Exercise 2 -Solution

SVM

Problem 1

1) Consider the following linearly separable problem:

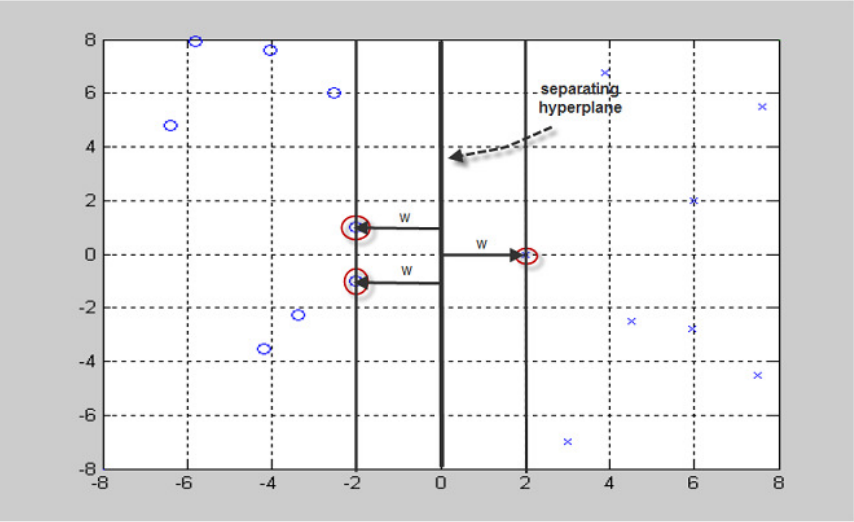


Fig. 1.1

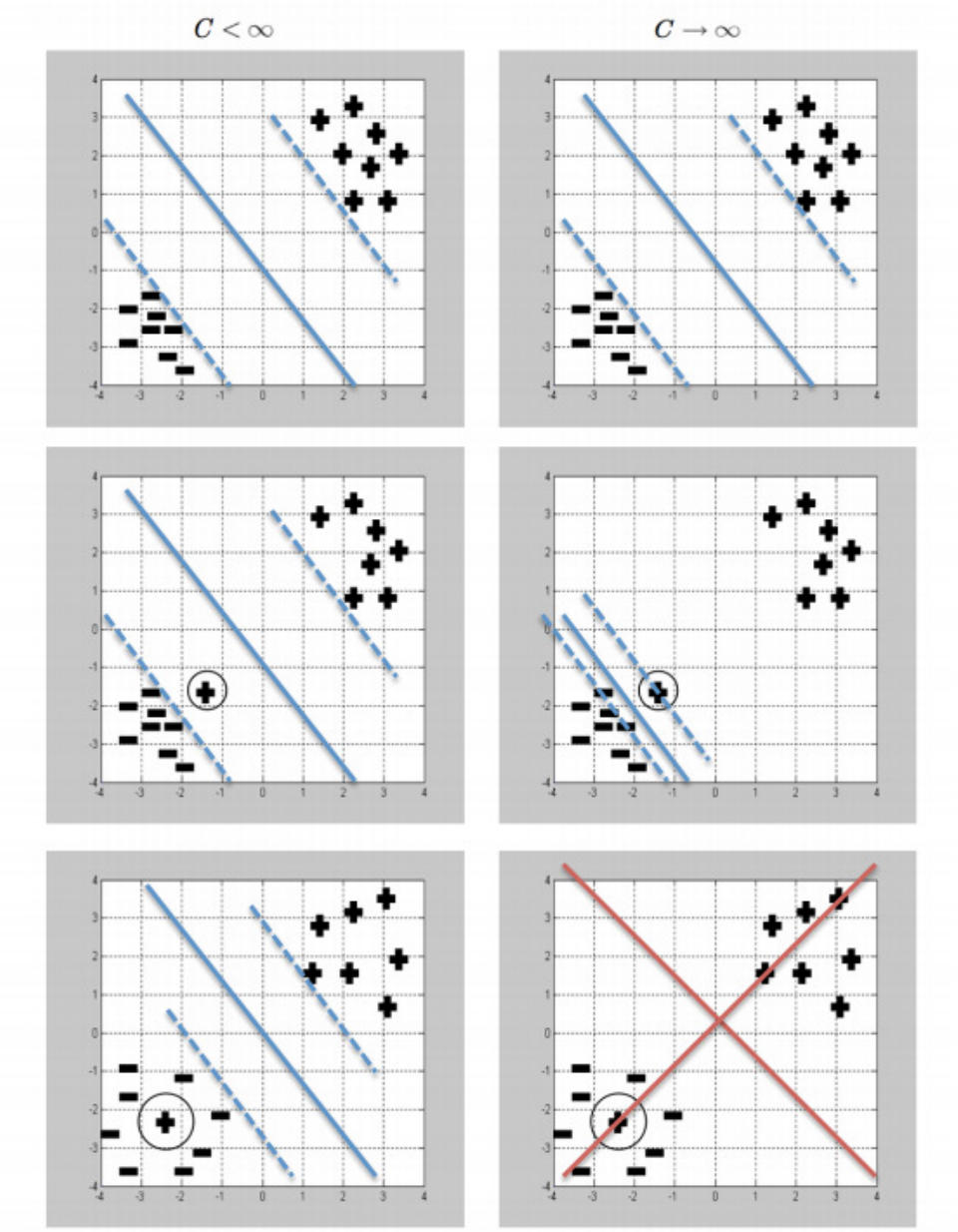
a. The above data points with red circle are the support vectors.

