Chapter 5 Support Vector Machines

The present form of support vector machines (SVMs) was largely developed by V. Vapnik and co-workers at AT&T Bell Labs in 1990's ([1], [2]) and has since been a subject of intensive studies. As a result, numerous variants of SVM are now available, with many of them demonstrating competitive performance in real-world applications. The material presented below is largely based on [18] and [11].

5.1 Support Vector Machines

A. *Support Vector Machines for Classification*

Given linearly separable training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)$ for $n = 1, \ldots, N\}$ with $\boldsymbol{x}_n$ belonging to an input space $\mathcal{X}$ of dimension d, there exists a decision boundary $\boldsymbol{w}^T\boldsymbol{x} = 0$ (where $\boldsymbol{x}$ assumes the form of $[1 \;\; x_1 \ldots x_d]^T$) that separates the points in $\mathcal{D}$ with $y_n = 1$ from those points with $y_n = -1$. Typically such separating weight vector is *not* unique, thus one may define a *margin* as the smallest distance between the decision boundary and the data points on both sides of the boundary. Among other things, a separating $\boldsymbol{w}$ with bigger margin offers more robust performance against data's uncertainty. A question naturally arises is how to find a separating $\boldsymbol{w}$ with largest margin? The SVM exactly addresses this question.

For clarity of description, in what follows we decide to denote the weight vector as

$$\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}$$

and $\boldsymbol{x} = [x_1 \ldots x_d]^T$ so that the decision boundary assumes the form

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0 \tag{5.1}$$

Let $\boldsymbol{x}_n$ be a sample point from $\mathcal{D}$ that is nearest to the decision boundary. Depending on which side of the boundary $\boldsymbol{x}_n$ resides, we have $\boldsymbol{w}^T\boldsymbol{x}_n + b = \pm\delta$ with $\delta > 0$, hence $\left|\boldsymbol{w}^T\boldsymbol{x}_n + b\right| = \delta$. By dividing both sides of the above by δ and renaming $\boldsymbol{w}/\delta$ and b/δ to $\boldsymbol{w}$ and $\boldsymbol{b}$ respectively, we have

$$\left|\boldsymbol{w}^T\boldsymbol{x}_n + b\right| = 1 \tag{5.2}$$

Since the same scaling can be applied to (5.1), the $\boldsymbol{w}$'s and b's in (5.1) and (5.2) are now considered to be the same.

To compute the distance between point $\boldsymbol{x}_n$ and the decision boundary, note that a separating w is orthogonal to the boundary. This is because for any two points $\boldsymbol{x}$ and $\boldsymbol{x}$' on the boundary, (5.1) implies that $\boldsymbol{w}^T(\boldsymbol{x}' - \boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}' + b - (\boldsymbol{w}^T\boldsymbol{x} + b) = 0$. Therefor the distance we look for can be found by projecting vector $\boldsymbol{x}_n - \boldsymbol{x}$ on $\boldsymbol{w}$, namely,

$$\|\hat{\boldsymbol{w}}\| = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \text{ the distance} = \left|\hat{\boldsymbol{w}}^T(\boldsymbol{x}_n - \boldsymbol{x})\right| = \frac{1}{\|\boldsymbol{w}\|}\left|\boldsymbol{w}^T\boldsymbol{x}_n - \boldsymbol{w}^T\boldsymbol{x}\right| = \frac{1}{\|\boldsymbol{w}\|}\left|\boldsymbol{w}^T\boldsymbol{x}_n + b - (\boldsymbol{w}^T\boldsymbol{x} + b)\right| = \frac{1}{\|\boldsymbol{w}\|}$$

Therefore, the a separating $\boldsymbol{w}$ with largest margin can be obtained by solving the constrained problem

$$\begin{aligned} &\text{maximize} && \frac{1}{\|\boldsymbol{w}\|} \\ &\text{subject to:} && \min_{1\le n\le N} |\boldsymbol{w}^T\boldsymbol{x}_n + b| = 1 \end{aligned}$$

which is equivalent (see Prob. 3.4) to

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ &\text{subject to:} && y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \ge 1 \ \text{ for } \ n = 1,\ldots,N \end{aligned} \tag{5.3}$$

Problem (5.3) is a convex quadratic programming (QP) problem. By defining

$$\hat{\boldsymbol{w}} = \begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}, \quad \boldsymbol{H} = \begin{bmatrix} 0 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_N \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} -y_1 & -y_1\boldsymbol{x}_1^T \\ \vdots & \vdots \\ -y_N & -y_N\boldsymbol{x}_N^T \end{bmatrix}, \quad \boldsymbol{e} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Problem (5.3) can be expressed in a standard form

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\hat{\boldsymbol{w}}^T\boldsymbol{H}\hat{\boldsymbol{w}} \\ &\text{subject to:} && \boldsymbol{A}\hat{\boldsymbol{w}} \le -\boldsymbol{e} \end{aligned} \tag{5.4}$$

The Lagrangian of (5.3) (see Sec. 1.2.2) is given by

$$L(\boldsymbol{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N}\mu_n(1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)) \tag{5.5}$$

where vector $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_N \end{bmatrix}^T$ collects N Lagrange multipliers. The KKT conditions (see Sec. 1.2.2.A) for the solution of (5.3) are the N constraints in (5.3) plus

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{n=1}^{N}\mu_n y_n \boldsymbol{x}_n = \boldsymbol{0} \tag{5.6}$$

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N}\mu_n y_n = 0 \tag{5.7}$$

$$\mu_n(1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b)) = 0 \quad \text{for } n = 1, \ldots, N \tag{5.8}$$

$$\mu_n \ge 0 \quad \text{for } n = 1, \ldots, N \tag{5.9}$$

Recall the Lagrange dual (see Sec. 1.2.2.D) of problem (5.3) is given by

$$\begin{aligned} &\text{maximize} && q(\boldsymbol{\mu}) = \inf_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\mu}) \\ &\text{subject to:} && \boldsymbol{\mu} \ge 0 \end{aligned} \tag{5.10}$$

It follows from (5.5) – (5.7) that $q(\boldsymbol{\mu}) = -\frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{d}\cdot\boldsymbol{d}^T)\boldsymbol{\mu} + \boldsymbol{e}^T\boldsymbol{\mu}$ subject to $\sum_{n=1}^{N}\mu_n y_n = 0$, where $\boldsymbol{d} = \begin{bmatrix} y_1\boldsymbol{x}_1 & y_2\boldsymbol{x}_2 & \cdots & y_N\boldsymbol{x}_N \end{bmatrix}^T$. This in conjunction with the positivity constraint on $\boldsymbol{\mu}$ (see (5.10)) leads (5.10) to an explicit convex QP dual problem as

$$\underset{\boldsymbol{\mu}}{\text{minimize}} \quad \frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{d}\cdot\boldsymbol{d}^T)\boldsymbol{\mu} - \boldsymbol{e}^T\boldsymbol{\mu} \qquad (5.11)$$
$$\text{subject to:} \quad \boldsymbol{\mu}^T\boldsymbol{y} = 0, \quad \boldsymbol{\mu} \geq \boldsymbol{0}$$

