Homework 6

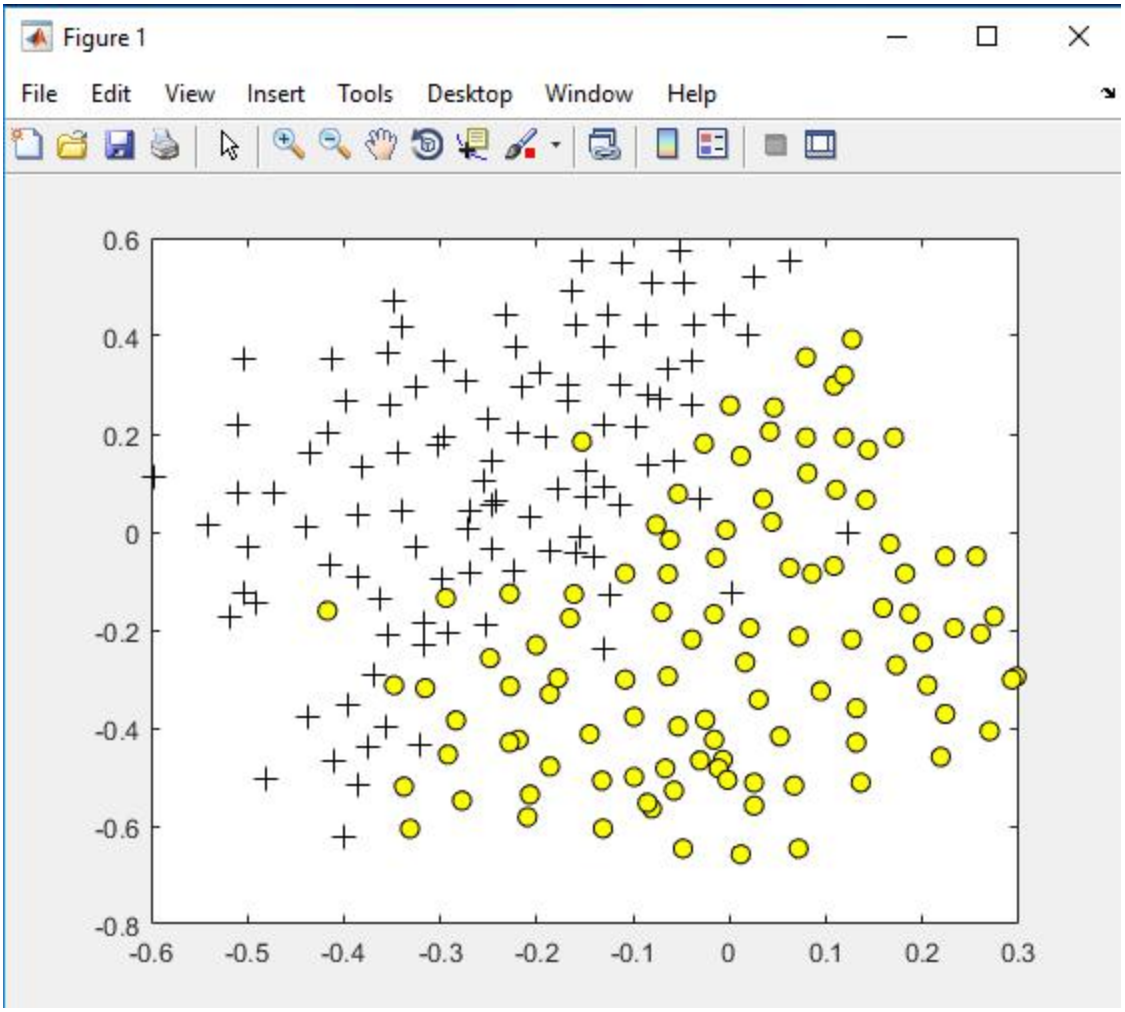
April 21st, 2018

Xiongming Dai

Problem 1

(1)A support vector machine is used to classify two classes. For "dataset2.mat",we can find the data X with dimension 211x2, the associated ground truth is y with dimension 211x2. We use X and y as training. The data Xval with dimension 200x2, and yval with dimension 200x1. We consider Xval and yval as test.

We run the code, and we can get the figure of the data for training (X and y), which is shown as follows



The corresponding MATLAB code is written as follows:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
clc
clear all
close all
%%
 %load('data1.mat');
load('data2.mat');

%% Design SVM
% Type your code here and save your architecture under tha name 'model'

x_train=X;
y_train=y;
C=2500;

model=fitcsvm(x_train,y_train,'KernelFunction','polynomial',...
    'BoxConstraint',C,'ClassNames',[0,1]);
```

```
%% Plot the data
pos = find(y == 1); neg = find(y == 0);
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 1, 'MarkerSize', 7)
hold on;
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7)
hold off;
x1plot = linspace(min(X(:,1)), max(X(:,1)), 100)';
x2plot = linspace(min(X(:,2)), max(X(:,2)), 100)';
[X1, X2] = meshgrid(x1plot, x2plot);
vals = zeros(size(X1));
for i = 1:size(X1, 2)
   this_X = [X1(:, i), X2(:, i)];
   vals(:, i) = predict(model, this_X);
end

system('pause');
x_test=Xval;
y_test=yval;
[y_svm,scoress]=predict(model,x_test);
accuracy=sum(y_svm==y_test)/length(y_test);
error_rate=1-accuracy;

