where its inputs are

a matrix X' containing the points of the data set (each row is a point),

the class labels of the data points (y'),

the type of kernel function to be used (in our case $'linear'$),

two kernel parameters $kpar1$ and $kpar2$ (in the linear case both are set to 0),

the parameter C,

the parameter tol,

the maximum number of iteration steps of the algorithm,

a threshold eps (a very small number, typically on the order of 10^{-10}) used in the comparison of two numbers (if their difference is less than this threshold, they are considered equal to each other),

the optimization method to be used ($0 \rightarrow$Platt, $1 \rightarrow$Keerthi modification 1, $2 \rightarrow$Keerthi modification 2),[1]

$alpha$ is a vector containing the Lagrange multipliers corresponding to the training points,

$w0$ is the threshold value,

w is the vector containing the hyperplane parameters, returned by the algorithm.

The parameter tol is a scalar that controls the accuracy of the obtained solution [Theo 09, Section 3.7.2]. The larger the value of tol is, the farther from the solution the algorithm may stop. A typical value for tol is 0.001.

Example 2.4.1. In the 2-dimensional space, we are given two equiprobable classes, which follow Gaussian distributions with means $m_1 = [0,\ 0]^T$ and $m_2 = [1.2,\ 1.2]^T$ and covariance matrices $S_1 = S_2 = 0.2I$, where I is the 2×2 identity matrix.

1. Generate and plot a data set X_1 containing 200 points from each class (400 points total), to be used for training (use the value of 50 as seed for the built-in MATLAB *randn* function). Generate another data set X_2 containing 200 points from each class, to be used for testing (use the value of 100 as seed for the built-in MATLAB *randn* function).
2. Based on X_1, run Platt's algorithm to generate six SVM classifiers that separate the two classes, using $C = 0.1,\ 0.2,\ 0.5,\ 1,\ 2,\ 20$. Set $tol = 0.001$.
   - **(a)** Compute the classification error on the training and test sets.
   - **(b)** Count the support vectors.
   - **(c)** Compute the margin ($2/||w||$).
   - **(d)** Plot the classifier as well as the margin lines.

Solution. Do the following:

Step 1. To generate the data set X_1, type

[1] More details can be found in the comments of the function.

```
randn('seed',50)
m=[0 0; 1.2 1.2]'; % mean vectors
S=0.2*eye(2); % covariance matrix
points_per_class=[200 200];
X1=mvnrnd(m(:,1),S,points_per_class(1))';
X1=[X1 mvnrnd(m(:,2),S,points_per_class(2))'];
y1=[ones(1,points_per_class(1))...
   -ones(1,points_per_class(2))];
```

To plot the data set X_1, type

```
figure(1), plot(X1(1,y1==1),X1(2,y1==1),'r.',...
X1(1,y1==-1),X1(2,y1==-1),'bo')
```

Notice that the classes overlap.

To generate X_2 repeat the code, replacing the first line with

```
randn('seed',100)
```

Step 2. To generate the required SVM classifier for $C = 0.1$, use the SMO2 function, typing

```
kernel='linear';
kpar1=0;
kpar2=0;
C=0.1;
tol=0.001;
steps=100000;
eps=10^(-10);
method=0;
[alpha, w0, w, evals, stp, glob] = SMO2(X1', y1',...
 kernel, kpar1, kpar2, C, tol, steps, eps, method)
```

The other classifiers are generated similarly.

(a) To compute the classification error on the training set, X_1, type

```
Pe_tr=sum((2*(w*X1-w0>0)-1).*y1<0)/length(y1)
```

The classification error on the test set, X_2, is computed similarly.

(b) To plot the classifier hyperplane as well as the margin lines, use the function *svcplot_book* by typing[2] the following:

[2]To plot the results of more than one experiment, change the value of *figt*4, which is the number of the figure where the plot will take place.