b. The direction of the vector w, which is perpendicular to the separable hyperplane, is shown in Fig. 1.1.

Problem 2

a. The hyperplanes are shown as follow. The last graph is not separable when C gets close to infinity since the plus in the circle is surrounded by the minus outside.

b. For the second row, when $C < \infty$, $\xi > 1$, since the circled plus resided on the negative side of the hyperplane. When $C \to \infty$, $\xi \to 0$.
For the third row, when $C < \infty$, ξ will be larger than 1, for similar reason above.



Problem 3

a. Assume that we have 5 original sample points in Fig. 3.1 and the yellow circled points are the support vector. And the blue point is the added one. In this case, $k = 2$.

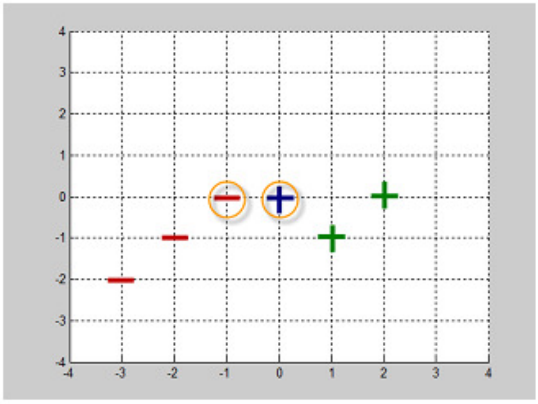


Fig. 3.1

b. When the points are added like in Fig. 3.2, the number of support vector grow to $k + 1$.

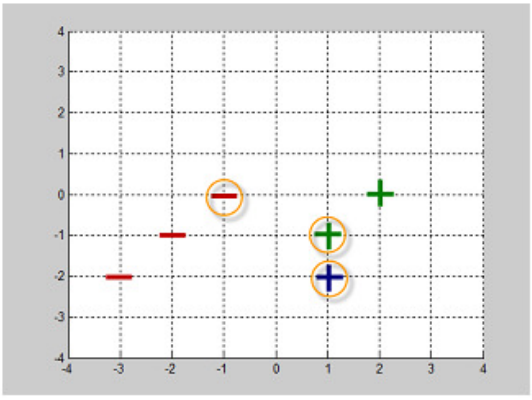


Fig. 3.2

c. When the points are added like in Fig. 3.3, the number of support vector grow to $n + 1$.

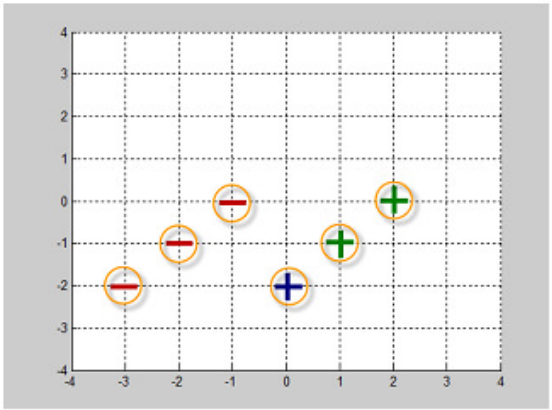


Fig. 3.3

Kernel Methods

Problem 4

a. The conditions required from a kernel function are:

Symmetric: Since k_1, k_2 are symmetric functions thus their sum must be a symmetric function:

$$k_3(x,x') = k_1(x,x') + k_2(x,x') = k_1(x',x) + k_2(x',x) = k_3(x',x)$$

PSD : Since k_1, k_2 are kernel function the matrices K_1, K_2 defined by

$\{K_{i,kl}\} = k(x_k, x_l)$ are PSD meaning, $a^T K_1 a \geq 0$, $b^T K_2 b \geq 0 \quad \forall a,b$

Therefore, K_3 is also PSD

$$c^T K_3 c = c^T (K_1 + K_2) c = c^T K_1 c + c^T K_2 c \geq 0 \quad \forall c$$

b. The problem is linearly separable for k_3 :

We denote $k_1(x,x') = \Phi_1^T(x)\Phi_1(x')$, $k_2(x,x') = \Phi_2^T(x)\Phi_2(x')$ (Mercer's theorem).