SVM1=model;
%% Plot the data and the decision boundary
d = 0.02;
[x1Grid,x2Grid] = meshgrid(min(x_test(:,1)):d:max(x_test(:,1)),...
    min(x_test(:,2)):d:max(x_test(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
[~,scores] = predict(SVM1,xGrid);
h(1:2) = gscatter(x_test(:,1),x_test(:,2),y_test,'rb','.');
hold on
%h(3) =
plot(x_test(SVM1.IsSupportVector,1),x_test(SVM1.IsSupportVector,2),'ko');
contour(x1Grid,x2Grid,reshape(scores(:,2),size(x1Grid)),[0 0],'k');
legend(h,{'0','+1','Support Vectors'});
axis equal
hold off

% % Plot the SVM boundary
% hold on
% contour(X1, X2, vals);
% hold off;
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

The classification error rates for different value of box constraint are shown as follows:

Box Constraint C----------Classification error rate

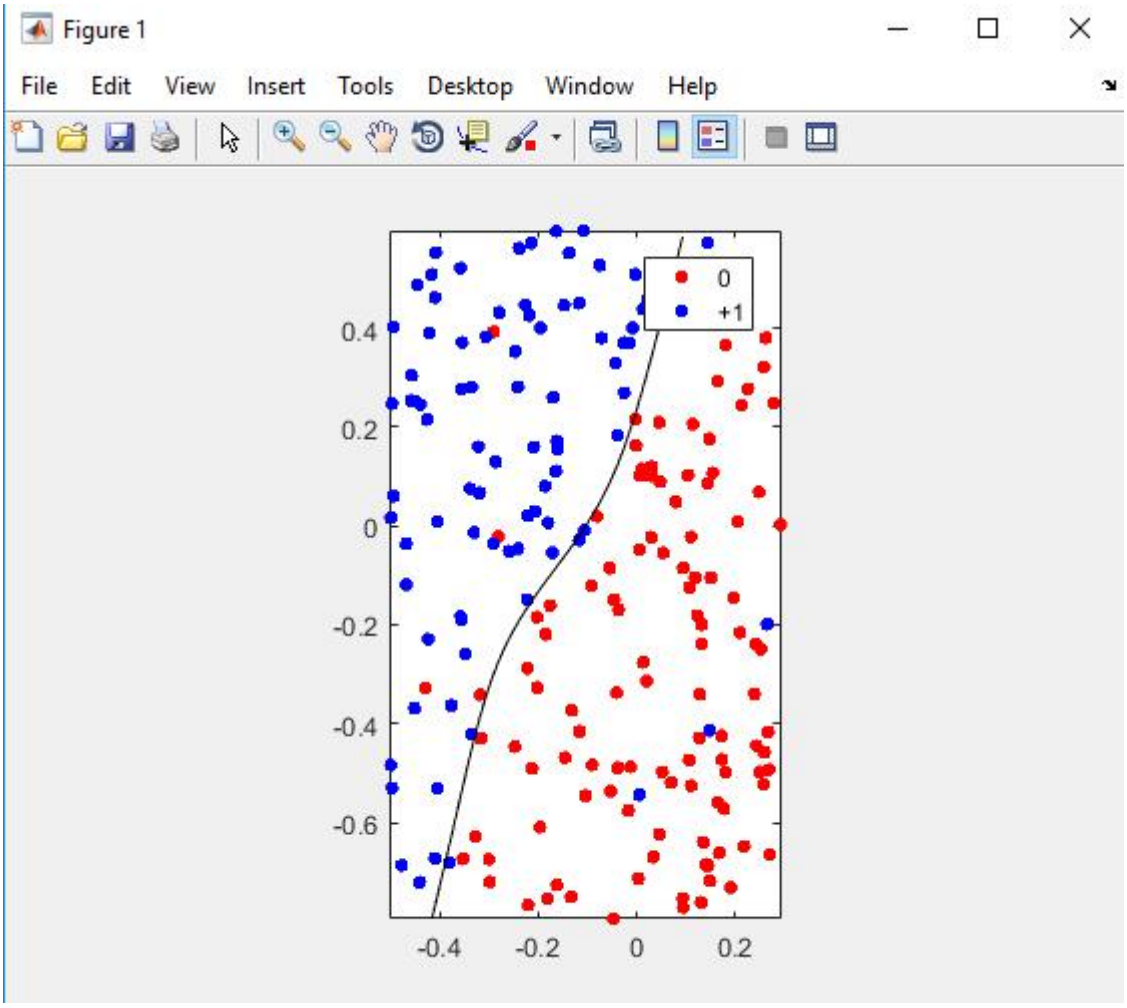
2500----------6.5%

1500----------6.5%

1000----------6.5%

10----------6.5%

As you can see from the result, the classification error rate is not changed when we change the parameter C, C is a coefficient for the training of the model. While the classification error rate is not changed, it is mainly on the test dataset, in this part, we choose the test dataset is “Xval” and “yval”. The figure is shown as follows(C=2500):



Now, we try to change the test dataset by "X" and "y" instead. We have following values:

