i$ is a feature vector in n dimensional space: $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$
- y_i is the corresponding class of the data $\mathbf{x}_i$, and $y_i \in \{-1,\ 1\}$
- $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i \cdot \mathbf{x}_j$ is the dot product between two vectors

A perceptron is a binary linear classifier which is formulated as follows:

1. $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$
2. Let $w_0 = b$ and $x_0 = 1$, then the above can be simply written as $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$
3. $h(\mathbf{x}) = sign\ (f(\mathbf{x})) = y_i\ (f(\mathbf{x}))$
4. Take the next training data $(\mathbf{x}_i, y_i) \in D$
5. *if* $h(\mathbf{x}_i) \geq 0$, $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k$
6. *if* $h(\mathbf{x}_i) < 0$
   then $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta\ y_i\ \mathbf{x}_i$, $\eta > 0$
7. *Repeat from* 4

8.3.2 SVM—The Primal Form

8.3.2.1 The Margin Between Two Classes

Continue from the perceptron discussion and its training data assumption.

The perceptron gives us a hyperplane to separate the two classes of data, however, there are an infinite number of hyperplanes between two classes of data as shown in Fig. 8.2. The one resulted from the perceptron is just one of them, and it is nothing optimal. Although an optimal hyperplane was given in Sect. 8.1, it is optimal only in terms of minimizing the total error, and it is still far from the optimal hyperplane we *perceive*.

The optimal or the best hyperplane we *perceive* is the one separating the two classes with the maximal margin as shown in Fig. 8.3. That is the hyperplane we are going to find out.

Assume the two subspaces corresponding to the two classes of data are, respectively,

$$\langle \mathbf{w}, \mathbf{x} \rangle + b \geq 1 for\ y_i = +1$$
$$\langle \mathbf{w}, \mathbf{x} \rangle + b \leq -1 for\ y_i = -1$$

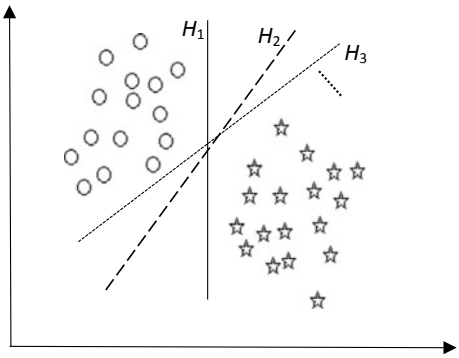


Fig. 8.2 Hyperplanes between two classes of data

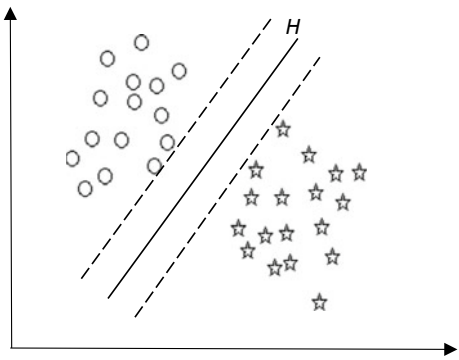


Fig. 8.3 The optimal hyperplane H between two classes of data

The above two inequalities can be combined into one

$$y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b) - 1 \geq 0 \tag{8.9}$$

The boundaries between the two subspaces are hyperplanes H_1 and H_2, respectively,

$$H_1 : \quad \langle \mathbf{w}, \mathbf{x} \rangle + b - 1 = 0 \tag{8.10}$$

$$H_2 : \quad \langle \mathbf{w}, \mathbf{x} \rangle + b + 1 = 0 \tag{8.11}$$

and the hyperplane between H_1 and H_2 is given as H_0:

$$H_0 : \quad \langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{8.12}$$

The two classes of data and the three hyperplanes separating them are shown in Fig. 8.4.

Our purpose is not only to find H_0 but also to maximize the distance between H_1 and H_2, which is the margin between the two classes of data. How to work out the distance between H_1 and H_2? Here is how it works out.

- Remember in a 2D space, a hyperplane is just a line which is expressed as: $ax + by + c = 0$. The constant c is called the intercept, and $|c|$ is associated with the distance from the origin to the line.
- This is also true in higher dimensional space. For example, in a 3D space, a plane is given as $Ax + By + Cz + D = 0$, and $|D|$ is associated with the distance from the origin to the plane. So on so forth.
- Therefore, the distance between H_1 and H_2 is equal to the difference between the distance of each of them to the origin.

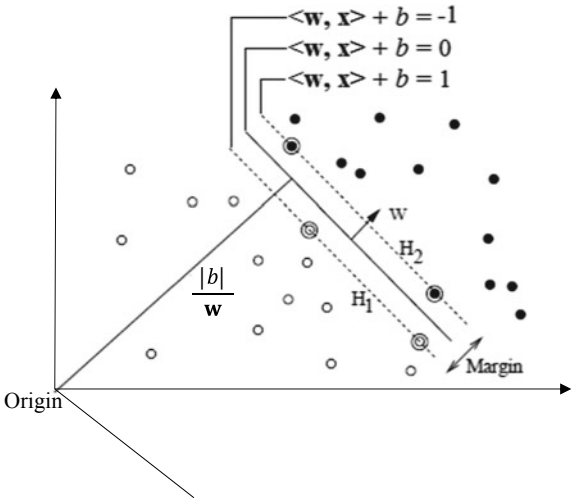


Fig. 8.4 Two classes of data and the hyperplanes separating them

Specifically, based on the theory of geometry, the distance of a point (x_0, y_0, z_0) to a plane in 3D space: $Ax + By + Cz + D = 0$ is given as follows:

$$\frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}} \tag{8.13}$$

Because this is also true for higher dimensional space, accordingly, the distance from H_0 to the origin (0, 0, …, 0) in n-dimensional space is given as

$$\frac{|b|}{||\mathbf{w}||} \tag{8.14}$$

where $\|\mathbf{w}\|$ is the magnitude or length of vector $\mathbf{w}$, and the distance from H_1 and H_2 to the origin (0, 0, …, 0) in n-dimensional space is given by the following two respectively:

$$\frac{|b+1|}{||\mathbf{w}||} \quad \text{and} \quad \frac{|b-1|}{||\mathbf{w}||} \tag{8.15}$$