Therefore we can write k_3 as:

$$k_3(x,x') = \Phi_1^T(x)\Phi_1(x') + \Phi_2^T(x)\Phi_2(x') = [\Phi_1, \Phi_2]^T \cdot [\Phi_1, \Phi_2] = \Phi_3^T(x)\Phi_3(x')$$

Meaning that the feature vector obtained by the mapping of k_3 contains the both the feature the obtained by the mapping of k_1 and k_2.

Since the problem is linearly separable for k_1 then it will linearly separable for k_3

Problem 5

For the given samples

x	Y
(-1,-1)	-1
(-1,+1)	1
(+1,-1)	1
(+1,+1)	-1

c. Linear SVM – cannot achieve a zero training error. (the samples given in a XOR arrangement)

d. SVM with a polynomial kernel function of degree 2: The features obtained by a this mapping are: $\left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$. Since the mapping function contains a multiplication between the first and the second dimension of the input the classifier will achieve a zero training error.

e. SVM with a Gaussian kernel function $K_\lambda(x,z) = e^{\frac{-\|x-z\|}{\lambda}}$: For a sufficiently large, λ the kernel matrix approaches the identity matrix. Thus for each sample from the training set:

$$\phi(x_i)^T w = \phi(x_i)^T \sum_{k=1}^{n} \alpha_k y_k \phi(x_k) = \sum_{k=1}^{n} \alpha_k y_k K(x_i, x_k) \approx \alpha_i y_i K(x_i, x_i) = \alpha_i y_i$$

$$\Rightarrow sign(\phi(x_i)^T w) = sign(\alpha_i y_i) = y_i$$

Therefore a zero training error is achieved.

Gradient Algorithm

Problem 6

6) Given the function:

$$f(x,y) = -20\left(\frac{x}{2} - x^2 - y^5\right) exp(-x^2 - y^2)$$

a. Using Matlab sketch this function within the area of $-3 \leq x,y \leq 3$, wen obtain

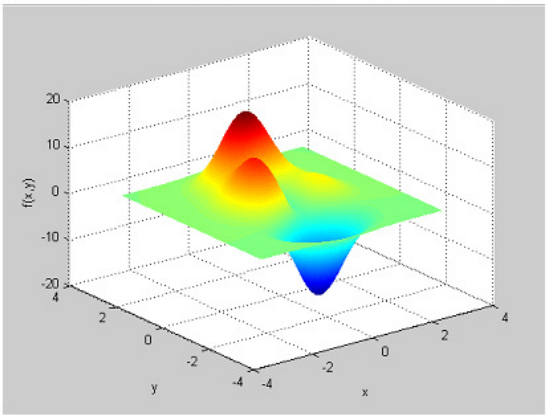


Fig. 6.1

b. Gradient descent method for finding the minimum point.
Pseudo code:

```
% Gradient descent algorithm
function gda(x,y,g)
close all
%------mesh the figure of the function f(x,y)----------------
[X,Y]=meshgrid(linspace(-3,3,300),linspace(-3,3,300));
Z=-20.*(X./2-X.^2-Y.^5).*exp(-X.^2-Y.^2);
mesh(X,Y,Z);
hold on;
xlabel('x');ylabel('y');zlabel('f(x,y)');
%--------------------------threshold--------------------------
```

```
thresh = 0.01;
%-------------------------iteration time-----------------------
iter_times = 0;
w=[x,y];
%------matrix to store the coordinate of the value of the-----
%------function at each step---------------------------------
coordi=ones(3,100);
for i=1:100
    fxy = -20*(0.5*w(1)-w(1)^2-w(2)^5)*exp(-w(1)^2-w(2)^2);
    coordi(1,i)=w(1); coordi(2,i)=w(2); coordi(3,i)=fxy;
    [dfx,dfy] = G(w(1),w(2));
    w = w - g.*[dfx,dfy];
    delta_w = g.*[dfx,dfy];
    iter_times = i;
    if norm(delta_w) < thresh || norm(delta_w) == thresh
        break;
    end
end
```

```
%------------show convergence path & iteration result----------
showpath(coordi(1,1:iter_times),coordi(2,1:iter_times),coordi(3,1:iter_t
imes))
w
iter_times
fxy=coordi(3,iter_times)
end

% subfucntion G(x) within GDA
function [fx1,fy1]=G(x1,y1)
%-------differentiation vector of given f(x,y)------------
fx1= (-10+40*x1+20*x1^2-40*x1^3-40*x1*y1^5)*exp(-x1^2-y1^2);
fy1= (20*x1*y1-40*x1^2*y1+100*y1^4-40*y1^6)*exp(-x1^2-y1^2);
end

% subfucntion showpath(x,y) show the convergence path
function showpath(x2,y2,z3)
Nx=length(x2);Ny=length(y2);Nz=length(z3);
plot3(x2,y2,z3,'--O');
text(x2(1,1),y2(1,1),z3(1,1),'initial ponit')
text(x2(1,Nx),y2(1,Ny),z3(1,Nz),'end point')
figure;
plot(1:Nz,z2);
xlabel('Interation Times')
ylabel('z')
title(['convergence graph with initial guess:[x0,y0]=[',...
    num2str(x2(1,1)),',',num2str(y2(1,1)),']'])
end
```