Box Constraint C----------Classification error rate

2500----------6.64%

1500----------7.11%

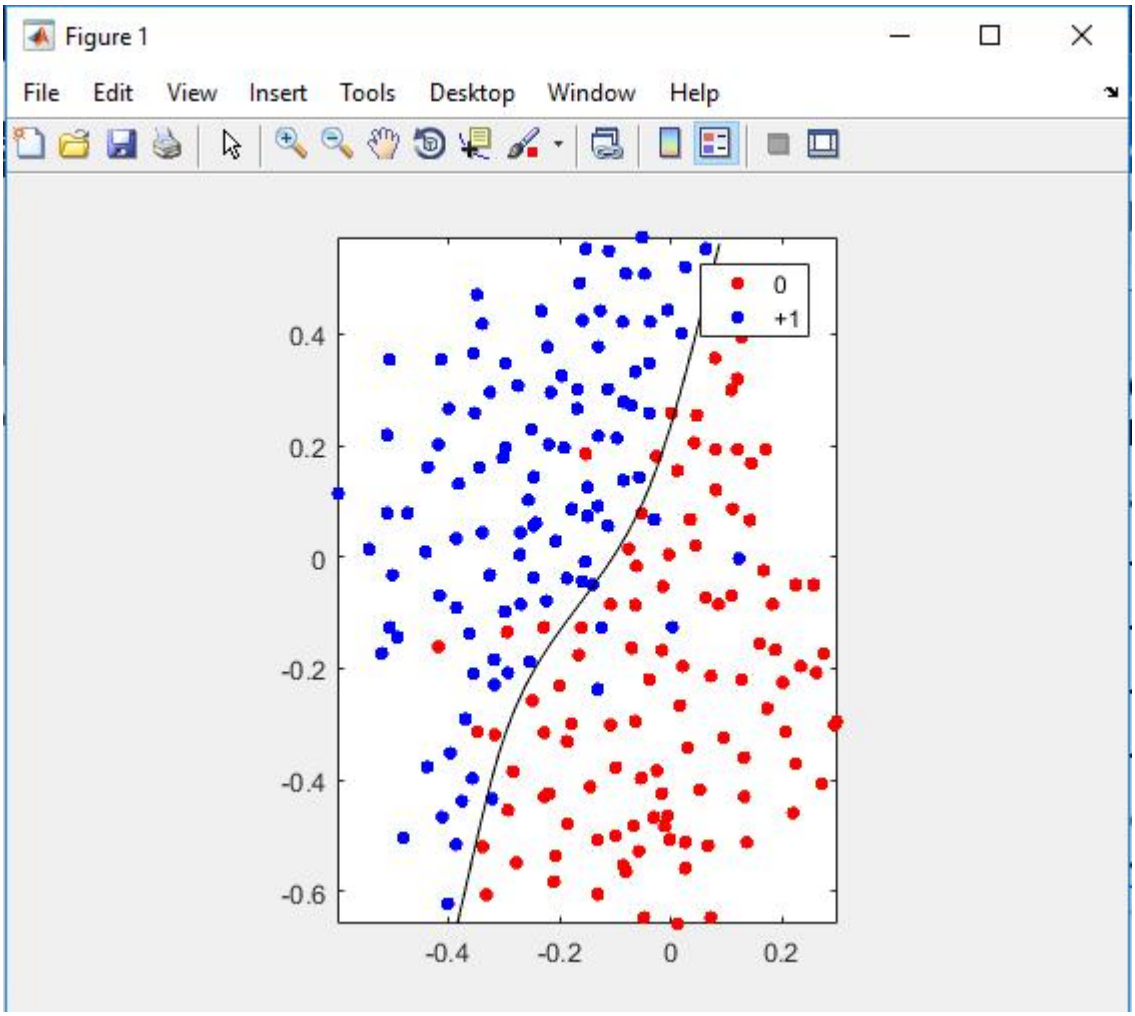
1000----------7.11%

10----------6.64%

As we can see from the above, the error rate when C is smaller(10) or larger(2500), we can obtain relative good accuracy and lower error rate(6.64% error rate) .

The figure is shown as follows

a.(C=2500):



b.(C=1500)