where $\boldsymbol{y} = [y_1\ y_2\ \ldots\ y_N]^T$. The solution $\boldsymbol{\mu}$ of the dual problem (5.11) is connected to the solution $(\boldsymbol{w}, b)$ of the primal problem (5.3) via (5.6) as

$$\boldsymbol{w} = \sum_{n=1}^{N} \mu_n y_n \boldsymbol{x}_n \qquad (5.12)$$

then via any equation in (5.8) with $\mu_m \neq 0$ as

$$b = y_m - \boldsymbol{w}^T\boldsymbol{x}_m \qquad (5.13)$$

Support Vectors

If we let $S \subseteq \{1,2,...,N\}$ be the set of index with which $\mu_n \neq 0$, (5.12) can be reduced to

$$\boldsymbol{w} = \sum_{n\in S} \mu_n y_n \boldsymbol{x}_n \qquad (5.14)$$

From the complementarity conditions in (5.8), we see that $\mu_n \neq 0$ implies $\left|\boldsymbol{w}^T\boldsymbol{x}_n + b\right| = 1$ which means that the distance from point $\boldsymbol{x}_n$ to the decision boundary is exactly equal to the margin. Such a vector $\boldsymbol{x}_n$ bears the name of *support vector* (SV). From (5.14) it follows that the solution $\boldsymbol{w}$ of problem (5.3) is merely a linear combination of SVs, hence the name of support vector machine (SVM). For a given data set of size N, the number of SVs is usually much smaller than N, formula (5.14) expresses the optimal separating weight $\boldsymbol{w}$ as a *sparse* linear representation of the data set. It is worthwhile to mention [18] that there exists an "in-sample" check of the expected out-of-sample error as

$$E[E_{\text{out}}] \leq \frac{E[\text{number of SV's}]}{N-1} \qquad (5.15)$$

Example 5.1 Apply linear SVM to classify the same data as that given in Example 2.3.

Solution

The linear SVM was applied to the data set by solving the dual problem 5.11 via CVX. Among 15 Lagrange multipliers, there are three nonzero ones. The three support vectors that are associated with these nonzero Lagrange multipliers are found to be

3.1000 8.4000 10.5000
4.9000 2.6000 0.8000

Using (5.14), the optimal $\boldsymbol{w}$ was found as a linear combination of these SV to be

w = [1.740968869612935 3.142234305706753]^T

and using (5.12) b was obtained as

b = -21.793951593763186

Each SV is marginal with a distance $1/\|\boldsymbol{w}\| = 0.2784$ to the decision boundary, see Fig. 5.1 where the SVs are marked by blue and red dots.

The MATLAB code implementing the SVM for Example 5.1 are given below.

```
% For Example 5.1 where linear SVM was applied to the same data as that
```

```
% used in Example 2.3 for PLA.
% Written by W.-S. Lu, University of Victoria. Last modified: March 23, 2015.
```

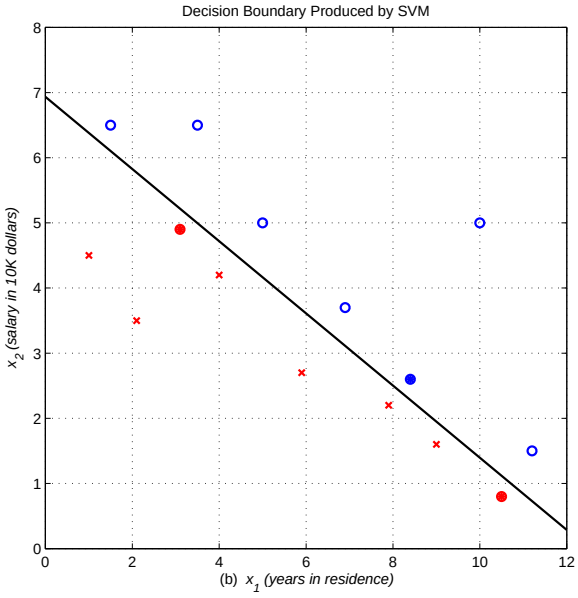

Figure 5.1 Decision boundary and three support vectors for Example 5.1.

```
% Output:
% (w,b): optimal separating weight.
% sv: support vectors.
% marg: optimal margin of decision boundary.
% Example:
% load data_ex2_3
% [w,b,sv,marg] = svm_linear(x,y,xp,xn);
function [w,b,sv,marg] = svm_linear(x,y,xp,xn)
N = length(y);
ind = 1:1:N;
y = y(:);
d = diag(y)*x';
e = ones(N,1);
cvx_begin quiet
   variable u(N,1);
   dw = d'*u;
   minimize(0.5*(dw'*dw) - e'*u);
   subject to
   y'*u == 0;
   u >= 0;
cvx_end
mu = u;
ind1 = find(mu < 1e-6);
ind2 = setdiff(ind,ind1);
sv = x(:,ind2);
```

```
mu1 = mu(ind2);
y1 = y(ind2);
c = mu1(:).*y1;
w = sv*c;
marg = 1/norm(w);
b = y1(1) - w'*sv(:,1);
wt = [b; w];
Dt = [ones(N,1) x'];
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp('optimal weights:')
wt
Ein = L/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
disp(sprintf('optimal margin of decision boundary: %d.',marg));
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
nt = size(sv,2);
for i = 1:nt,
   if y1(i) == 1,
      plot(sv(1,i),sv(2,i),'b.','linewidth',1.5)
      plot(sv(1,i),sv(2,i),'b+','linewidth',1.5)
      plot(sv(1,i),sv(2,i),'bx','linewidth',1.5)
   else
      plot(sv(1,i),sv(2,i),'r.','linewidth',1.5)
      plot(sv(1,i),sv(2,i),'r+','linewidth',1.5)
      plot(sv(1,i),sv(2,i),'ro','linewidth',1.5)
   end
end
grid
xlabel('(b)  \itx_1 (years in residence)')
ylabel('\itx_2 (salary in 10K dollars)')
p1 = 0;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 12;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([0 12 0 8])
axis square
```

■

B. ***Support Vector Machine with Nonlinear Transformation***

In this section, we describe SVMs as applied to a dilated feature space $\mathcal{Z}$ using *nonlinear* combinations of the individual features x_i's. In spirit, this bears an analogy to what we did in

Sec. 3.4 where linear models are extended from the basic feature space $\mathcal{X}$ to nonlinearly dilated feature space $\mathcal{Z}$.

We consider a transform that maps a feature vector $\boldsymbol{x} \in R^d$ to a nonlinearly dilated feature vector $\boldsymbol{z} \in R^{\tilde{d}}$ by

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) \tag{5.16}$$

For a 2-dimesinal input space R^2 for example, the general 2nd-order, 3rd-order, and 5th-order dilations assume the form

$$\Phi_2(\boldsymbol{x}) = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^T$$

$$\Phi_3(\boldsymbol{x}) = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & x_1^2x_2 & x_1x_2^2 & x_2^3 \end{bmatrix}^T$$

and

$$\Phi_5(\boldsymbol{x}) = [x_1 \; x_2 \; x_1^2 \; x_1x_2 \; x_2^2 \; x_1^3 \; x_1^2x_2 \; \cdots \; x_1^5 \; x_1^4x_2 \; x_1^3x_2^2 \; x_1^2x_2^3 \; x_1x_2^4 \; x_2^5]^T$$

respectively. Notice the subtle difference between what we did here and in Sec. 3.4 – the component "1" in $\Phi(\boldsymbol{x})$ is now moved because we have denoted the weight vector in a way to separate the "insect" b from the rest part of the weight.