c. Initialize the algorithm with $[x_0, y_0] = [0.1,1]$ and step size $\eta = 0.01$.
Run the Matlab code, we gain the convergence graph showed in Fig. 6.2

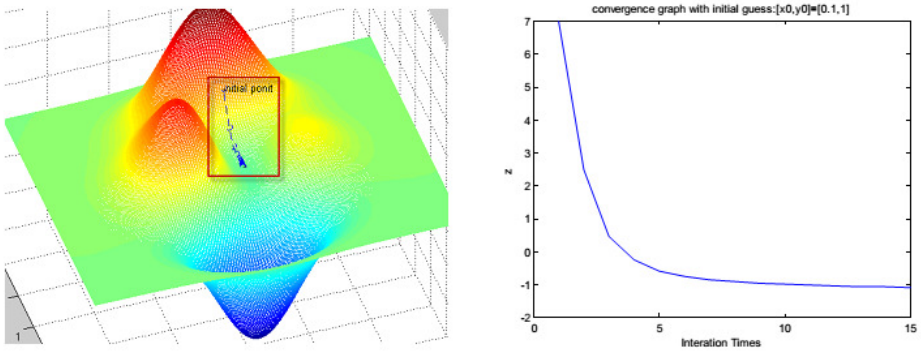


Fig. 6.2

And the coordinates of the converging point is

$$x_{min} = 0.2356, y_{min} = 0.2514, f(x_{min}, y_{min}) = -1.0886$$

d. Initialize the algorithm with $[x_0, y_0] = [1.5, -1]$ and step size $\eta = 0.05$.
Run the Matlab code, we gain the convergence graph showed in Fig. 6.3.

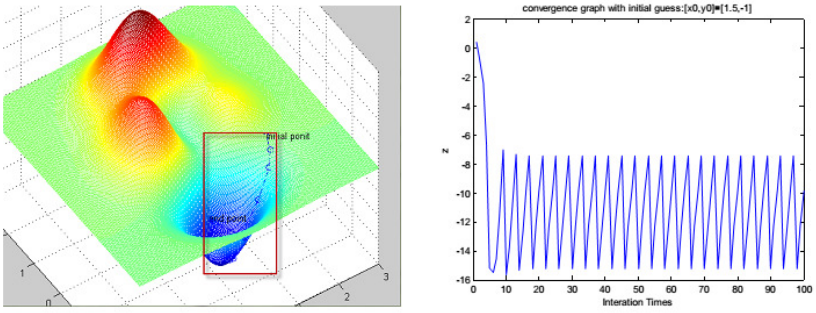


Fig. 6.3

As we can see from results, the iteration stops at the point $x_{end} = 0.1455, y_{end} = -2.233, f(x_{end}, y_{end}) = -9.8503$ when the iteration number goes to 100 which is also set to be the upper limit number of the iteration.

Thus, the iteration is not converged when the initial guess is $[x_0, y_0] = [1.5, -1]$ and the step size is $\eta = 0.05$

e. Initialize the algorithm with $[x_0, y_0] = [1.5, -1]$ and step size $\eta = 0.01$.
Run the Matlab code, we gain the convergence graph showed in fig. 6.4.