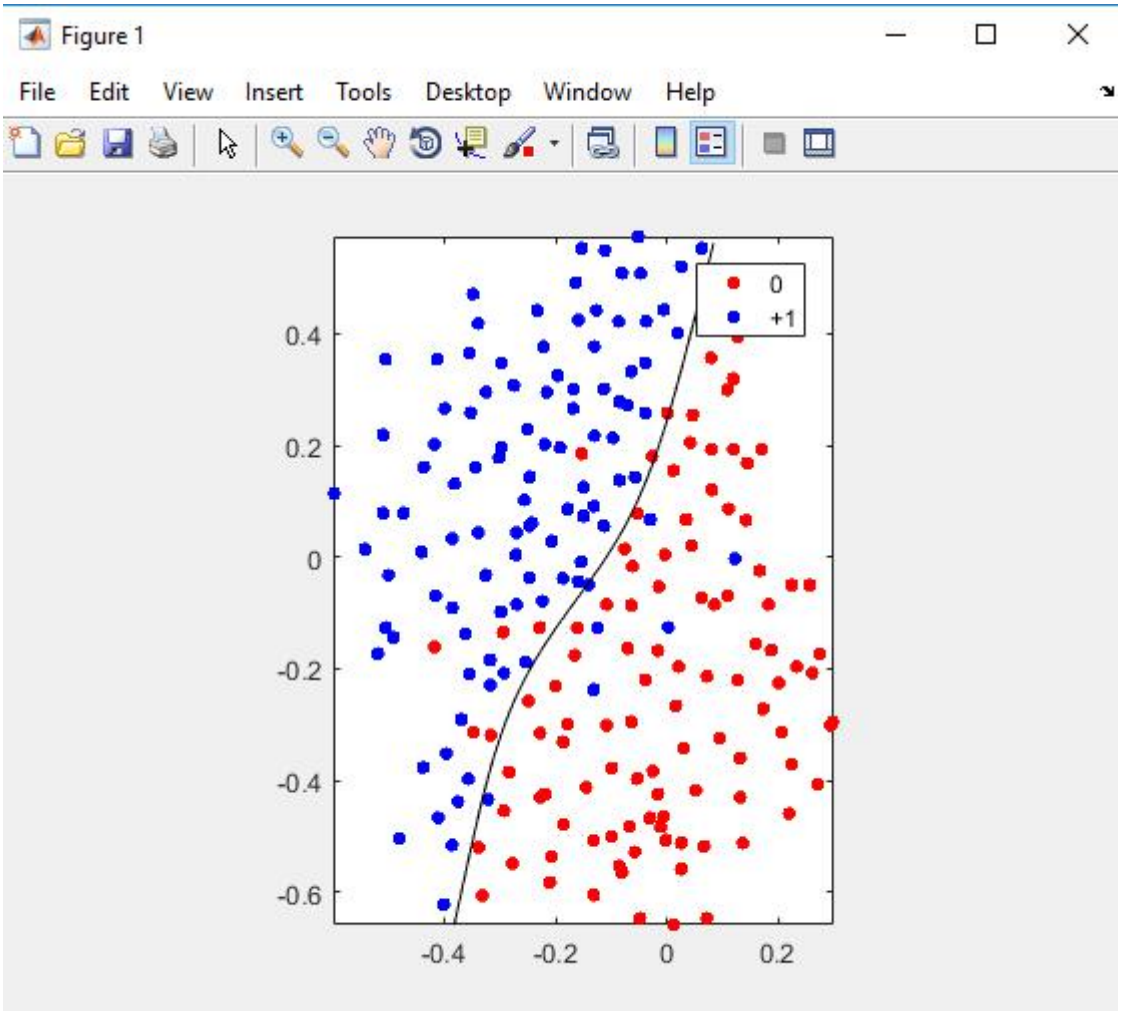


Figure 1
File Edit View Insert Tools Desktop Window Help
0
+1
0.4
0.2
0
-0.2
-0.4
-0.6
-0.4
-0.2
0
0.2

c.(C=1000)