Therefore, by calculating the difference between the two terms of (8.15), the margin between the two hyperplanes H_1 and H_2 is obtained as

$$\frac{2}{||\mathbf{w}||} \tag{8.16}$$

The data points which lie on H_1 and H_2 are called *support vectors* (marked by circles in Fig. 8.3), which are both necessary and sufficient to define the boundary hyperplanes.

8.3.2.2 Margin Maximization

Therefore, based on (8.16), to maximize the distance between the two subspaces is equivalent to the following optimization problem:

$$\textit{Minimize}: \quad f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle \tag{8.17}$$

$$\textit{Subject to}: \quad g_i(\mathbf{w}, b) = y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) - 1 \geq 0,\ i = 1, 2, \ldots, N \tag{8.18}$$

The above is a constrained optimization problem, and there are a few important facts to be pointed out [2]:

- b is one of the weights to be found because if we let $x_0 = 1$, then $w_0 = b$
- Equation (8.17) is a paraboloid in n-dimensional space
- A paraboloid has a single global minimum at the bottom
- Equation (8.18) is a hyperplane in n-dimensional space
- The solution to this constrained optimization problem is at the tangent point of the paraboloid and the hyperplane
- At the tangent point, the normal vectors or gradient vectors of both the paraboloid and the hyperplane are parallel
- That is, $\nabla f = \alpha_i \nabla g_i$, ($i$ = 1, 2, …, N), where ∇ is the gradient and α_i is a constant.

Based on the above analysis, the optimization problem of (8.17) and (8.18) is equivalent to combining them into the following *Lagrange function* and solve $\nabla L = 0$ or $\partial_{\mathbf{w},\, b}\, L = 0$:

$$L(\mathbf{w},\ b,\ \alpha_i) = f(\mathbf{w}) - \sum_i \alpha_i g_i(\mathbf{w}, b) \tag{8.19}$$

$$\begin{aligned} L(\mathbf{w}, b, \alpha_i) &= \frac{1}{2}|\mathbf{w}|^2 - \sum_i \alpha_i [y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) - 1] \\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \sum_i \alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b) + \sum\nolimits_i \alpha_i \end{aligned} \tag{8.20}$$

8.3.2.3 The Primal Form of SVM

From (8.20), we can obtain the *primal form* of the SVM:

$$\begin{aligned} &\textit{Minimize}: \quad L(\mathbf{w}, b, \alpha_i) = \tfrac{1}{2}\mathbf{w}\cdot\mathbf{w} - \sum_i \alpha_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b) + \sum_i \alpha_i \\ &\textit{Subject to}: \quad \alpha_i \geq 0, i = 1, 2, \ldots, N \end{aligned} \tag{8.21}$$

8.3.3 The Dual Form of SVM

Although the primal form (8.21) let us to find the weights $\mathbf{w}$ and a hyperplane which separates the two classes of data with the maximum margin, the optimization is too expensive. Because we have to optimize two sets of parameters at the same time: $\mathbf{w}$ and α_i, this is very undesirable. Next, we want to make it more efficient.

Since (8.21) is a quadratic function, based on mathematics, at the global minima of the quadratic function, the gradient or the partial derivatives of $L(\mathbf{w}, b, \alpha_i)$ are 0. Therefore, we have

$$\nabla L(\mathbf{w}, b, \alpha_i) = \nabla f(\mathbf{w}) - \nabla \left[\sum_i \alpha_i g_i(\mathbf{w}, b) \right] = 0 \tag{8.22}$$

Now let

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \qquad and \qquad \frac{\partial L}{b} = 0$$

This leads to

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad and \quad \sum\nolimits_i \alpha_i y_i = 0 \tag{8.23}$$

Substituting the primal form (8.21) with (8.23) leads to the following *dual form* of the SVM:

$$\begin{aligned} &Maximize: \quad L_D = \sum_i \alpha_{i-} \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &Subject\ to: \quad \alpha_i \geq 0 \quad and \quad \sum_i \alpha_i y_i = 0 \end{aligned} \tag{8.24}$$

To see why it has changed from minimization in the primal form to maximization in the dual form, let us have a good look at (8.24). The value of L_D is determined by the following three cases [2]:

- If the two features $\mathbf{x}_{i,}$ $\mathbf{x}_j$ are completely dissimilar ($\mathbf{x}_{i,}$ $\mathbf{x}_j$ are from different classes and are very different), their dot product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = 0$, that means, *features from different classes are far away from the boundaries between two classes don't contribute to* L_D
- If the two features $\mathbf{x}_{i,}$ $\mathbf{x}_j$ are completely alike and from the same class, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx 1$ and $y_i\ y_j = 1$. Therefore, $\alpha_i\ \alpha_j\ y_i\ y_j\ \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ would be positive and this would decrease the L_D. That means, L_D *downgrades similar features in the same class but far away from the boundaries between two classes*
- If the two features $\mathbf{x}_{i,}$ $\mathbf{x}_j$ are completely alike but from the different class, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx 1$ but $y_i\ y_j = -1$. Therefore, $\alpha_i\ \alpha_j\ y_i\ y_j\ \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ would be negative, this would increase L_D or maximize it. That means, L_D *is maximized with similar*

features from different classes or L_D *is maximized with features on the opposite boundaries of two classes*

To summarize the above analysis, by *maximizing* L_D, the dual form SVM

1. Emphasizes the feature vectors on the opposite boundaries between two classes
2. Ignores or suppresses those feature vectors far away from the boundaries between two classes.