Table 2.4 Results for Various Values of C in Example 2.4.1

	$C = 0.1$	$C = 0.2$	$C = 0.5$	$C = 1$	$C = 2$	$C = 20$
No. support vectors	82	61	44	37	31	25
Training error	2.25%	2.00%	2.00%	2.25%	3.25%	2.50%
Test error	3.25%	3.00%	3.25%	3.25%	3.50%	3.50%
Margin	0.9410	0.8219	0.7085	0.6319	0.6047	0.3573

```
global figt4
figt4=2;
svcplot_book(X1',y1',kernel,kpar1,kpar2,alpha,-w0)
```

(c) To count the support vectors, type

```
sup_vec=sum(alpha>0)
```

(d) To compute the margin, type

```
marg=2/sqrt(sum(w.^2))
```

The results of these experiments are shown in Table 2.4. It is readily seen that the margin of the solution increases as C decreases. This is natural because decreasing C makes the "margin term" in Eq. (2.8) more significant. For the problem at hand, the best performance (minimum test error) is obtained for $C = 0.2$. ■

Exercise 2.4.1

Repeat Example 2.4.1, now with the covariance matrices of the Gaussian distributions $S_1 = S_2 = 0.3I$. Comment on the results.

2.4.1 Multiclass Generalizations

In the previous section, we dealt with the SVM for the 2-class case. Although mathematical generalizations for the multiclass case are available, the task tends to become rather complex. When more than two classes are present, there are several different approaches that evolve around the 2-class case. In this section, we focus on one of these methods, known as *one-against-all* (for more details on the multiclass problem see [Theo 09, Section 3.7.3]). These techniques are not tailored to the SVM. They are general and can be used with any classifier developed for the 2-class problem. Moreover, they are not just pedagogical toys, but are actually widely used.

According to the one-against-all method, c classifiers have to be designed. Each one of them is designed to separate one class from the rest (recall that this was the problem solved in Section 2.3.1, based on the LS criterion). For the SVM paradigm, we have to design c linear classifiers:

$$w_j^T x + w_{j0}, \quad j = 1, 2, \ldots, c$$

For example, to design classifier w_1, we consider the training data of all classes other than ω_i to form the second class. Obviously, unless an error is committed we expect all points from class ω_1 to result in

$$w_1^T x + w_{10} > 0$$

and the data from the rest of the classes to result in negative outcomes. A x is classified in ω_i if

$$w_i^T x + w_{i0} > w_j^T x + w_{j0}, \quad \forall i \neq j$$

Remark

- A drawback of one-against-all is that after the training there are regions in the space, where no training data lie, for which more than one hyperplane gives a positive value or all of them result in negative values [Theo 09, Section 3.7.3, Problem 3.15].

Example 2.4.2

1. Generate and plot two data sets X_1 (training) and X_2 (test) using the prescription of Example 2.3.3, except that now each set consists of 120 data points.
2. Based on X_1, estimate the parameter vectors w_1, w_2, w_3 of the three linear discriminant functions using the first modification of Platt's algorithm [Keer 01] (SVM classifiers). Estimate the classification error rate based on X_2.

Solution. Proceed as follows:

Step 1. To generate X_1 and X_2, work as in Example 2.3.3. To plot X_1, type

```
figure(1), plot3(X1(1,z1(1,:)==1),X1(2,z1(1,:)==1),...
X1(3,z1(1,:)==1),'r.',X1(1,z1(2,:)==1),X1(2,z1(2,:)==1),...
X1(3,z1(2,:)==1),'gx',X1(1,z1(3,:)==1),X1(2,z1(3,:)==1),...
X1(3,z1(3,:)==1),'bo')
```

Step 2. In this case, matrices z_1 and z_2 are created in the same spirit as in Example 2.3.3, but now the 0 elements are replaced by -1. Specifically, type

```
z1=-ones(c,N1);
for i=1:N1
    z1(y1(i),i)=1;
end
```

where c is the number of classes and N_1 is the number of training vectors. Similarly obtain z_2.