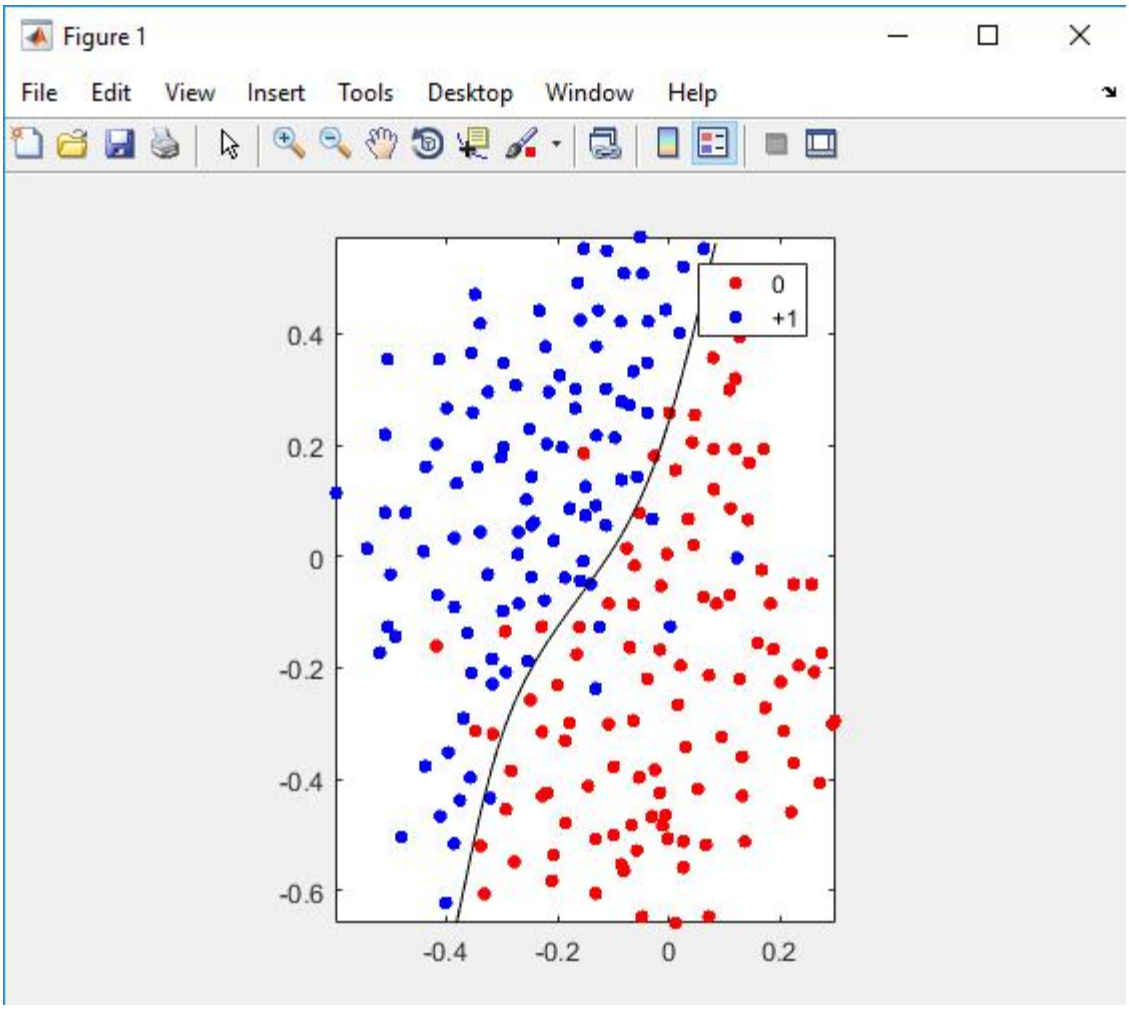
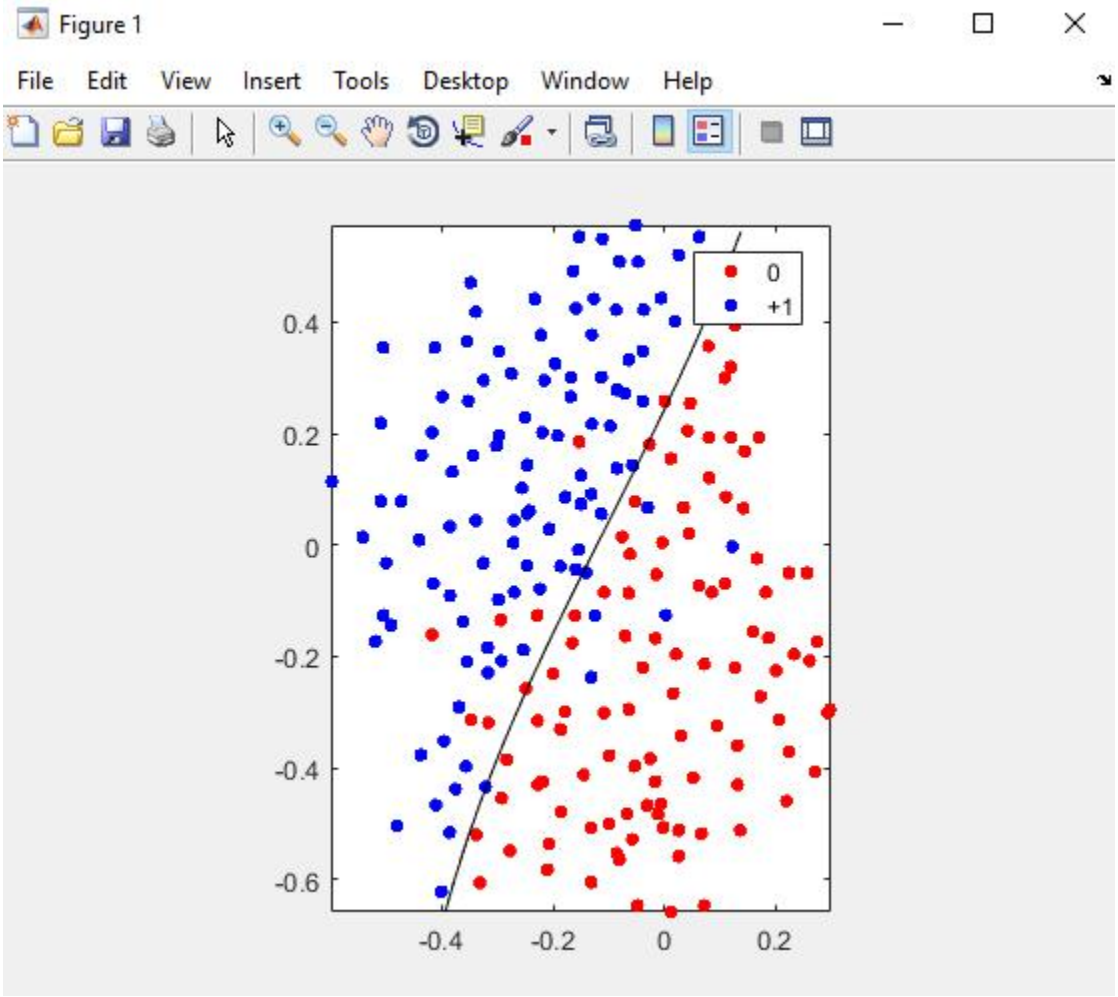


Figure 1
File Edit View Insert Tools Desktop Window Help
0
+1
0.4
0.2
0
-0.2
-0.4
-0.6
-0.4
-0.2
0
0.2

d.(C=10)