This is exactly what we want because in terms of finding the hyperplanes separating the two classes with maximum margin, only those vectors on or close to the boundaries between the two classes matter most. Those feature vectors are called *support vectors* and the classifier defined by support vectors is called a *support vector machine*.

8.3.3.1 The Dual Form Perceptron

Because L_D is determined by the small number of support vectors on the boundaries between the two classes, not surprisingly, most of the α_i would be zero. Once α_i, $i = 1, 2, \ldots, N$ are solved, the weights for the hyperplane separating the two classes of data with the maximum margin are given as follows:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{8.25}$$

Therefore, the weight of the SVM hyperplane is just a linear combination of the training data, and this is consistent with the weight updating methods used in the linear and perceptron classifiers introduced earlier.

A set of α_i can be estimated using the *dual form* perceptron:

1. $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$
   $= \sum_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b$
2. Take the next training data $(\mathbf{x}_j, y_j) \in D$
3. *if* y_j $(\Sigma_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b) \geq 0$
   then $\alpha_{i+1} \leftarrow \alpha_i$
4. *if* y_j $(\Sigma_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b) < 0$
   then $\alpha_{i+1} \leftarrow \alpha_i + \eta$, $\eta > 0$
5. *Repeat from step* 2

8.3.4 Kernel-Based SVM

8.3.4.1 The Dual Form SVM Versus NN Classifier

With the dual form SVM (8.24), we have successfully reduced the primal form optimization problem to optimizing just one set of parameters: α_i, $i = 1, 2, \ldots, N$. This is much more efficient than (8.21). However, this is just a small part of the story about SVM, the more important part of the story is the transform of SVM optimization from testing $\langle \mathbf{w}, \mathbf{x}_i \rangle$ to testing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. This is explained in the following:

- An n-dimensional data $\mathbf{x}$ is a feature vector in space, and geometrically, the dot product is defined as follows:

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \|\mathbf{x}_i\| \|\mathbf{x}_j\| \cos\theta \tag{8.26}$$

  where θ is the angle between the two feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$
- In practice, the magnitudes of all feature vectors are normalized to unit or 1 so that they can be fairly matched
- Therefore, the dot production of two feature vector is just $\cos\theta$
- Because all feature values are positive, θ is between 0° and 90°
- For two feature vectors at the same direction or $\theta = 0^\circ$ (identical), the dot product is 1: $\cos\theta = 1$
- For two feature vectors at vertical angle or $\theta = 90^\circ$ (completely different), the dot product is 0: $\cos\theta = 0$
- For two feature vectors at an angle $0^\circ < \theta < 90^\circ$ (between similar to different), the dot product is between (0, 1): $0 < \cos\theta < 1$
- Therefore, the dot product $\cos\theta$ actually measures the similarity between the two feature vectors, or, *the dot product is just the cosine distance between the two feature vectors*

Equipped with this key finding, now let us go back to (8.24):

$$\text{Maximize} \quad L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

It is equivalent to

$$\text{Minimize} \quad \sum_{ij} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{8.27}$$

Because $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is just the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, by recalling what has been discussed in the K-NN section, we can see that (8.27) is just the *weighted nearest neighbors* classification.

Now, if we look at the dual form perceptron at the end of Sect. 8.3.3, the connection between SVM and K-NN is even clearer. The dual form classifier is given as (8.28)

$$f(\mathbf{x}) = \sum_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \tag{8.28}$$

The classification of each training data $\mathbf{x}_j$ is done by testing

$$y_i \left(\sum_i \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right) \geq 0, \; j = 1, 2, \ldots, N \tag{8.29}$$

Again, this is just a *weighted nearest neighbors classifier*.

This is a significant development, because, by using the dual form, we not only make the SVM more efficient but also make it a *nonlinear classifier*.

8.3.4.2 Kernel Definition

These are some of the key points obtained from the above:

- The dot product is a kind of distance
- The dual form SVM is a kind of weighted nearest neighbors classifier
- The weighted nearest neighbors classifier is a nonlinear classifier

Now that we understand how an important role the dot product plays in the dual form SVM, we can extend this idea to any function behaves like a dot product.

It turns out the dot product of data points can be generalized as *kernelling*. Any function $K(\mathbf{x})$ which has the following property can be regarded as a *kernel*

$$K(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle \tag{8.30}$$

where $\Phi(\mathbf{x})$ is a function transforming feature vector $\mathbf{x}$ in one space R^m to another higher dimensional space R^n ($n > m$). From the definition, a kernel behaves like a dot product, it takes two feature vectors as input and maps the two vectors to a scalar or a real value. The difference of a kernel from a dot product is that a kernel do the dot product at a higher dimensional space, called the Hilbert space. We will explain the benefit of doing this.

Not surprisingly, with this definition, the dot product itself is a kernel because

$$K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle \; = \; \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \tag{8.31}$$

where $\Phi(\mathbf{x}) = \mathbf{x}$.