Working in space $\mathcal{Z}$, the decision boundary as a counterpart of (5.1) becomes

$$\boldsymbol{w}^T\boldsymbol{z} + b = 0 \tag{5.17}$$

where $\boldsymbol{w} \in R^{\tilde{d}}$. Since the decision boundary in space $\mathcal{Z}$ remains linear as seen in (5.16), the SVM can be obtained by solving the convex QP problem

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ &\text{subject to:} && y_n(\boldsymbol{w}^T\boldsymbol{z}_n + b) \geq 1 \;\; \text{for} \;\; n = 1, \ldots, N \end{aligned} \tag{5.18}$$

where $\boldsymbol{z}_n$ are generated by $\boldsymbol{z}_n = \Phi(\boldsymbol{x}_n)$ for n = 1, …, N. Like the case in Sec. 5.1.A, the primal problem (5.18) may be tackled by solving its Lagrange dual problem

$$\begin{aligned} &\underset{\boldsymbol{\mu}}{\text{minimize}} && \frac{1}{2}\boldsymbol{\mu}^T(\tilde{\boldsymbol{d}} \cdot \tilde{\boldsymbol{d}}^T)\boldsymbol{\mu} - \boldsymbol{e}^T\boldsymbol{\mu} \\ &\text{subject to:} && \boldsymbol{\mu}^T\boldsymbol{y} = 0, \quad \boldsymbol{\mu} \geq \boldsymbol{0} \end{aligned} \tag{5.19}$$

where $\tilde{\boldsymbol{d}} = \begin{bmatrix} y_1\boldsymbol{z}_1 & y_2\boldsymbol{z}_2 & \cdots & y_N\boldsymbol{z}_N \end{bmatrix}^T$. The KKT conditions for the solution of (5.18) are the N constraints in (5.18) plus

$$\boldsymbol{w} - \sum_{n=1}^{N} \mu_n y_n \boldsymbol{z}_n = \boldsymbol{0} \tag{5.20}$$

$$-\sum_{n=1}^{N} \mu_n y_n = 0 \tag{5.21}$$

$$\mu_n(1 - y_n(\boldsymbol{w}^T\boldsymbol{z}_n + b)) = 0 \quad \text{for } n = 1, \ldots, N \tag{5.22}$$

$$\mu_n \geq 0 \quad \text{for } n = 1, \ldots, N \tag{5.23}$$

The solution $\boldsymbol{\mu}$ of the dual problem (5.19) is connected to the solution ($\boldsymbol{w}$, b) of the primal

problem (5.18) via (5.20) as

$$\boldsymbol{w} = \sum_{n=1}^{N} \mu_n y_n \boldsymbol{z}_n \tag{5.24}$$

then via any of the complementarity conditions in (5.22) with $\mu_m \neq 0$ as

$$b = y_m - \boldsymbol{w}^T \boldsymbol{z}_m \tag{5.25}$$

If we let $\mathcal{S} \subseteq \{1, 2, ..., N\}$ be the set of index with which $\mu_n \neq 0$, (5.24) can be reduced to

$$\boldsymbol{w} = \sum_{n \in \mathcal{S}} \mu_n y_n \boldsymbol{z}_n \tag{5.26}$$

From the complementarity conditions in (5.22), we see that the vectors $\boldsymbol{z}_n$ that present in (5.26) satisfy $\left| \boldsymbol{w}^T \boldsymbol{z}_n + b \right| = 1$ which means that the distance from $\boldsymbol{z}_n$ to the decision boundary is exactly equal to the margin, hence the $\boldsymbol{z}_n$'s in (5.26) are *support vectors*. In words, for an SVM with a certain nonlinear transformation $\boldsymbol{z} = \Phi(\boldsymbol{x})$, the data set $\mathcal{D}$ = {($\boldsymbol{x}_n$, y_n) for n = 1, …, N} is transformed into space $\mathcal{Z}$ as $\tilde{\mathcal{D}}$ = {($\boldsymbol{z}_n$, y_n) for n = 1, …, N}, and the nonzero (optimized) Lagrange multipliers {μ_n, for $n \in \mathcal{S}$} in that space help identify support vectors {$\boldsymbol{z}_n$, for $n \in \mathcal{S}$}. And it is in space $\mathcal{Z}$ formula (5.26) gives the optimal separating weight $\boldsymbol{w}$ as a *sparse* linear representation of $\tilde{\mathcal{D}}$ = {($\boldsymbol{z}_n$, y_n) for n = 1, …, N}.

Once an optimal weight $\boldsymbol{w}$ is found (in space $\mathcal{Z}$), a *nonlinear* decision boundary is space $\mathcal{X}$ can be obtained by combining (5.17) with (5.16) as

$$\boldsymbol{w}^T \Phi(\boldsymbol{x}) + b = 0 \tag{5.27}$$

Example 5.2 Apply SVM with nonlinear transformation to classify the linearly non-separable "double semi-circle", a data set utilized in Chapter 3. Here the training data is the same as that generated in Example 3.1 with N = 2000.

Solution

For SVM with nonlinear transformation, the dimension of dual variable $\boldsymbol{\mu}$ in dual problem (5.19) is N while the dimension of ($\boldsymbol{w}$, b) in primal problem (5.18) is $\tilde{d}+1$ which typically is much less than N = 2000. The number of constraints in the primal and dual problems are practically the same (being N + 1 and N, respectively). Since The SVM with third-order nonlinear transform $\boldsymbol{z} = \Phi_3(\boldsymbol{x})$ was applied to the data set by solving the primal problem (5.18).

Optimal $\boldsymbol{w}$ was found to be

-0.006919943508105
0.000017123243147
-0.037769011917928
-0.003294822314068
0.000891025633946
0.001678288960348
-0.000156410899954
0.000379761282649
-0.003379784198872

and optimal b was obtained as b = 1.232073071853626, which defines the decision boundary

$$\boldsymbol{w}^T \Phi_3(\boldsymbol{x}) + b = 0$$

The support vectors in this case reside in 9-dimensional space Z and can be identified as those zn that satisfy $y_n(\boldsymbol{w}^T \boldsymbol{z}_n + b) = 1$. In this way, seven SVs were detected whose counterparts in the basic feature space X were found to be

10.2288	2.4884	3.3034	21.2379	9.6739	-2.4578	22.5842
0.1156	0.7152	-2.9492	-3.8696	2.6056	0.8448	0.9720

The decision boundary together with the training data is depicted in Fig. 5.2. We stress that by definition the vectors given above are *not* SV's, however inspecting the figure indicates that the distance between each of these points to the decision boundary remains small although not guaranteed to be the smallest.