For “dataset1.mat”, we have the following values:

Box Constraint C----------Classification error rate

2500----------15.76%

1500----------15.87%

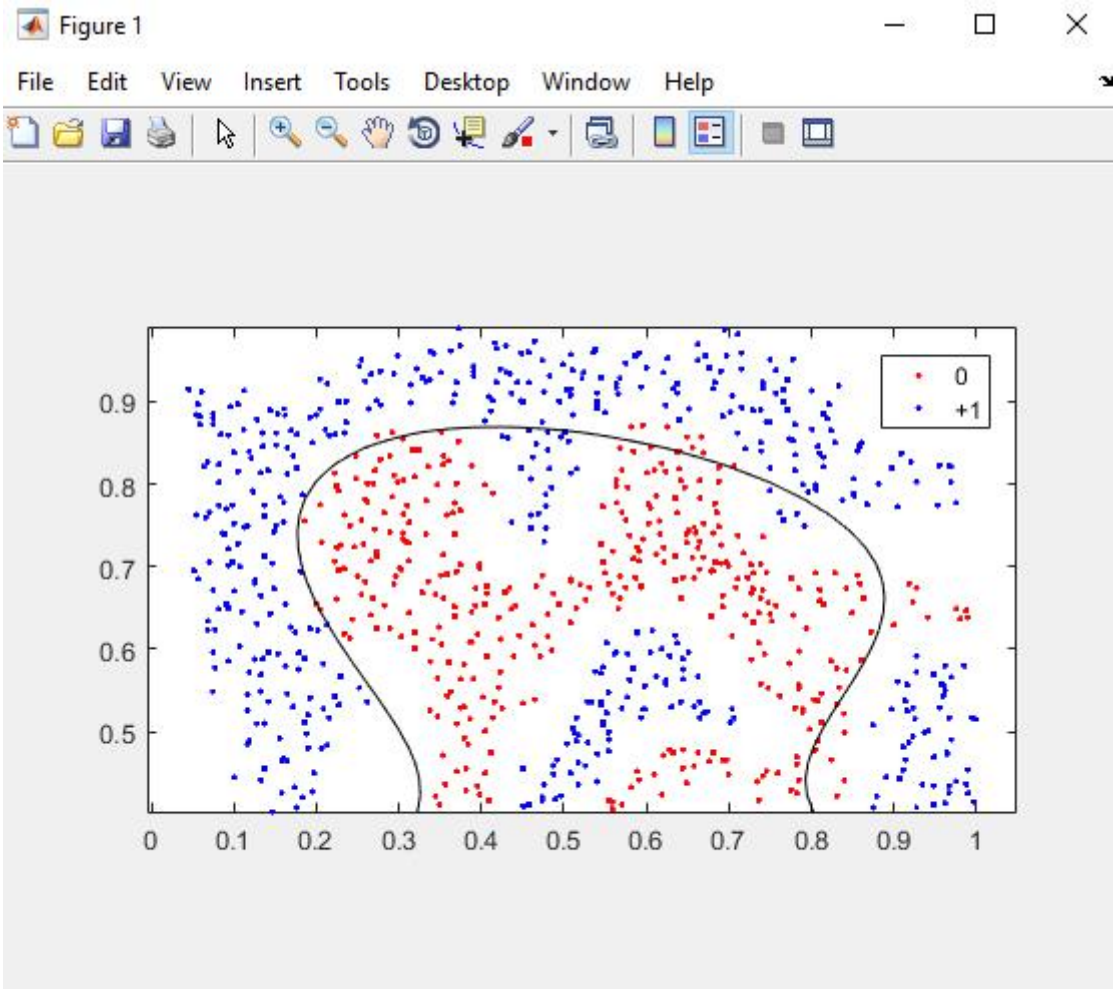
1000----------15.99%

10----------17.50%

As we can see, the value of C becomes larger, and the classification error rate will become smaller.

The visual figure is shown as follows

a.(C=2500):



b.(C=1500)