Given a kernel, the kernel-based SVM can now be written as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K\langle \mathbf{x}_i, \mathbf{x}\rangle + b \tag{8.32}$$

The questions now are:

1. Are there any other kernel functions than the dot product?
2. How useful is a kernel?

The answer to the first question is yes, there are many such kinds of kernel functions. Common kernel functions used in multimedia data classification include the following:

1. Quadratic Kernel

$$K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y}\rangle^2 \text{ and } [1 + \langle \mathbf{x}, \mathbf{y}\rangle]^2 \tag{8.33}$$

2. Polynomial Kernel

$$K(\mathbf{x}, \mathbf{z}) = <\mathbf{x}, \mathbf{y}>^d \quad \text{and} \quad [1 + <\mathbf{x}, \mathbf{y}>]^d, d > 2 \tag{8.34}$$

3. Radial Basis Function (RBF) Kernel

$$K(\mathbf{x}, \mathbf{y}) = e^{-\gamma||\mathbf{x}-\mathbf{y}||^2}, \gamma > 0 \tag{8.35}$$

To demonstrate these functions having the kernel property of (8.30), let us assume $\mathbf{x}$ and $\mathbf{y}$ are in R^2 and $\mathbf{x} = (x_1, x_2)$, $\mathbf{y} = (y_1, y_2)$.

For $\langle \mathbf{x}, \mathbf{y}\rangle^2$:

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y}\rangle^2 &= [(x_1, x_2) \cdot (v_1, y_2)]^2 = (x_1 y_1 + x_2 v_2)^2 \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 \\ &= \langle (x_1^2, \sqrt{2}x_1 x_2, x_2^2), (y_1^2, \sqrt{2}y_1 y_2, y_2^2)\rangle \end{aligned} \tag{8.36}$$

$$\begin{aligned} \text{or} \quad &= \langle (x_1^2, x_1 x_2, x_2 x_1, x_2^2), (y_1^2, y_1 y_2, y_2 y_1, y_2^2)\rangle \\ &= \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle \end{aligned} \tag{8.37}$$

where $\Phi(\mathbf{x}) = \left(x_1^2, \sqrt{2}x_1 x_2, x_2^2\right)$ or $(x_1^2, x_1 x_2, x_2 x_1, x_2^2)$ is a function which maps a 2D feature vector to a 3D or 4D feature vector. Therefore, $\langle \mathbf{x}, \mathbf{y}\rangle^2$ is a kernel, and so is $\langle \mathbf{x}, \mathbf{y}\rangle^d$ when $d > 2$.

For $(1+\mathbf{x}\cdot\mathbf{y})^2$, we have:

$$\begin{aligned}(1+\mathbf{x}\cdot\mathbf{y})^2 &= [1+(x_1,x_2)\cdot(y_1,y_2)]^2\\ &= (1+x_1y_1+x_2y_2)^2\\ &= 1+2x_1y_1+2x_2y_2+2x_1x_2y_1y_2+x_1^2y_1^2+x_2^2y_2^2\\ &= \langle(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2),(1,\sqrt{2}y_1,\sqrt{2}y_2,y_1^2,\sqrt{2}y_1y_2,y_2^2)\rangle\\ &= \langle\Phi(\mathbf{x}),\Phi(\mathbf{y})\rangle\end{aligned} \tag{8.38}$$

where $\Phi(\mathbf{x}) = \left(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,\sqrt{2}x_1x_2,x_2^2\right)$ is a function mapping a 2D feature vector to a 6 dimensional feature vector. Therefore, $(1+\mathbf{x}\cdot\mathbf{y})^2$ is also a kernel, so is $(1+\mathbf{x}\cdot\mathbf{y})^{\mathrm{d}}$ for $d > 2$.

In general, a quadratic kernel $\langle\mathbf{x},\mathbf{y}\rangle^2$ transforms an n dimensional vector $\mathbf{x}$ = (x_1, x_2, …, x_n) to vector in $n(n+1)/2$-dimensional space:

$$\Phi:\mathbf{x}\rightarrow(x_1^2,x_2^2,\ldots,x_n^2,x_1x_2,x_1x_3,\ldots,x_1x_n,x_2x_3,\ldots,x_2x_n,\ldots,x_{n-1}x_n) \tag{8.39}$$

For RBF $e^{-\gamma\|\mathbf{x}-\mathbf{z}\|^2}$, again assume $\mathbf{x}$ and $\mathbf{z}$ are in 2D, since

$$\begin{aligned}\|\mathbf{x}-\mathbf{z}\|^2 &= (x_1-z_1)^2+\left(x_2-z^2\right)^2\\ &= x_1^2+z_1^2-2x_1z_1+x_2^2+z_2^2-2x_2z_2\end{aligned}$$

Without loss of generality, let γ = ½, then we have

$$\begin{aligned}e^{-\gamma\|\mathbf{x}-\mathbf{z}\|^2} &= e^{-\frac{1}{2}\left(x_1^2+z_1^2-2x_1z_1+x_2^2+z_2^2-2x_2z_2\right)}\\ &= e^{-\frac{1}{2}\left(x_1^2+x_2^2\right)}e^{-\frac{1}{2}\left(z_1^2+z_2^2\right)}e^{(x_1z_1+x_2z_2)}\\ &= e^{-\frac{1}{2}\|\mathbf{x}\|^2}e^{-\frac{1}{2}\|\mathbf{z}\|^2}e^{\langle\mathbf{x},\mathbf{z}\rangle}\\ &= Ce^{\langle\mathbf{x},\mathbf{z}\rangle}\\ &= C\sum_{n=0}^{\infty}\frac{\langle\mathbf{x},\mathbf{z}\rangle^n}{n!}\quad(\text{Taylor expansion of } e^x)\\ &= C\sum_{n=0}^{\infty}\frac{K_{poly(n)}(\mathbf{x},\mathbf{z})}{n!}\end{aligned} \tag{8.40}$$

where $C = e^{-\frac{1}{2}\|\mathbf{x}\|^2}e^{-\frac{1}{2}\|\mathbf{z}\|^2}$ is a constant because feature vector are normalized to unit length, and $K_{poly(n)}(\mathbf{x},\mathbf{z})$ is a polynomial kernel [3]. Therefore, the RBF is a kernel

because the sum of kernels is also kernel (see the following). Equation (8.40) shows that the RBF maps a vector into a space with infinite dimensions.