To compute the SVM classifiers, type

```
kernel='linear';  %SVM parameter defintion
kpar1=0;
kpar2=0;
C=20;
```

```
tol=0.001;
steps=100000;
eps=10^(-10);
method=1;
for i=1:c
   [alpha(:,i), w0(i), w(i,:), evals, stp, glob] =...
   SMO2(X1', z1(i,:)', kernel, kpar1, kpar2, C,...
   tol, steps, eps, method)
   marg(i)=2/sqrt(sum(w(i,:).^2))  % Margin
   %Counting the number of support vectors
   sup_vec(i)=sum(alpha(:,i)>0)
end
```

To estimate the classification error rate based on X_2, type

```
[vali,class_est]=max(w*X2-w0'*ones(1,N2));
err_svm=sum(class_est~=y2)/N2
```

The classification error in this case turns out to be 5.00%. For comparison, we mention that the Bayesian classification error is 3.33%. (Explain why the latter value is different from that extracted in Example 2.3.3.)

2.5 SVM: THE NONLINEAR CASE

To employ the SVM technique for solving a nonlinear classification task, we adopt the philosophy of mapping the feature vectors in a higher-dimensional space, where we expect, with high probability, the classes to be linearly separable. This is guaranteed by the celebrated Cover's theorem [Theo 09, Section 4.13]. The mapping is as follows:

$$x \mapsto \phi(x) \in H$$

where the dimension of H is higher than $\mathcal{R}^l$ and, depending on the choice of the (nonlinear) $\phi(\cdot)$, can even be infinite. Moreover, if the mapping function is carefully chosen from a known family of functions that have specific desirable properties, the inner product between the images $(\phi(x_1), \phi(x_2))$ of two points x_1, x_2 can be written as

$$< \phi(x_1), \phi(x_2) >= k(x_1, x_2)$$

where $< \cdot, \cdot >$ denotes the inner product operation in H and $k(\cdot, \cdot)$ is a function known as *kernel* function. That is, inner products in the high-dimensional space can be performed in terms of the associated kernel function acting in the original low-dimensional space. The space H associated with $k(\cdot, \cdot)$ is known as a reproducing kernel Hilbert space (RKHS) (for more formal definitions see [Theo 09, Section 4.18] and references therein).

A notable characteristic of the SVM optimization is that all operations can be cast in terms of inner products. Thus, to solve a linear problem in the high-dimensional space (after the mapping), all we have to do is replace the inner products with the corresponding kernel evaluations. Typical examples of kernel functions are (a) the *radial basis function* (RBF), defined as

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{\sigma^2}\right)$$

where σ is a user-defined parameter that specifies the rate of decay of $k(x,y)$ toward zero, as y moves away from x and (b) the *polynomial function*, defined as

$$k(x,y) = (x^T y + \beta)^n$$

where β and n are user-defined parameters.

Note that solving a linear problem in the high-dimensional space is equivalent to solving a nonlinear problem in the original space. This is easily verified. As in Eq. (2.12), the hyperplane computed by the SVM method in the high-dimensional space H is

$$w = \sum_{i=1}^{N} \lambda_i y_i \phi(x_i) \tag{2.13}$$

Given a x, we first map it to $\phi(x)$ and then test whether the following is less than or greater than zero:

$$\begin{aligned} g(x) \equiv < w, \phi(x) > + w_0 &= \sum_{i=1}^{N} \lambda_i y_i < \phi(x), \phi(x_i) > + w_0 \\ &= \sum_{i=1}^{N} \lambda_i y_i k(x, x_i) + w_0 \end{aligned} \tag{2.14}$$

From the previous relation, it becomes clear that the explicit form of the mapping function $\phi(\cdot)$ is not required; all we have to know is the kernel function since data appear only in inner products. Observe that the resulting discriminant function, $g(x)$, is nonlinear because of the nonlinearity of the kernel function.

To generate a nonlinear SVM classifier, the *SMO*2 MATLAB function, discussed in Section 2.4, may be used. The input argument *kernel* takes the values $'poly'$ for the polynomial kernel or $'rbf'$ for the RBF kernel. In the former case, *kpar*1 and *kpar*2 correspond to the β and n parameters, respectively; in the latter case, *kpar*1 corresponds to the σ parameter.