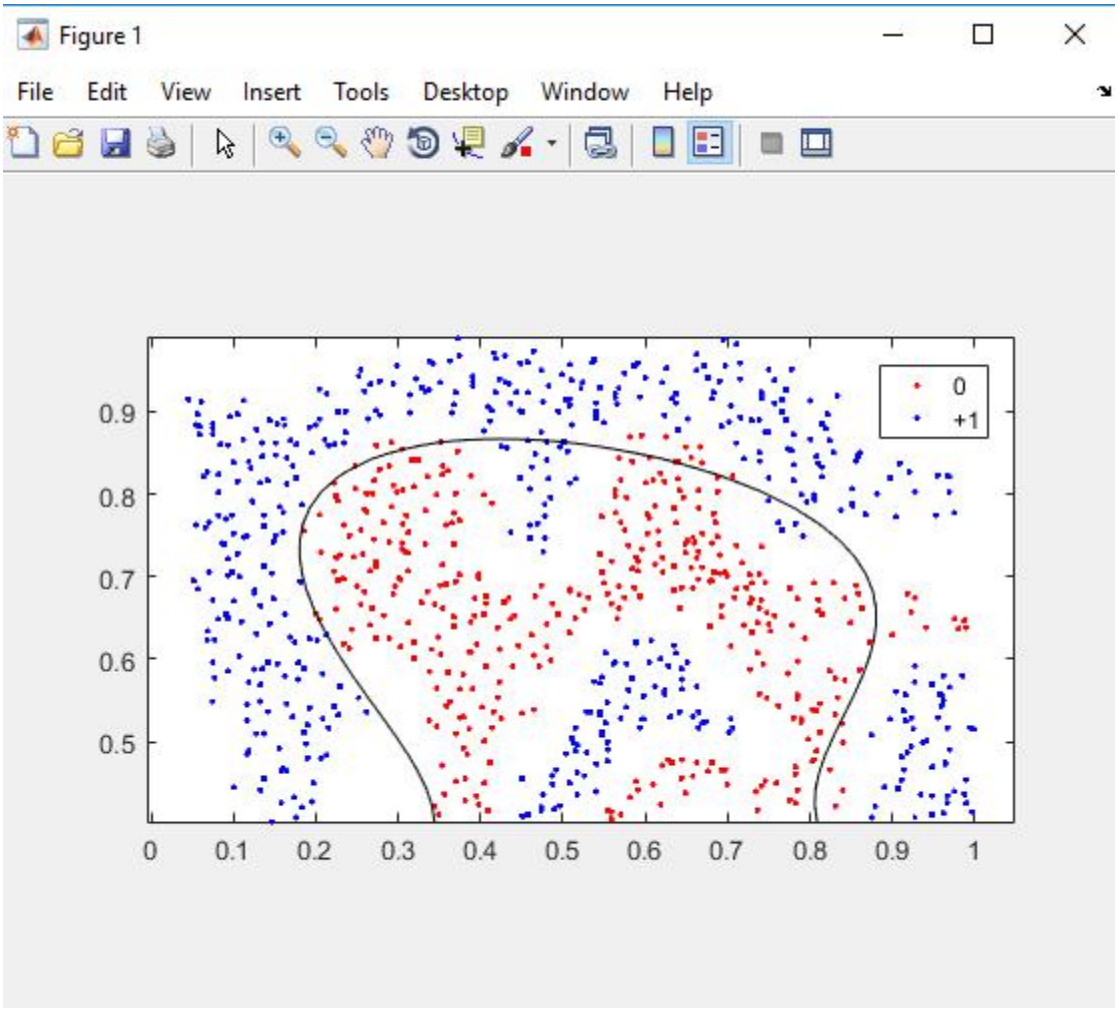


Figure 1
File Edit View Insert Tools Desktop Window Help
0
+1
0.9
0.8
0.7
0.6
0.5
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

c.(C=1000)