8.3.4.3 Building New Kernels

It can be shown that the following rules are true:

1. The sum of two kernels is also a kernel

$$K(\mathbf{x}, \mathbf{y}) = K_1(x, y) + K_2(x, y) \tag{8.41}$$

2. A scalar times a kernel is also a kernel

$$K(\mathbf{x}, \mathbf{y}) = aK_1(x, y) \tag{8.42}$$

3. The product of two kernels is also a kernel

$$K(\mathbf{x}, \mathbf{y}) = K_1(x, y) \times K_2(x, y) \tag{8.43}$$

Therefore, by using these rules and existing kernels, we may build more kernels for different applications.

8.3.4.4 The Kernel Trick

Now that we have defined the kernels and understood their behaviors, the next is to answer the *second question* we mentioned earlier. That is, why kernels or why we transform a feature vector to a higher dimensional space? It appears the dual form SVM is good enough because it not only gives us a SVM but also let us do nonlinear classification. So what is the benefit of using kernels?

There are two reasons to use a kernel instead of just the dot product.

- One is to transform nonlinear data in lower dimensional space to linear data in higher dimensional space so that they can be separated linearly using the SVM.
- The other is to have more and better choices of distance measurement than the dot product, so as to improve the performance of an SVM.

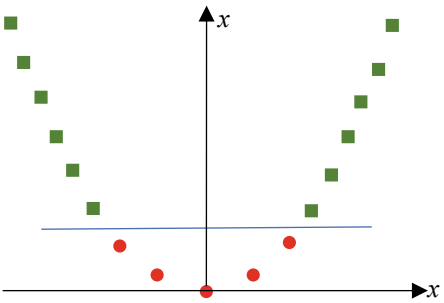
To demonstrate how a kernel can transform nonlinear data into linear data, we will use the quadratic kernels as examples.

Consider the following 1D binary data (red and green dots) which is a nonlinear data because it cannot be separated by a point or a line.

Now map each of the samples using the following function:

$$\Phi : x \rightarrow \{x, x^2\} \tag{8.44}$$

Φ is a quadratic mapping, it transforms a 1D line into a 2D parabola:



By transforming the 1D data into a 2D space, now the data in 2D space can be separated using a line (blue) or linearly separable. This is exactly the first reason for using kernel. This phenomenon can also be demonstrated using a 2D nonlinear data (Fig. 8.5) [4]. By using the following mapping function to map the 2D data on the left of Fig. 8.5 to a paraboloid in 3D space, the data can now be separated using a 2D plane and is linearly separable:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{8.45}$$

Because the dot product is kind of distance measure, therefore, all kernels behave like a distance measure. Just like a good distance measure is crucial to a classifier, the choice of a good kernel can affect a classifier significantly. This is the reason why a kernel-based SVM is always better than an SVM just using the simple dot product.

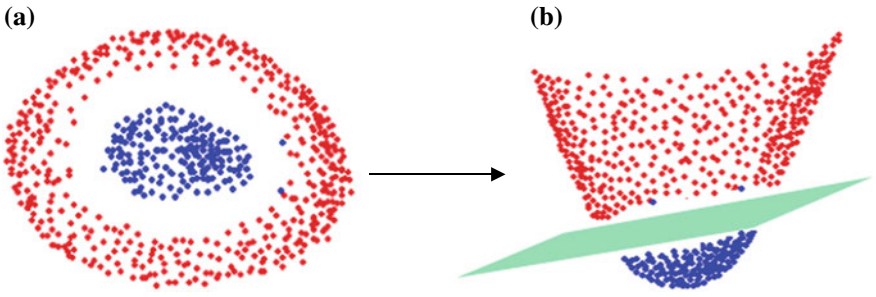


Fig. 8.5 Mapping of nonlinear data to linear data in higher dimensional space. **a** An original nonlinear data in 2D space; **b** transformed data in 3D space using a quadratic mapping function φ

Although the use of kernel gives us the advantage to do nonlinear classification, the explicit mapping from a lower dimensional space to a higher dimensional space is undesirable and can be expensive in terms of computation, given the fact that a feature vector usually has high dimension. Furthermore, data representation using very high dimension in Hilbert space is inefficient too.

Fortunately, the mapping does not need to be done explicitly. So long it is a kernel, it *implicitly* maps a data to one in another space which is linearly separable. Put the other way, a kernel is just a dot product (*implicit*) regardless the space where the dot product is done, and according to (8.32), a kernel SVM is just a weighted nearest neighbors classifier by which a data can always be separated nonlinearly. Therefore, all we need to do for a kernel SVM is just to replace the dot product with a kernel. This is called the *kernel trick*.