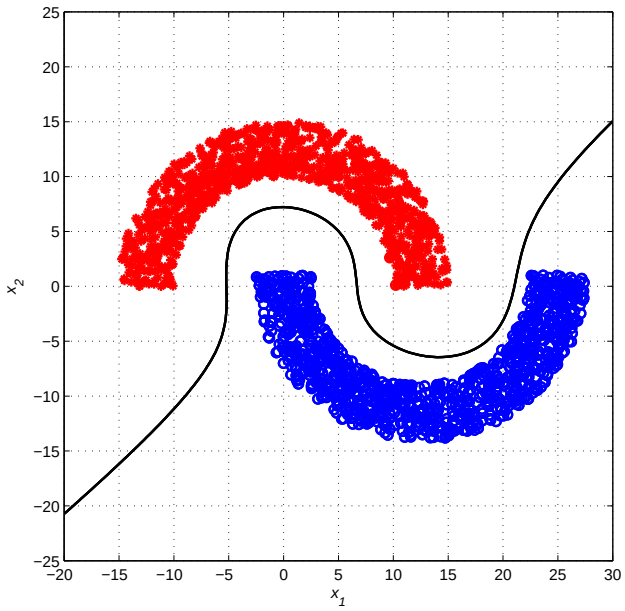

Figure 5.2 Decision boundary and training data for Example 5.2.

The MATLAB code implementing the SVM for Example 5.2 are given below.

```
% For Example 5.2 where SVM with 3rd-order nonlinearity was
% applied to the same data as that produced in Example 3.1.
% Written by W.-S. Lu, University of Victoria. Last modified: March 23, 2015.
% Output:
% (w,b): optimal separating weight.
% sv: "support vectors" in basic feature space.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
```

```
% [w,b,sv] = svm_NL_semi_circle(x,y,xp,xn,17);
function [w,b,sv] = svm_NL_semi_circle(x,y,xp,xn,st)
N = length(y);
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
y = y(:);
z1 = zeros(7,N);
for i = 1:N,
    z1(:,i) = [x(1,i)^2; x(1,i)*x(2,i); x(2,i)^2; x(1,i)^3; x(1,i)^2*x(2,i); x(1,i)*x(2,i)^2; x(2,i)^3];
end
z = [x; z1];
D = diag(y)*z';
e = ones(N,1);
cvx_begin quiet
   variable w(9,1);
   variable b(1)
   minimize(w'*w);
   subject to
   D*w + y*b >= e;
cvx_end
c = D*w + y*b;
ind1 = find(abs(c - e) < (1e-5)*e);
sv = x(:,ind1);
y1 = y(ind1);
wt = [b; w];
Dt = [ones(N,1) z'];
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp('optimal weights:')
wt
Ein = L/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
nt = size(sv,2);
for i = 1:nt,
    if y1(i) == 1,
        plot(sv(1,i),sv(2,i),'b.','linewidth',1.5)
        plot(sv(1,i),sv(2,i),'b+','linewidth',1.5)
        plot(sv(1,i),sv(2,i),'bx','linewidth',1.5)
    else
        plot(sv(1,i),sv(2,i),'r.','linewidth',1.5)
```

```
      plot(sv(1,i),sv(2,i),'r+','linewidth',1.5)
      plot(sv(1,i),sv(2,i),'ro','linewidth',1.5)
   end
end
[x1,x2] = meshgrid(-20:50/100:30,-25:50/100:25);
w0 = wt(1); w1 = wt(2); w2 = wt(3); w3 = wt(4); w4 = wt(5); w5 = wt(6);
w6 = wt(7); w7 = wt(8); w8 = wt(9); w9 = wt(10);
h1 = w0 + w1*x1 + w2*x2 + w3*(x1.^2) + w4*(x1.*x2) + w5*(x2.^2);
h = h1 + w6*(x1.^3) + w7*((x1.^2).*x2) + w8*(x1.*(x2.^2)) + w9*(x2.^3);
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis square
axis([-20 30 -25 25])
hold off
```

■

5.2 Kernel Trick and Soft-Margin Support Vector Machines

A. ***The Kernel Trick***

Definition $K(\boldsymbol{x}, \boldsymbol{x}')$ is said to be a valid *kernel* function if it is symmetric and positive definite, i.e. $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}', \boldsymbol{x})$, and the matrix

$$\begin{bmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & K(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ K(\boldsymbol{x}_2, \boldsymbol{x}_1) & K(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ K(\boldsymbol{x}_N, \boldsymbol{x}_1) & K(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & K(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \tag{5.28}$$

is positive semidefinite for any $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, which is known as *Mercer's condition*.

Commonly used kernels include polynomial kernels of the form

$$K_Q(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^Q \text{ with } c > 0 \text{ and } Q \text{ a positive integer} \tag{5.29}$$

and 'Gaussian' kernel

$$K_G(\boldsymbol{x}, \boldsymbol{x}') = e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\sigma^2} \tag{5.30}$$

There are two reasons that motivate introduction and study of kernel functions. First, given a basic feature space $\mathcal{X}$, a kernel function induces a dilated feature space $\mathcal{Z}$ such that *computing an inner product in space $\mathcal{Z}$ becomes straightforward*. Consider a kernel $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + 1)^2$ for a two-dimensional feature space $\mathcal{X} = R^2$ with $\boldsymbol{x} = [x_1\ x_2]^T$ and $\boldsymbol{x}' = [x_1'\ \ x_2']^T$. We can write

$$K_2(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + 1)^2 = 1 + 2x_1x_1' + 2x_2x_2' + x_1^2x_1'^2 + 2x_1x_1'x_2x_2' + x_2^2x_2'^2 \tag{5.31}$$

If we consider a dilated feature space $\mathcal{Z}$ with

$$\boldsymbol{z} = \hat{\Phi}_2(\boldsymbol{x}) = \begin{bmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}$$

then we have

$$\boldsymbol{z}^T\boldsymbol{z}' = (\boldsymbol{x}^T\boldsymbol{x}'+1)^2 = K_2(\boldsymbol{x},\boldsymbol{x}')$$

Similarly, a Q^{th}-order polynomial kernel $K_Q(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}'+1)^Q$ induces a Q^{th}-order feature space $\mathcal{Z}$ such that $\boldsymbol{z}^T\boldsymbol{z}' = (\boldsymbol{x}^T\boldsymbol{x}'+1)^Q = K_Q(\boldsymbol{x},\boldsymbol{x}')$. Furthermore, note that for a scalar feature space $\mathcal{X}$ we can write the Gaussian kernel with $\sigma = 1/\sqrt{2}$ as

$$K_G(x,x') = e^{-(x-x')^2} = e^{-x^2}e^{-x'^2}\sum_{k=0}^{\infty}\frac{2^k x^k x'^k}{k!} \tag{5.32}$$

hence a Gaussian kernel induces an infinite-dimensional feature space $\mathcal{Z}$ such that

$$\boldsymbol{z}^T\boldsymbol{z}' = e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\sigma^2} = K_G(\boldsymbol{x},\boldsymbol{x}')$$

In summary, a valid kernel $K(\boldsymbol{x},\boldsymbol{x}')$ induces a dilated feature space $\mathcal{Z}$ in which

$$\boldsymbol{z}^T\boldsymbol{z}' = K(\boldsymbol{x},\boldsymbol{x}') \tag{5.33}$$

Note that using (5.33) to compute the inner product in space $\mathcal{Z}$ can be quite economical, especially when both the dimension of the basic feature space $\mathcal{X}$ and order of the nonlinear transformation Q are not too small.