Example 2.5.1

1. Generate a 2-dimensional data set X_1 (training set) as follows. Select $N=150$ data points in the 2-dimensional $[-5,\ 5] \times [-5,\ 5]$ region according to the uniform distribution (set the seed for the *rand* function equal to 0). Assign a point $x = [x(1),\ x(2)]^T$ to the class $+1$ (-1) according to the rule $0.05(x^3(1) + x^2(1) + x(1) + 1) > (<) x(2)$. (Clearly, the two classes are nonlinearly separable;

in fact, they are separated by the curve associated with the equation $0.05(x^3(1)+x^2(1)+x(1)+1) = x(2)$.) Plot the points in X_1. Generate an additional data set X_2 (test set) using the same prescription as for X_1 (set the seed for the *rand* function equal to 100).

2. Design a linear SVM classifier using the first modification of Platt's algorithm with parameters $C = 2$ and $tol = 0.001$. Compute the training and test errors and count the number of support vectors.
3. Generate a nonlinear SVM classifier using the radial basis kernel functions for $\sigma = 0.1$ and 2. Use the first modification of Platt's algorithm, with $C = 2$ and $tol = 0.001$. Compute the training and test error rates and count the number of support vectors. Plot the decision regions defined by the classifier.
4. Repeat step 3 using the polynomial kernel functions $(x^T y + \beta)^n$ for $(n, \beta) = (5, 0)$ and $(3, 1)$. Draw conclusions.
5. Design the SVM classifiers using the radial basis kernel function with $\sigma = 1.5$ and using the polynomial kernel function with $n = 3$ and $\beta = 1$. Use the first modification of Platt's algorithm with $tol = 0.001$ for $C = 0.2, 20, 200$.

Solution. Take the following steps:

Step 1. To generate the data set X_1 and the vector y_1 containing the class labels for the vectors in X_1, type

```
l=2;   % Dimensionality
N=150; % Number of vectors
% Generating the training set
rand('seed',0)
X1=10*rand(l,N)-5;
for i=1:N
    t=0.05*(X1(1,i)^3+X1(1,i)^2+X1(1,i)+1);
    if(t>X1(2,i))
        y1(i)=1;
    else
        y1(i)=-1;
    end
end
```

To plot the data set X_1 (see Figure 2.5(a)), type

```
figure(1), plot(X1(1,y1==1),X1(2,y1==1),'r+',...
X1(1,y1==-1),X1(2,y1==-1),'bo')
figure(1), axis equal
```

To generate X_2 work as in the case of X_1.

Step 2. To generate a linear SVM classifier based on X_1 with $C = 2$ and $tol = 0.001$, type

```
kernel='linear';
kpar1=0;
```