To further improve the efficiency, in practice, an $N \times N$ kernel (or Gram) matrix is precomputed for a dataset of N elements before the actual learning, so that there is no need to recompute the dot products at every iteration of the optimization. A kernel matrix K has the following properties:

- K is a positive definite matrix
- $K(i,j) = K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \langle \Phi(\mathbf{x}_i), \Phi\left(\mathbf{x}_j\right) \rangle$ (implicit dot product in higher dimension space)
- K is symmetric, or $K(i, j) = K(j, i)$
- $K(i, j)$ measures the similarity between ith and jth training samples in feature space

$$K = \begin{array}{|c|c|c|c|c|} \hline K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & K(\mathbf{x}_1, \mathbf{x}_3) & \ldots\ldots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \hline K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & K(\mathbf{x}_2, \mathbf{x}_3) & \ldots\ldots & K(\mathbf{x}_2, \mathbf{x}_N) \\ \hline K(\mathbf{x}_3, \mathbf{x}_1) & K(\mathbf{x}_3, \mathbf{x}_2) & K(\mathbf{x}_3, \mathbf{x}_3) & \ldots\ldots & K(\mathbf{x}_3, \mathbf{x}_N) \\ \hline \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \hline K(\mathbf{x}_N, \mathbf{x}_1) & K(\mathbf{x}_N, \mathbf{x}_2) & K(\mathbf{x}_N, \mathbf{x}_3) & \ldots\ldots & K(\mathbf{x}_N, \mathbf{x}_N) \\ \hline \end{array}$$

8.3.5 The Pyramid Match Kernel

A well-designed kernel is crucial to an SVM classifier. Conventional kernel design is independent of the feature itself. However, the selection of a kernel for a

particular type of features is difficult because there is no natural connection between a feature and a kernel. Consequently, the selection of kernel for an SVM classifier is often arbitrary or empirical at best. The Pyramid Match Kernel or PMK [5] is a method to design a kernel which matches the specific type of image features.

The idea is to extract a pyramid histogram feature at different level of resolutions and build a kernel using a weighted sum of histogram intersections. The idea of the PMK is described in details in the following:

- Start with image X itself as level 0 and the total number of levels is L.
- Divide image into grids at different levels of resolutions. The grid at level l has a total of 2^l x $2^l = 4^l$ cells, with 2^l cells along each dimension.
- A histogram is computed for each block at each level of resolutions.
- Histograms at each level l are given a different weight.
- The weighted histograms from all levels are concatenated as the pyramid histogram of the image.
- A kernel of weighted histogram intersection is built for the SVM.

$$K(X, Y) = \sum_{l=1}^{L} \alpha_l k^l(X_m, Y_m) \tag{8.46}$$

where X_m and Y_m are two weighted pyramid histograms and k is the histogram intersection.

- The idea is illustrated in Fig. 8.6 [5].

Let X^l and Y^l stand for the histograms of X and Y at level l, then the number of matches at this level is given by the histogram intersection

$$k^l = \sum_{i=1}^{4^l} \min[X^l(i), Y^l(i)] \tag{8.47}$$

where $l = 0, 1, 2, \ldots, L$. k^l at lower levels represent global features while k^l at higher levels represent local features. Since global features can cause more confusion than local features, global features should be given less weights than local features. Therefore, the weight given to level l is set to $1/2^{L-l}$, which is inversely proportional to the block width at that level. Since the total weights must sum to 1, the combined matching result between two images is given in (8.48).

$$K(X, Y) = \frac{1}{2^L} k^0 + \sum_{l=1}^{L} \frac{1}{2^{L-l+1}} k^l \tag{8.48}$$

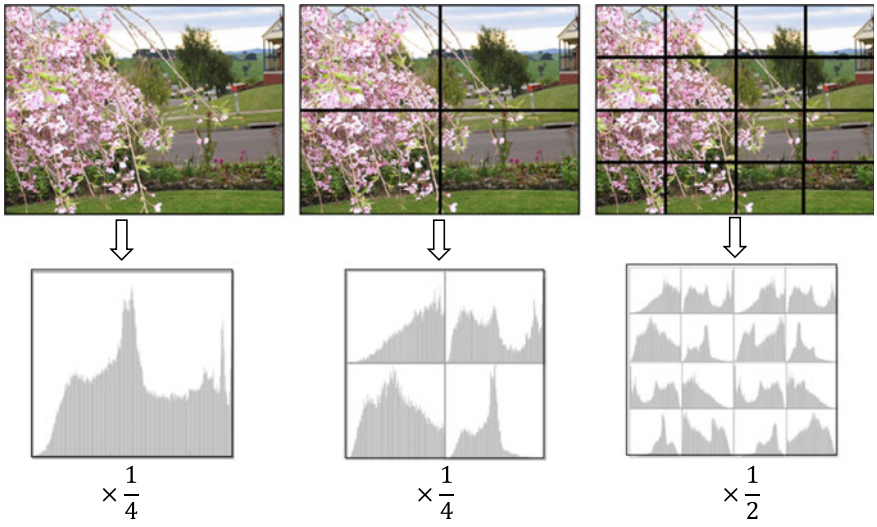


Fig. 8.6 Computation of pyramid match kernel. An image is divided into three levels of grids. At each level of the grid, a histogram is computed for each block of the grid. Histograms at each level are given a weight and the weighted histograms are then concatenated as a feature vector

The next is to prove (8.48) is a kernel. Because a linear combination of kernels is also a kernel, we just need to prove each histogram intersection k^l is a kernel.