The second reason we are interested in kernel functions is the role they may ML algorithms including SVMs. To be specific, we reexamine the dual problem for SVM with nonlinear transformation that is addressed in Sec. 5.1.B. In the objective function in (5.19), its quadratic term is given by $\frac{1}{2}\boldsymbol{\mu}^T(\tilde{\boldsymbol{d}}\cdot\tilde{\boldsymbol{d}}^T)\boldsymbol{\mu}$ with $\tilde{\boldsymbol{d}} = \begin{bmatrix} y_1\boldsymbol{z}_1 & y_2\boldsymbol{z}_2 & \cdots & y_N\boldsymbol{z}_N \end{bmatrix}^T$, hence

$$\tilde{\boldsymbol{d}}\cdot\tilde{\boldsymbol{d}}^T = \begin{bmatrix} y_1y_1\boldsymbol{z}_1^T\boldsymbol{z}_1 & y_1y_2\boldsymbol{z}_1^T\boldsymbol{z}_2 & \cdots & y_1y_N\boldsymbol{z}_1^T\boldsymbol{z}_N \\ y_2y_1\boldsymbol{z}_2^T\boldsymbol{z}_1 & y_2y_2\boldsymbol{z}_2^T\boldsymbol{z}_2 & \cdots & y_2y_N\boldsymbol{z}_2^T\boldsymbol{z}_N \\ \vdots & \vdots & \vdots & \vdots \\ y_Ny_1\boldsymbol{z}_N^T\boldsymbol{z}_1 & y_Ny_2\boldsymbol{z}_N^T\boldsymbol{z}_2 & \cdots & y_Ny_N\boldsymbol{z}_N^T\boldsymbol{z}_N \end{bmatrix}$$

Following (5.33), we have

$$\tilde{\boldsymbol{d}}\cdot\tilde{\boldsymbol{d}}^T = \begin{bmatrix} y_1y_1K(\boldsymbol{x}_1,\boldsymbol{x}_1) & y_1y_2K(\boldsymbol{x}_1,\boldsymbol{x}_2) & \cdots & y_1y_NK(\boldsymbol{x}_1,\boldsymbol{x}_N) \\ y_2y_1K(\boldsymbol{x}_2,\boldsymbol{x}_1) & y_2y_2K(\boldsymbol{x}_2,\boldsymbol{x}_2) & \cdots & y_2y_NK(\boldsymbol{x}_2,\boldsymbol{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ y_Ny_1K(\boldsymbol{x}_N,\boldsymbol{x}_1) & y_Ny_2K(\boldsymbol{x}_N,\boldsymbol{x}_2) & \cdots & y_Ny_NK(\boldsymbol{x}_N,\boldsymbol{x}_N) \end{bmatrix} \tag{5.34}$$

Moreover, using (5.26), the decision boundary can be expressed as

$$\boldsymbol{w}^T\boldsymbol{z}+b = \sum_{n\in\mathcal{S}}\mu_n y_n\boldsymbol{z}_n^T\boldsymbol{z}+b = \sum_{n\in\mathcal{S}}\mu_n y_n K(\boldsymbol{x}_n,\boldsymbol{x})+b = 0 \tag{5.35}$$

where b can be written as

$$b = y_m - \boldsymbol{w}^T\boldsymbol{z}_m = y_m - \sum_{n\in\mathcal{S}}\mu_n y_n\boldsymbol{z}_n^T\boldsymbol{z}_m = y_m - \sum_{n\in\mathcal{S}}\mu_n y_n K(\boldsymbol{x}_n,\boldsymbol{x}_m) \tag{5.36}$$

By means by (5.34)-(5.36), training SVMs in dual space becomes computationally more attractive.

Example 5.3 Apply SVM with a Gaussian kernel to classify the linearly non-separable "double semi-circle", a data set utilized in Chapter 3. The training data is the same as that generated in Example 3.1 with $N = 2000$.

Solution

As can be seen from (5.34), a kernel is inherently associated with the dual problem which in the present case involves $N = 2000$ variables as the components of $\boldsymbol{\mu}$. The dual problem in (5.19), where matrix $\tilde{\boldsymbol{d}} \cdot \tilde{\boldsymbol{d}}^T$ is given by (5.33) and $K(\boldsymbol{x}, \boldsymbol{x}') = K_G(\boldsymbol{x}, \boldsymbol{x}') = e^{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\sigma^2}$, was solved using CVX. The CPU time consumed by CVX was 158.15 seconds on a 64-bit desktop PC with Intel i7-3770 CPU@3.4GHz with 32GB RAM. It is of importance to note the role parameter σ in the Gaussian kernel (see (5.30)) plays – it controls the degree of closeness between the decision boundary and the "support vectors". Here the parameter was set to $\sigma = 0.8$ as it yielded a satisfactory decision boundary as seen in Fig. 5.3. Another thing to note is that the optimal weight vector $\boldsymbol{w}$ itself cannot be made available because $\boldsymbol{w}$ resides in space $\mathcal{Z}$ which in the case of Gaussian kernel is an infinite-dimensional space. Nevertheless, the decision boundary is explicitly defined in input space $\mathcal{X}$, see (5.35), where b is computed by (5.36).

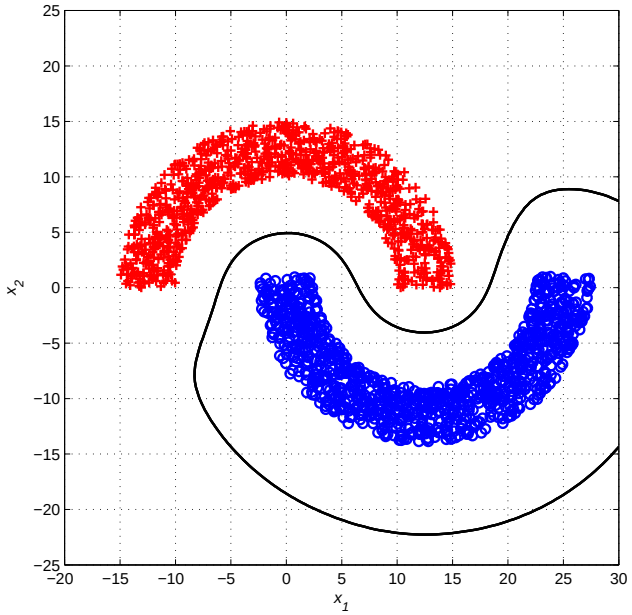

Figure 5.3 Decision boundary and training data for Example 5.3.