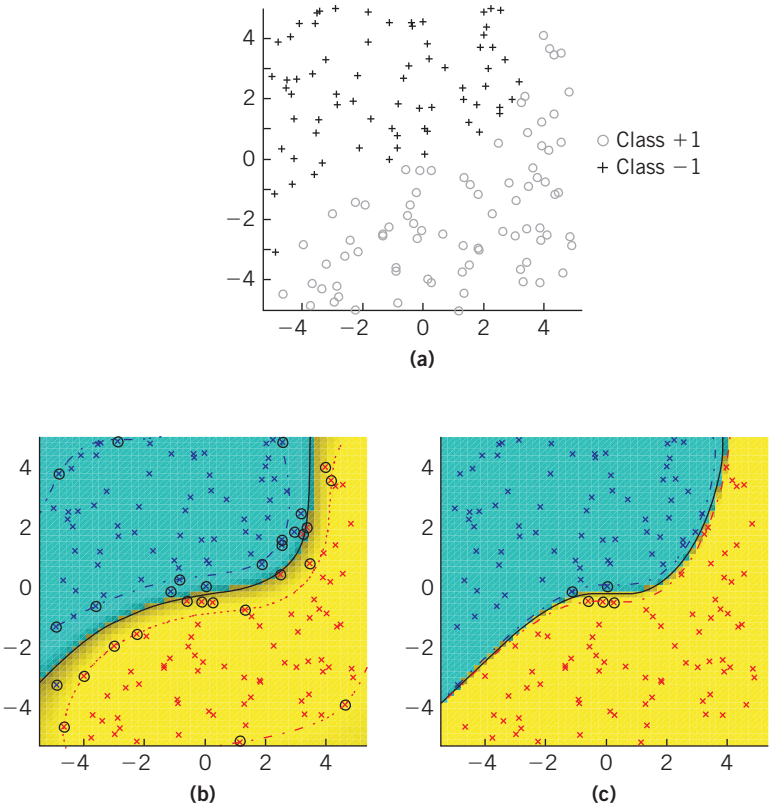

FIGURE 2.5

(a) Training set for Example 2.5.1. (b) Decision curve associated with the classifier using the radial basis kernel function ($\sigma = 2$) and $C = 2$. (c) Decision curve realized by the classifier using the polynomial kernel function ($\beta = 1$, $n = 3$) and $C = 2$. Observe that different kernels result in different decision surfaces. Support vectors are encircled. Dotted lines indicate the margin.

```
kpar2=0;
C=2;
tol=0.001;
steps=100000;
eps=10^(-10);
method=1;
[alpha, w0, w, evals, stp, glob] = SMO2(X1', y1', ...
kernel, kpar1, kpar2, C, tol, steps, eps, method)
```

To compute the training error, type

```
Pe1=sum((2*(w*X1-w0>0)-1).*y1<0)/length(y1)
```

Similarly compute the test error. To count the number of support vectors, type

```
sup_vec=sum(alpha>0)
```

Step 3. To generate a nonlinear SVM classifier employing the radial basis kernel function with $\sigma = 0.1$, work as in step 2 but now set

```
kernel='rbf';
kpar1=0.1;
kpar2=0;
```

Work similarly for the other value of σ. To compute the training error, the process is as follows:

- The support vectors are stacked in a matrix, while their Lagrange multipliers and their class labels are stacked to vectors. Type

```
X_sup=X1(:,alpha'~=0);
alpha_sup=alpha(alpha~=0)';
y_sup=y1(alpha~=0);
```

- Each vector is classified separately. Type

```
for i=1:N
    t=sum((alpha_sup.*y_sup).*...
    CalcKernel(X_sup',X1(:,i)',kernel,kpar1,kpar2)')-w0;
    if(t>0)
       out_train(i)=1;
    else
       out_train(i)=-1;
    end
end
```

- Compute the training error as

```
Pe1=sum(out_train.*y1<0)/length(y1)
```

The test error is computed in a similar manner. To count the number of support vectors, type

```
sup_vec=sum(alpha>0)
```

To plot the decision regions formed by the classifier (see Figure 2.5(b)), type

```
global figt4=3;
svcplot_book(X1',y1',kernel,kpar1,kpar2,alpha,-w0)
```

Table 2.5 Results for the SVM Classifiers Designed in Steps 2, 3, and 4 of Example 2.5.1

	Training Error	Testing Error	No. Support Vectors
Linear	7.33%	7.33%	26
RBF(0.1)	0.00%	32.67%	150
RBF(2)	1.33%	3.33%	30
poly(5,0)	—	—	—
poly(3,1)	0.00%	2.67%	8

Note: *RBF(a) denotes the SVM classifier corresponding to the radial basis kernel function with $\sigma = a$; poly (n, β) denotes the SVM classifer with the polynomial kernel function of the form $(x^T y + \beta)^n$. The algorithm does not converge for the case poly(5,0).*

Step 4. To generate a nonlinear SVM classifier with the polynomial kernel function using $n = 3$ and $\beta = 1$, work as before but now set

```
kernel='poly';
kpar1=1;
kpar2=3;
```

The training and test errors as well as the number of support vectors are computed as in the previous step (see also Figure 2.5(c)). Work similarly for the other combinations of β and n. The results obtained by the different SVM classifiers are summarized in Table 2.5. From this table, the following conclusions can be drawn.

First, the linear classifier performs worse than the nonlinear SVM classifiers. This is expected since the involved classes in the problem at hand are nonlinearly separable. Second, the choice of parameters for the kernel functions used in the nonlinear SVM classifiers significantly affect the performance of the classifier; parameters should be chosen carefully, after extensive experimentation (see, for example, *RBF*(0.1) and try *RBF*(5)). Finally, low training error does not necessarily guarantee low test error; note that the latter should be considered in evaluating performance.