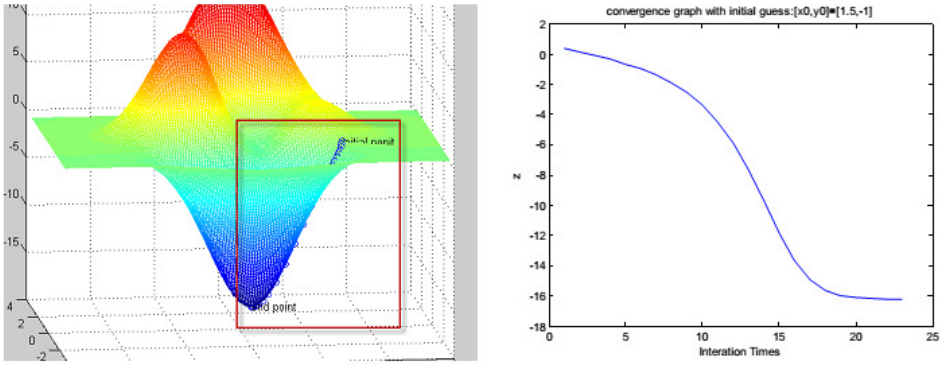


Fig. 6.4

And the coordinates of the converging point is

$$x_{min} = 0.0366, y_{min} = -1.5793, f(x_{min}, y_{min}) = -16.2249$$

The iteration number is 23. Compare to the result of section (d), the only difference is this time the step size is $\eta = 0.01$ and this leads to the iteration converges with the same initial guess point $[x_0, y_0] = [1.5, -1]$.

Perceptron

Problem 7

The answer is n.

Proof :

We assume $sign(0) = 0$, then w_{final} will have the form of $\sum_{i=1}^{n} \alpha_i x_i$.

Since $sign(w_{final}^T x_i) = d_i$, and $x_i^T x_j = 0 \quad (i \neq j)$, the coefficients α_i must have the same sign as d_i.

Therefore, to classify all x s correctly, all α s cannot be zero.

Considering each update sets one α to nonzero at most, the algorithm performs n updates at the least.

Regression

Problem 8

a.

i.

$$E_{SSE,\lambda} = \sum_{k=1}^{n}(\omega^T\varphi(x_k)\varphi(x_k)^T\omega - 2\sum_{k=1}^{n}\omega^T\varphi(x_k)y_k + y_k^2) = \omega^T\Phi^T\Phi\omega - 2\omega^T\Phi^T y + \lambda\omega^T\omega + y^T y.$$

Set $\partial_\omega E_{SSE,\lambda} = 0$, we have

$$2\Phi^T\Phi\omega^* - 2\Phi^T y + 2\lambda I\omega^* = 0.$$

Thus,

$$\omega^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T y.$$

Let R equal to λI, then we achieve the desired form.

ii. According to previous calculation, in this case $R = \lambda I$.

iii. For high values of λ, since $(\Phi^T\Phi + \lambda I)^{-1}$ will be close to 0, ω^* will be close to zero.

b.

For an arbitrary non-zero vector x in R^d, we have

$$x^T(\Phi^T\Phi + \lambda I)x = (x\Phi)^2 + \lambda x^2 > 0$$

which implies that $\Phi^T\Phi + \lambda I$ is a positive definite matrix. Since positive definite matrices have all positive eigenvalues, they are invertible.

c.

i. In this case, Φ is a zero matrix, $\omega^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T y = 0$.

ii. Given $\varphi(\cdot) = (1, \cdots, 1)$ and let α denote the all one **row** vector of dimension d, we have

$$\omega^* = (\sum_{k=1}^{n}\varphi(x_k)\varphi(x_k)^T + \lambda I)^{-1}\Phi^T y$$

$$= (n\alpha^T\alpha + \lambda I)^{-1}\Phi^T y = (\tfrac{1}{\lambda}I - \tfrac{\frac{1}{\lambda^2}n\alpha^T\alpha}{1+\frac{1}{\lambda}nd})\sum_{k=1}^{n} y_k\alpha^T = \tfrac{\sum_{k=1}^{n} y_k}{\lambda}(\alpha^T - \tfrac{nd\alpha^T}{\lambda+nd})$$

$$= \tfrac{\sum_{k=1}^{n} y_k}{\lambda}(1 - \tfrac{nd}{\lambda+nd})\alpha^T$$

$$= \tfrac{\sum_{k=1}^{n} y_k}{\lambda+nd}\alpha^T.$$

d.

If the we change $\lambda\omega^T\omega$ in cost function to $\lambda\alpha\omega$, the optimal weight $\omega^* = (\Phi y + \lambda\alpha)(\Phi\Phi^T)^{-1}$. A high value of λ does not imply low ω.