The MATLAB code implementing the SVM for Example 5.3 are given below.

```
% For Example 5.3 where SVM with a Gaussian kernel was applied to the same
% data as that produced in Example 3.1.
% Written by W.-S. Lu, University of Victoria. Last modified: March 26, 2015.
% Input:
```

```
% (x,y); training data.
% xp: training data with positive labels.
% xn: training data with negative labels.
% st: initial random state for mixing training data.
% sig: \sigma for Gaussian kernel.
% Output:
% mu: optimal solution of the dual problem.
% b: optimal intersect.
% sv: "support vectors" in basic feature space.
% cpt: CPU time consumed by CVX.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [mu,b,sv] = svm_gaussian_semi_circle(x,y,xp,xn,17,0.8);
function [mu,b,sv,cpt] = svm_gaussian_semi_circle(x,y,xp,xn,st,sig)
N = length(y);
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
y = y(:);
D1 = eye(N);
for i = 1:(N-1),
    for j = (i+1):N,
        nij = (norm(x(:,i)-x(:,j)))^2;
        D1(i,j) = exp(-nij/(2*sig^2));
        D1(j,i) = D1(i,j);
    end
end
Y = y*y';
D = Y.*D1;
clear Y
e = ones(N,1);
cvx_begin quiet
   variable u(N,1);
   minimize(0.5*u'*D*u - e'*u);
   subject to
   u'*y == 0;
   u >= 0;
cvx_end
cpt = cvx_cputime
mu = u;
ind = 1:1:N;
ind1 = find(mu < 1e-5);
ind2 = setdiff(ind,ind1);
sv = x(:,ind2);
mu1 = mu(ind2);
y1 = y(ind2);
c = mu1(:).*y1;
di = D1(ind2(1),ind2);
```

```
b = y1(1) - di*c;
nt = length(ind2);
dw = b*ones(N,1);
for i = 1:N,
   xi = x(:,i);
   dwi = dw(i);
   for j = 1:nt,
      nij = (norm(sv(:,j)-xi))^2;
      dij = exp(-nij/(2*sig^2));
      dwi = dwi + c(j)*dij;
   end
   dw(i) = dwi;
end
dwt = (dw >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
Ein = L/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
N2 = 100;
[x1,x2] = meshgrid(-20:50/N2:30,-25:50/N2:25);
h = b*ones(N2+1,N2+1);
for i = 1:nt,
   x1w = sv(1,i);
   x2w = sv(2,i);
   m1w = (x1w - x1).^2;
   m2w = (x2w - x2).^2;
   m2 = sqrt(m1w + m2w)/(2*sig^2);
   Kw = exp(-m2);
   h = h + c(i)*Kw;
end
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis square
axis([-20 30 -25 25])
hold off   ■
```

B. ***Soft-Margin Support Vector Machines***

The standard SVM works for linearly separable data, but is not suited for non-separable data. A variant of SVM known as *soft-margin* SVM deal was developed to deal with non-separable data.

The soft-margin SVM modifies the standard SVM formulation (see (5.3)) in constraints as well as objective function, and the modifications are accomplished by introducing N nonnegative

slack variables where N is the size of the training data. Specifically, the constraints in (5.3) are *relaxed* to

$$y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - s_n \text{ with } s_n \geq 0 \text{ for } n = 1,...,N$$

where slack variable s_n may be regarded as an amount of violation for data pair $(\boldsymbol{x}_n, y_n)$. Therefore, the "total violation" by all constraints can be measured by $\sum_{n=1}^{N} s_n$. To control the amount of total violation while maximizing the margin, the objective in (5.3) is modified by including a term $\tau\sum_{n=1}^{N} s_n$ with parameter $\tau > 0$ making a tradeoff between maximizing the margin and minimizing the total violation. Summarizing, the soft-margin SVM solves the convex QP problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \tau\sum_{n=1}^{N} s_n \\ \text{subject to:} \quad & y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - s_n \quad \text{for } n = 1,...,N \\ & s_n \geq 0 \qquad\qquad\quad \text{for } n = 1,...,N \end{aligned} \tag{5.37}$$

The Lagrangian of (5.37) is given by

$$L(\boldsymbol{w}, b, \boldsymbol{s}, \boldsymbol{\mu}, \boldsymbol{\nu}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \tau\sum_{n=1}^{N} s_n + \sum_{n=1}^{N}\mu_n(1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - s_n) - \sum_{n=1}^{N}\nu_n s_n$$

and the KKT conditions constitute the constraints in (5.37) plus

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{n=1}^{N}\mu_n y_n \boldsymbol{x}_n = \boldsymbol{0} \tag{5.38}$$

$$\frac{\partial L}{\partial b} = -\sum_{n=1}^{N}\mu_n y_n = 0 \tag{5.39}$$

$$\frac{\partial L}{\partial s_n} = \tau - \mu_n - \nu_n = 0 \quad \text{for } n = 1, \ldots, N \tag{5.40}$$

$$\mu_n(1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - s_n) = 0 \quad \text{for } n = 1, \ldots, N \tag{5.41}$$

$$\nu_n \cdot s_n = 0 \quad \text{for } n = 1, \ldots, N \tag{5.42}$$

$$\mu_n \geq 0 \quad \text{for } n = 1, \ldots, N \tag{5.43}$$

$$\nu_n \geq 0 \quad \text{for } n = 1, \ldots, N \tag{5.44}$$

Using (5.38), (5.39), (5.40), and (5.44), the Lagrangian is reduced to a function of $\boldsymbol{\mu}$ as

$$L(\boldsymbol{\mu}) = -\frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{d}\cdot\boldsymbol{d}^T)\boldsymbol{\mu} + \boldsymbol{e}^T\boldsymbol{\mu} \tag{5.45}$$

where $\boldsymbol{d} = \begin{bmatrix} y_1\boldsymbol{x}_1 & y_2\boldsymbol{x}_2 & \cdots & y_N\boldsymbol{x}_N \end{bmatrix}^T$ hence the dual problem is formulated as

$$\begin{aligned} \underset{\boldsymbol{\mu}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{d}\cdot\boldsymbol{d}^T)\boldsymbol{\mu} - \boldsymbol{e}^T\boldsymbol{\mu} \\ \text{subject to:} \quad & \boldsymbol{\mu}^T\boldsymbol{y} = 0, \quad \boldsymbol{0} \leq \boldsymbol{\mu} \leq \tau\boldsymbol{e} \end{aligned} \tag{5.46}$$

On comparing (5.46) with its counterpart (5.11) for the standard SVM, we see that the only

difference between them is that an upper bound τ is imposed for soft-margin SVM.
Having solved the dual problem (5.46) for $\boldsymbol{\mu}$, the optimal $\boldsymbol{w}$ is obtained from (5.38) as

$$\boldsymbol{w} = \sum_{n=1}^{N} \mu_n y_n \boldsymbol{x}_n$$

which can be reduced to only include the terms with nonzero μ_n, namely

$$\boldsymbol{w} = \sum_{n \in \mathcal{S}} \mu_n y_n \boldsymbol{x}_n \tag{5.47}$$

The points $\{\boldsymbol{x}_n,\ n \in \mathcal{S}\}$ are called support vectors. For soft-margin SVM, however, it is important to realize there are two types of support vectors:

• ***Margin support vectors*** These vectors $\boldsymbol{x}_n$ are associated μ_n with $0 < \mu_n < \tau$. It follows (5.41) that $1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = s_n$. Because $\mu_n < \tau$, (5.40) implies that ν_n is nonzero, hence (5.42) implies that $s_n = 0$. Therefore we have

$$y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = 1 \tag{5.48}$$

• ***Non-margin support vectors*** These vectors $\boldsymbol{x}_n$ are associated μ_n with $\mu_n = \tau$. Again, it follows (5.41) that $1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) = s_n$. Now, because $\mu_n = \tau$, (5.40) leads to $\nu_n = 0$, hence (5.42) and (5.44) imply that

$$y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) < 1$$

To determine optimal intersect b, let $\boldsymbol{x}_m$ be a margin support vector. From (5.48) it follows that

$$b = y_m - \boldsymbol{w}^T \boldsymbol{x}_m \tag{5.49}$$

where $\boldsymbol{w}$ is given by (5.47).