Step 5. To generate the corresponding SVM classifiers, work as before but set C to $0.2, 20, 200$. ■

Example 2.5.2

1. Generate a 2-dimensional data set X_1 (training set) as follows. Consider the nine squares $[i, i+1] \times [j, j+1]$, $i = 0, 1, 2$, $j = 0, 1, 2$ and draw randomly from each one 30 uniformly distributed points. The points that stem from squares for which $i + j$ is even (odd) are assigned to class $+1$ (-1) (reminiscent of the white and black squares on a chessboard). Plot the data set and generate an additional data set X_2 (test set) following the prescription used for X_1 (as in Example 2.5.1, set the seed for *rand* at 0 for X_1 and 100 for X_2).
2. **(a)** Design a linear SVM classifier, using the first modification of Platt's algorithm, with $C = 200$ and $tol = 0.001$. Compute the training and test errors and count the number of support vectors.

(b) Employ the previous algorithm to design nonlinear SVM classifiers, with radial basis kernel functions, for $C = 0.2, 2, 20, 200, 2000, 20,000$. Use $\sigma = 1, 1.5, 2, 5$. Compute the training and test errors and count the number of support vectors.

(c) Repeat for polynomial kernel functions, using $n = 3, 5$ and $\beta = 1$.

3. Draw conclusions.

Solution. Do the following:

Step 1. To generate the data set X_1, type

```
l=2; %Dimensionality
poi_per_square=30; %Points per square
N=9*poi_per_square; %Total no. of points
%Generating the training set
rand('seed',0)
X1=[];
y1=[];
for i=0:2
    for j=0:2
        X1=[X1 rand(l,poi_per_square)+...
        [i j]'*ones(1,poi_per_square)];
        if(mod(i+j,2)==0)
            y1=[y1 ones(1,poi_per_square)];
        else
            y1=[y1 -ones(1,poi_per_square)];
        end
    end
end
```

To plot X_1 work as in Example 2.5.1 (see Figure 2.6(a)). To generate X_2, work as in the case of X_1.

Step 2. For all these experiments, work as in Example 2.5.1 (see also Figure 2.6(b)).

Step 3. From the results obtained, the following conclusions can be drawn.

- First, as expected, the linear classifier is inadequate to handle this problem (the resulting training and test errors are greater than 40%). The same holds true for the SVMs with polynomial kernels (>20% test error for all combinations of parameters). This has to do with the specific nature of this example.
- Second, the SVM classifiers with radial basis kernel functions give very good results for specific choices of the parameter σ. In Table 2.6, the best results for the radial basis kernel SVM classifiers are presented (all of them have been obtained for $C = 2000$). From this table, once more it can be verified that very low values of σ lead to very poor generalization (k training error, high test error). An intuitive explanation is that very small σ's cause the

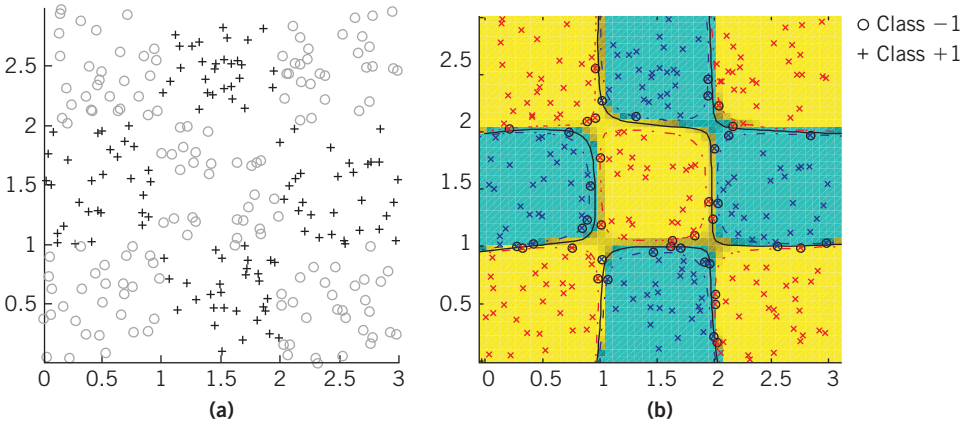

FIGURE 2.6

(a) Training set for Example 2.5.2. (b) Decision curve of the SVM classifier, with radial basis kernel functions ($\sigma = 1$) and $C = 2000$. Support vectors are encircled; dotted lines indicate the margin.