Let X_m and Y_m be the histograms of two images or image blocks X and Y. Each image has N pixels. We can then represent X_m and Y_m as two N x m dimensional binary vectors [6].

$$X_m = (\overbrace{1, 1, \ldots, 1}^{x_1}, \underbrace{0, 0, \ldots, 0}_{N-x_1}; \overbrace{1, 1, \ldots, 1}^{x_2}, \underbrace{0, 0, \ldots, 0}_{N-x_2}; \ldots; \overbrace{1, 1, \ldots, 1}^{x_m}, \underbrace{0, 0, \ldots, 0}_{N-x_m}) \quad (8.49)$$

$$Y_m = (\overbrace{1, 1, \ldots, 1}^{y_1}, \underbrace{0, 0, \ldots, 0}_{N-y_1}; \overbrace{1, 1, \ldots, 1}^{y_2}, \underbrace{0, 0, \ldots, 0}_{N-y_2}; \ldots; \overbrace{1, 1, \ldots, 1}^{y_m}, \underbrace{0, 0, \ldots, 0}_{N-y_m}) \quad (8.50)$$

With the above representation, the histogram intersection of X_m and Y_m is given as the dot product of the two histograms:

$$k(X_m, Y_m) = X_m \cdot Y_m \quad (8.51)$$

Therefore, k^l is a kernel and as a result, $K(X, Y)$ in (8.48) is also a kernel.

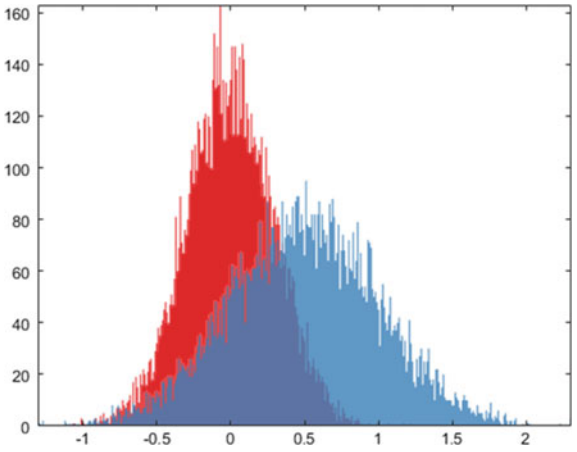


Fig. 8.7 Histogram intersection of two normal distributions

A histogram is a statistical feature, it captures the feature distribution in an image or an image block. Histogram intersection tells how much area two distributions share, the more area they share, the more similar the two distributions are. Figure 8.7 shows an example of histogram intersection. The shared region is about 33% of the two histograms, therefore, the similarity between the two histograms is about 33%.

8.3.6 Discussions

Kernel-based support vector machine is essentially a training-based nearest neighbor classifier. The use of dot product transforms the support vector machine into a nonlinear nearest neighbor classifier. Traditional nearest neighbor has two limitations, the determining of k and it does not support training. However, if the training set is sufficiently large, both limitations can be overcome. First, the k can be determined empirically. The second limitation can be overcome by determining the class boundary with a piecewise linear approximation. For example, the class boundary of the following data can be approximated by 5 hyperplanes, which can then be used to classify new data (Fig. 8.8).

Although the piecewise linear boundary given by the K-NN is not optimal as the boundary provided by the kernel-based SVM, in terms of classification, the effectiveness of the two classifiers can be comparable. However, it would not be as efficient as SVM.

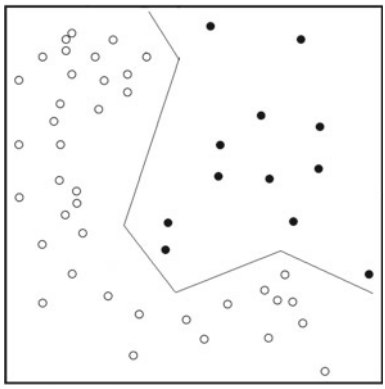


Fig. 8.8 Approximation of class boundary using piecewise hyperplanes

8.4 Fusion of SVMs

8.4.1 Fusion of Binary SVMs

An SVM is essentially a binary classifier. However, automatic image classification and annotation needs a multi-class classifier. The most common approach is to train a separate SVM for each concept c and each SVM generates a decision value $d_c(\mathbf{x})$. During the testing phase, the decisions from all classifiers are fused to obtain the final class label of a test image. Figure 8.9 demonstrates this two level fusion process [7, 8]. The first level consists of multiple binary classifiers and the second level fuses the decisions from the first level classifiers.