Soft-Margin Support Vector Machines with Nonlinear Transformation

It is rather straightforward to extend the soft-margin SVM from feature space $\mathcal{X}$ to a dilated space $\mathcal{Z}$ with nonlinear transformation $\boldsymbol{z} = \Phi(\boldsymbol{x})$. The primal and dual problems for space $\mathcal{Z}$ are given by

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} + \tau \textstyle\sum_{n=1}^{N} s_n \\ &\text{subject to:} && y_n(\boldsymbol{w}^T \boldsymbol{z}_n + b) \geq 1 - s_n \quad \text{for } n = 1, \ldots, N \\ & && s_n \geq 0 \qquad\qquad\ \ \text{for } n = 1, \ldots, N \end{aligned} \tag{5.50}$$

and

$$\begin{aligned} &\underset{\boldsymbol{\mu}}{\text{minimize}} && \frac{1}{2}\boldsymbol{\mu}^T (\tilde{\boldsymbol{d}} \cdot \tilde{\boldsymbol{d}}^T)\boldsymbol{\mu} - \boldsymbol{e}^T \boldsymbol{\mu} \\ &\text{subject to:} && \boldsymbol{\mu}^T \boldsymbol{y} = 0, \quad \boldsymbol{0} \leq \boldsymbol{\mu} \leq \tau \boldsymbol{e} \end{aligned} \tag{5.51}$$

respectively, where $\tilde{\boldsymbol{d}} = \begin{bmatrix} y_1 \boldsymbol{z}_1 & y_2 \boldsymbol{z}_2 & \cdots & y_N \boldsymbol{z}_N \end{bmatrix}^T$.

Example 5.4 Apply soft-margin SVM with a second-order nonlinear transformation to the data set utilized in Example 4.6 to classify handwritten digit "1" from other 9 digits. The size of the

training data is $N = 500$ in a 2-dimensional feature space.

Solution

With $\boldsymbol{z} = \Phi_2(\boldsymbol{x})$ and $\tau = 0.7$, soft-margin SVM was applied by solving the primal problem (5.50). The optimal separating weight $\boldsymbol{w}$ was found to be

-1.723630104245624
2.833732449476559
-1.273873229835718
-0.130448829121387
3.332236270017219

and b = -2.171268426130970 which defines a decision boundary shown in Fig. 5.4. Out of 500 digits, a total of 488 digits were classified correctly, representing an error rate of 2.4%.

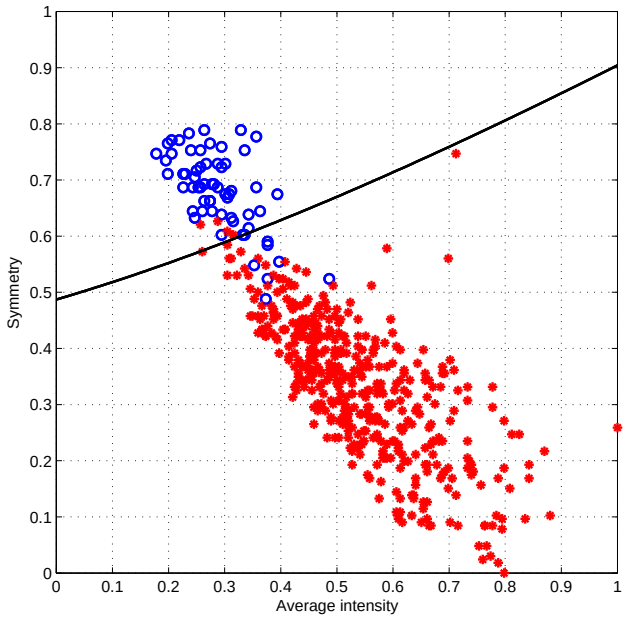

Figure 5.4 Decision boundary and training data for Example 5.4.

The MATLAB code implementing the SVM for Example 5.4 are given below.

```
% For Example 5.4 where soft-margin SVM with 2rd-order nonlinearity
% was applied to the same data as that produced in Example 4.6.
% Written by W.-S. Lu, University of Victoria. Last modified: March 27, 2015.
% Input:
% (x,y): training data.
% tau: parameter \tau for the total violation term.
% Output:
% (w,b): optimal separating weight.
% Example:
% load data_ex5_4
```

```
% [w,b] = svm_soft_NL(x,y,xp,xn,0.7);
function [w,b] = svm_soft_NL(x,y,xp,xn,tau)
N = length(y);
y = y(:);
z1 = zeros(3,N);
for i = 1:N,
    z1(:,i) = [x(1,i)^2; x(1,i)*x(2,i); x(2,i)^2];
end
z = [x; z1];
D = diag(y)*z';
e = ones(N,1);
cvx_begin quiet
   variable w(5,1);
   variable b(1);
   variable s(N,1);
   minimize(0.5*w'*w + tau*sum(s));
   subject to
   D*w + y*b + s >= e;
   s >= 0;
cvx_end
wt = [b; w];
Dt = [ones(N,1) z'];
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp('optimal weights:')
wt
Ein = L/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
[x1,x2] = meshgrid(0:1/100:1,0:1/100:1);
w0 = wt(1); w1 = wt(2); w2 = wt(3); w3 = wt(4); w4 = wt(5); w5 = wt(6);
h = w0 + w1*x1 + w2*x2 + w3*(x1.^2) + w4*(x1.*x2) + w5*(x2.^2);
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
grid
xlabel('Average intensity')
ylabel('Symmetry')
axis square
axis([0 1 0 1])
hold off     ■
```

5.3 Radial Basis Functions for Classification and Regression

The radius of a diverging circle centered at point $\boldsymbol{x}_n$ is measure by the 2- norm $\|\boldsymbol{x} - \boldsymbol{x}_n\|$, and a "local" function associated with a data point $\boldsymbol{x}_n$ is said to be a *radial basis function* (RBF) if it is a function of $\|\boldsymbol{x} - \boldsymbol{x}_n\|$ and can act as a member of basis functions for functional representation and approximation purposes. A class of commonly used radial basis functions (RBFs) are *Gaussian* RBFs defined by

$$K(\boldsymbol{x}, \boldsymbol{x}_n) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{x}_n\|^2} \tag{5.52}$$

where $\gamma > 0$ is a user-defined parameter to control the decay rate of the RBF. It turns out that the value of γ effects the performance of a RBF-based model in a significant way, hence should be chosen with care. On comparing (5.52) with (5.30), we see that Gaussian RBFs are essentially Gaussian kernels. Recall a kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ is said to be positive definite if matrix $\left(K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)$ is positive semidefinite for *any* $\{\boldsymbol{x}_i, i = 1, \ldots, N\}$ (see Sec. 5.2.A). It can be shown that Gaussian RBFs are positive definite.