Table 2.6 Results for the SVM Classifiers Obtained for $\sigma = 0.1, 1, 1.5, 2, 5$ in Example 2.5.2

	Training Error	Test Error	No. Support Vectors
RBF(0.1)	0.00%	10.00%	216
RBF(1)	1.85%	3.70%	36
RBF(1.5)	5.56%	7.04%	74
RBF(2)	8.15%	8.52%	128
RBF(5)	35.56%	31.48%	216

Note: *RBF(a) denotes the SVM classifier with radial basis kernel functions with* $\sigma = a$.

$k(x, x_i)$'s to drop rapidly toward zero around each $x_i \in X_1$, which leads to an increase in the number of support vectors (since many points are required to "cover" the space where the data lie). For each training point, x_i, the summation in the classification rule in Eq. (2.14) is mostly affected by the corresponding term $\lambda_i y_i k(x_i, x_i)$. This explains the low training error. In contrast, since the $k(x, x_i)$'s do not sufficiently "cover" the space away from the training points, the summation in Eq. (2.14) may be almost zero for several test points (from both classes) and the respective labels cannot be accurately predicted.

On the other hand, large values of σ (in our example $\sigma = 5$ is considered as such) lead to poor results for both the training and test sets. An intuitive explanation for this is that, when σ is large, all $k(x, x_i)$'s remain almost constant in the area where the data points lie. This also leads to an increase in support vectors (since almost all points are of equal importance) with

almost equal values for the λ_i's. Thus, the summation in the classification rule in Eq. (2.14) exhibits low variation for the various x's (from both the training and the test set), which leads to reduced discrimination capability. Values between these two ends lead to more acceptable results, with the best performance being achieved for $\sigma = 1$ in the present example. The previous discussion makes clear the importance of choosing, for each problem, the right values for the involved parameters.

- Third, for fixed kernel parameters and C varying from 0.2 to 20,000, the number of SVs (in general) decreases, as expected.

2.6 THE KERNEL PERCEPTRON ALGORITHM

As stated before, all operations in obtaining the SVM classifier can be cast in the form of inner products. This is also possible with a number of other algorithms; the perceptron algorithm is a notable example. The "inner product formulation" can be exploited by following the so-called kernel trick.

Kernel trick: Substitute each inner product $x^T y$ with the kernel function $k(x, y)$. This is equivalent to solving the problem in some high-dimensional space where the inner product is defined in terms of the respective kernel function. Adopting this trick, a linear task in the high-dimensional space is equivalent to a nonlinear task in the original feature space, where the training data lie. More on this issue may be found in [Theo 09, Section 4.19].

To call the kernel perceptron algorithm, type

$$[a, iter, count\_misclas] = kernel\_perce(X, y, kernel, kpar1, kpar2, max\_iter)$$

where

X is the matrix whose columns are the data vectors,

y is a vector containing the class labels of the data points,

kernel, *kpar*1, and *kpar*2 are defined as in the *SMO*2 function,

max_iter is the maximum allowable number of iterations that the algorithm can perform,

a is a vector, whose ith coordinate contains the number of times the ith point was misclassified,

iter is the number of iterations performed by the algorithm,

count_misclas is the number of misclassified points.

A given vector x is classified to class $+1$ or class -1 according to whether the following is positive or negative:

$$g(x) = \sum_{i=1}^{N} a_i y_i k(x, x_i) + \sum_{i=1}^{N} a_i y_i$$

Example 2.6.1

1. Consider the data sets X_1 (training set) and X_2 (test set) from Example 2.5.1. Run the kernel perceptron algorithm using X_1 as the training set where the kernel functions are (a) linear, (b) radial basis functions with $\sigma = 0.1, 1, 1.5, 2, 5, 10, 15, 20$, and (c) polynomials of the form $(x^T y + \beta)^n$