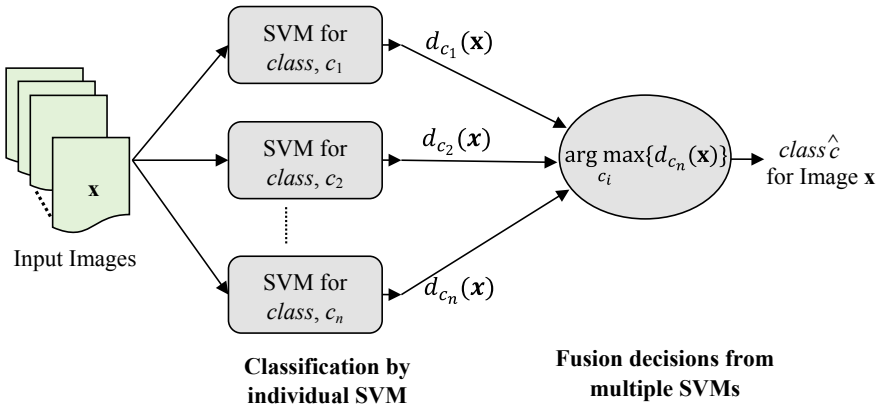


Fig. 8.9 A fusion of binary SVM classifiers

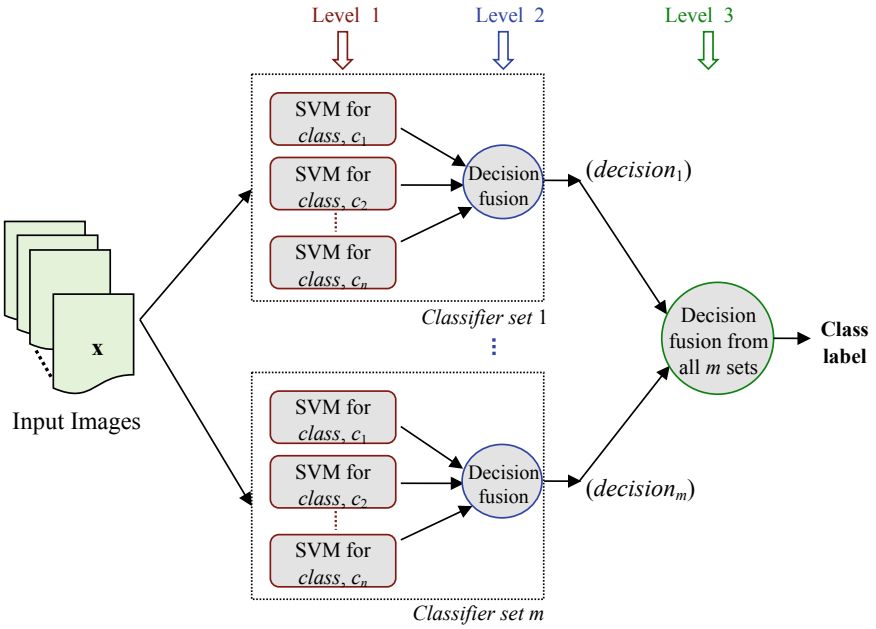


Fig. 8.10 A 3 levels fusion of SVMs

8.4.2 Multilevel Fusion of SVMs

The above approach can be regarded as a base level fusion, it works well for a small number of concepts. The quality of classification degrades with the increase of the number of concepts due to the increase of the noise and class imbalance in the training data. To be more robust, *multiple sets* of base level fusion of SVMs can be merged to make a more powerful fusion as shown in Fig. 8.10 [7, 8]. Each set of SVMs in level 1 and level 2 is similar to the base level fusion shown in 8.9 and independently classifies an input image, the final decision is fused from the decisions of all the individual sets at level 3.

The key advantage of using multiple sets of SVMs is to learn a more accurate and robust classifier using different types of SVMs, such as classification SVMs, regression SVMs, SVMs with/without soft margins, etc.

8.4.3 Fusion of SVMs with Different Features

Fusion of classifiers can also be done with combination of different types of features. For example, both global and local features can be used to train two different sets of SVMs at level 1 as shown in Fig. 8.11 [7, 8]. The results from the two sets of SVMs are then fused in two steps. First, decisions of each concept made by each set

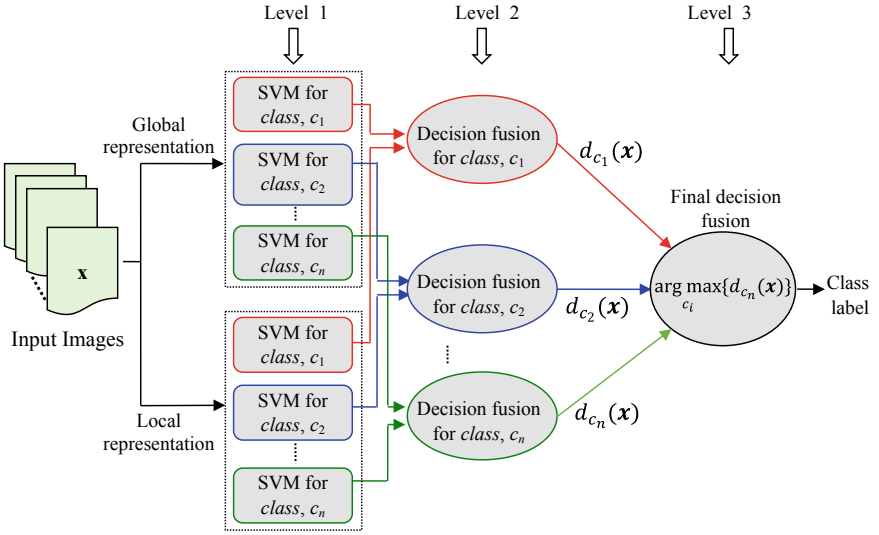


Fig. 8.11 A 3 levels class-by-class fusion of SVMs with both global and local features

of SVM are fused at level 2. Next, the final decision is made using a maximization at level 3.

Although the fusion methods discussed in this section are shown as fusion of SVMs, they can also be applied to fusion of different types of classifiers such as Bayesian, ANN, DT, etc.

8.5 Summary

SVM is basically a supervised linear classifier which divides a dataset into two classes with a hyperplane in data space. However, different from an ordinary linear classifier, it offers an optimal hyperplane which separates two classes of data with maximum margin between them. The data points make the hyperplane are called the support vectors. SVM works by repeat guessing with candidate hyperplanes until the optimal hyperplane is found.

The biggest progress of SVM is the kernel-based SVM which achieves non-linearity without the use of networking like ANN. Nonlinearity of SVM is achieved through transforming data into higher space so that they can be separated linearly. Due to the kernel trick, this transformation is even unnecessary so long as the distance is a kernel. This makes SVM is very efficient compared with ANN.

However, SVM is essentially a binary classifier or non-probabilistic classifier. This makes it less robust than other probabilistic classifiers such as Bayesian classifiers and DT. In addition, a multi-class SVM needs to be achieved through fusion or assembly.