In a RBF model with training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, \ldots, N\}$, the hypothesis function assumes the form

$$h(\boldsymbol{x}) = \sum_{n=1}^{N} w_n K(\boldsymbol{x}, \boldsymbol{x}_n) \tag{5.53}$$

for regression problems, and

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{n=1}^{N} w_n K(\boldsymbol{x}, \boldsymbol{x}_n)\right) \tag{5.54}$$

for classification problems, where coefficients $\{w_n, n = 1, \ldots, N\}$ are determined by minimizing an error measure. Because of the structure of function $h(\boldsymbol{x})$ (having exactly N terms) and the particular choice of RBF in (5.52) which is known to be *positive define* (i.e., it obeys Mercer's condition), alternatively coefficients $\{w_n, n = 1, \ldots, N\}$ may be determined by solving the *exact interpolation* problem

$$h(\boldsymbol{x}_m) = \sum_{n=1}^{N} w_n K(\boldsymbol{x}_m, \boldsymbol{x}_n) = y_m \quad \text{for } m = 1, \ldots, N \tag{5.55}$$

Define

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \boldsymbol{K} = \begin{bmatrix} K(\boldsymbol{x}_i, \boldsymbol{x}_j) & K(\boldsymbol{x}_i, \boldsymbol{x}_j) & \cdots & K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ K(\boldsymbol{x}_i, \boldsymbol{x}_j) & K(\boldsymbol{x}_i, \boldsymbol{x}_j) & \cdots & K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \vdots & \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_i, \boldsymbol{x}_j) & K(\boldsymbol{x}_i, \boldsymbol{x}_j) & \cdots & K(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{bmatrix} \tag{5.56}$$

the linear system of equations in (5.55) can be expressed as

$$\boldsymbol{K}\boldsymbol{w} = \boldsymbol{y} \tag{5.57}$$

where $\boldsymbol{K}$ is positive semidefinite. If $\boldsymbol{K}$ is positive definite, the exact interpolation problem in (5.55) has a unique solution which can be found efficiently by solving (5.57) using for example conjugate-gradient method [5]. If $\boldsymbol{K}$ is only positive semidefinite, system (5.57) is modified to

$$(\boldsymbol{K} + \lambda \boldsymbol{I})\boldsymbol{w} = \boldsymbol{y} \tag{5.58}$$

where parameter $\lambda > 0$ regularizes the linear system to deal with the singularity of $\boldsymbol{K}$, data noise as well as possible overfitting issues. Since system (5.58) is always positive definite, conjugate-gradient method finds its solution efficiently. In the literature, the classification hypothesis function in (5.54) with $\boldsymbol{w}$ given by the solution of (5.58) is also known as *regularized least-squares classification* (RLSC) [11].

Example 5.5 Apply RLSC to the data set utilized in Example 5.4 to classify handwritten digit "1" from other 9 digits. The size of the training data is $N = 500$ in a 2-dimensional feature space.

Solution

With $\gamma = 90$ and $\lambda = 7$, RLSC was the data set utilized in Example 5.4. The optimal $\boldsymbol{w}$ was found by solving (5.58) using conjugate-gradient method [5]. The decision boundary obtained is shown in Fig. 5.5. Out of 500 digits, a total of 491 digits were classified correctly, representing an error rate of 1.8%.

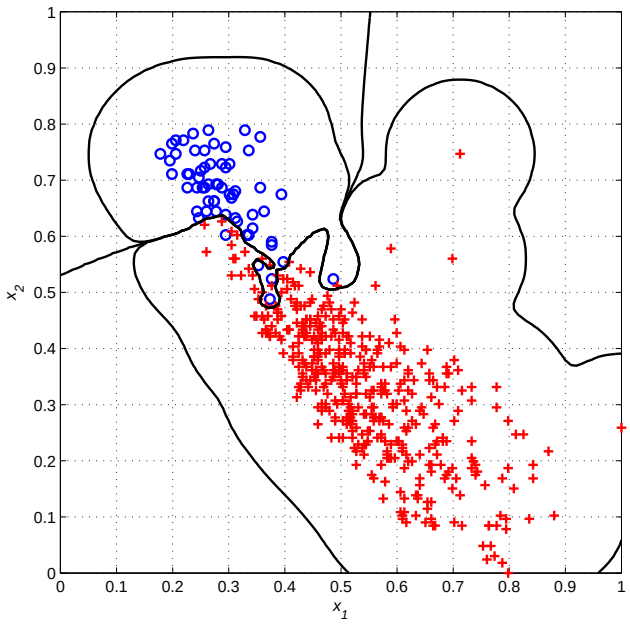

Figure 5.5 Decision boundary and training data for Example 5.5.

The MATLAB codes implementing RLSC for Example 5.5 are given below.

```
% For Example 5.5 where a Gaussian RBF was applied to the same data
% in Example 5.4 for classify digit 1 from other digits..
% Written by W.-S. Lu, University of Victoria. Last modified: March 28, 2015.
% Input:
% (x,y); training data.
% xp: training data with positive labels.
% xn: training data with negative labels.
% gam: parameter \gamma for Gaussian RBF.
```

```
% lam: parameter \lambda for regularized LS classification (RLSC).
% Output:
% mu: optimal solution of the dual problem.
% b: optimal intersect.
% sv: "support vectors" in basic feature space.
% cpt: CPU time consumed by CVX.
% Example:
% load data_ex5_4
% w = rlsc_gaussian_digits(x,y,xp,xn,90,7);
function w = rlsc_gaussian_digits(x,y,xp,xn,gam,lam)
N = length(y);
y = y(:);
I = eye(N,N);
K = I;
for i = 1:(N-1),
   for j = (i+1):N,
      nij = (norm(x(:,i)-x(:,j)))^2;
      K(i,j) = exp(-gam*nij);
      K(j,i) = K(i,j);
   end
end
w = conj_gradient_2015(lam*I+K,y,1e-6);
% w = inv(lam*I+K)*y;
dw = zeros(N,1);
for i = 1:N,
   xi = x(:,i);
   dwi = dw(i);
   for j = 1:N,
      nij = (norm(x(:,j)-xi))^2;
      dij = exp(-gam*nij);
      dwi = dwi + w(j)*dij;
   end
   dw(i) = dwi;
end
dwt = (dw >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
Ein = L/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
N2 = 100;
[x1,x2] = meshgrid(0:1/N2:1,0:1/N2:1);
h = zeros(N2+1,N2+1);
for i = 1:N,
   x1w = x(1,i);
   x2w = x(2,i);
   m1w = (x1w - x1).^2;
   m2w = (x2w - x2).^2;
   m2 = gam*sqrt(m1w + m2w);
   Kw = exp(-m2);
```

```
    h = h + w(i)*Kw;
end
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis square
axis([0 1 0 1])
hold off
```

```
% Conjugate-gradient algorithm for Ax = b with A positive definite.
% Theory: See Practical Optimization Sec. 6.9.
% Input:
% {A,b} -- data for linear equation A and b.
% epsi: termination tolerance.
% Output:
% xs: solution point.
% Written by W.-S. Lu, University of Victoria. Last modified: March 28, 2015.
function xs = conj_gradient_2015(A,b,epsi)
n = length(b);
x0 = zeros(n,1);
xk = x0;
rk = b - A*xk;
dk = rk;
rk2_old = rk'*rk;
err = norm(rk);
while  err >= epsi,
    adk = A*dk;
    ak = rk2_old/(dk'*adk);
    xk = xk + ak*dk;
    rk = rk - ak*adk;
    rk2 = rk'*rk;
    bk = rk2/rk2_old;
    rk2_old = rk2;
    dk = rk + bk*dk;
    err = norm(rk);
end
xs = xk;     ■
```