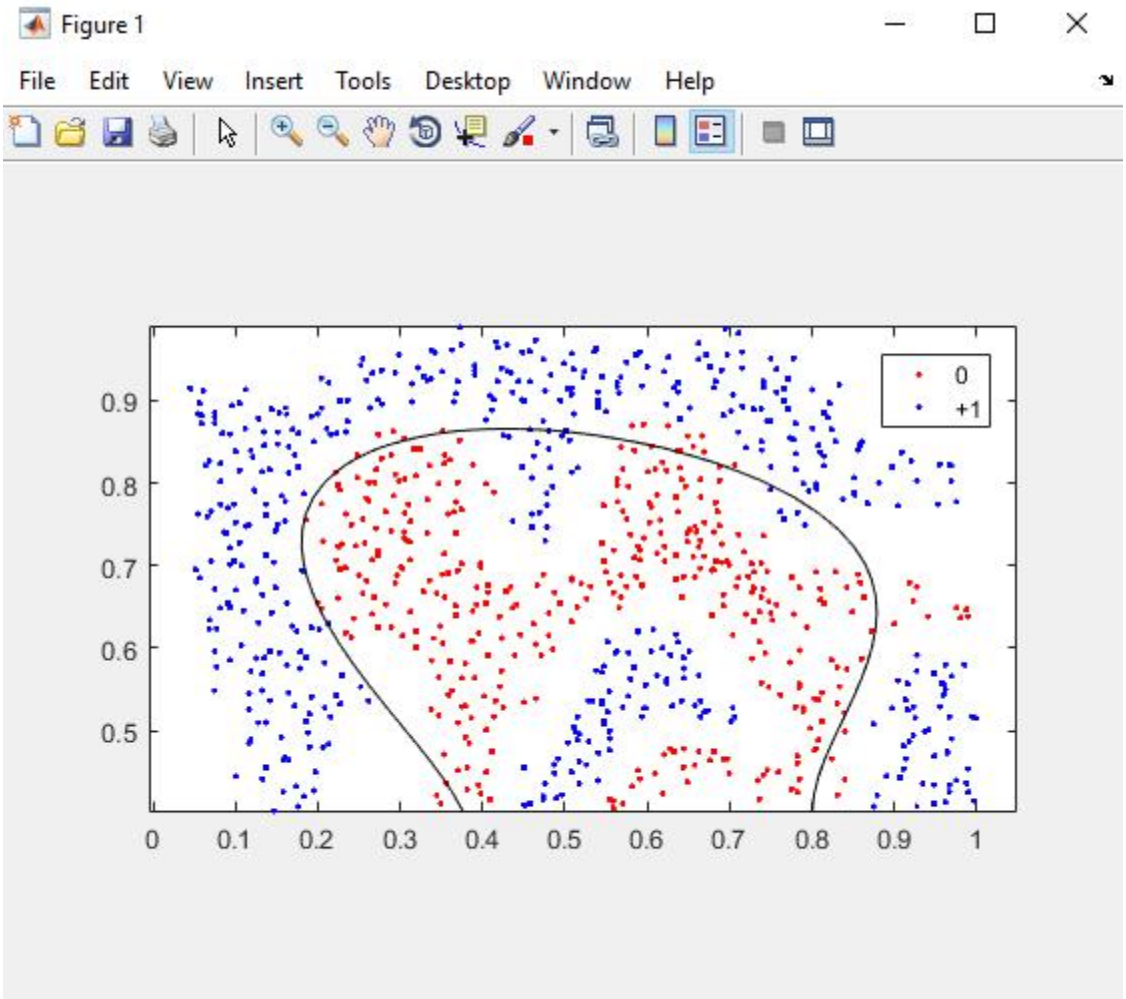
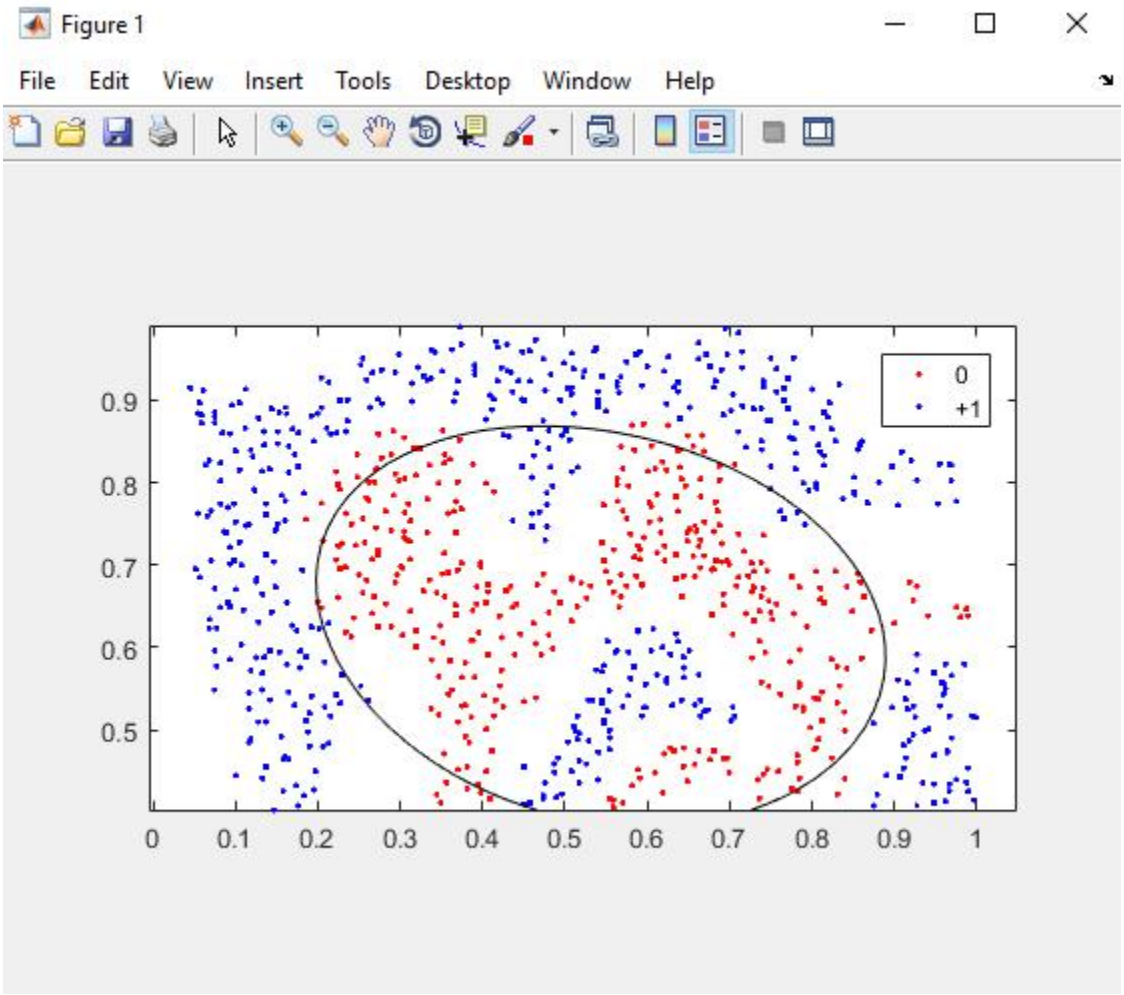


Figure 1
File Edit View Insert Tools Desktop Window Help
0
+1
0.9
0.8
0.7
0.6
0.5
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

d.(C=10)



Summary: A suitable box constraint parameter value C determines the high performance of classification.

(2)Design a custom kernel function and classify the given datasets.

We design a gaussian kernel function to classify the dataset. And you will see the result is shown as follows:

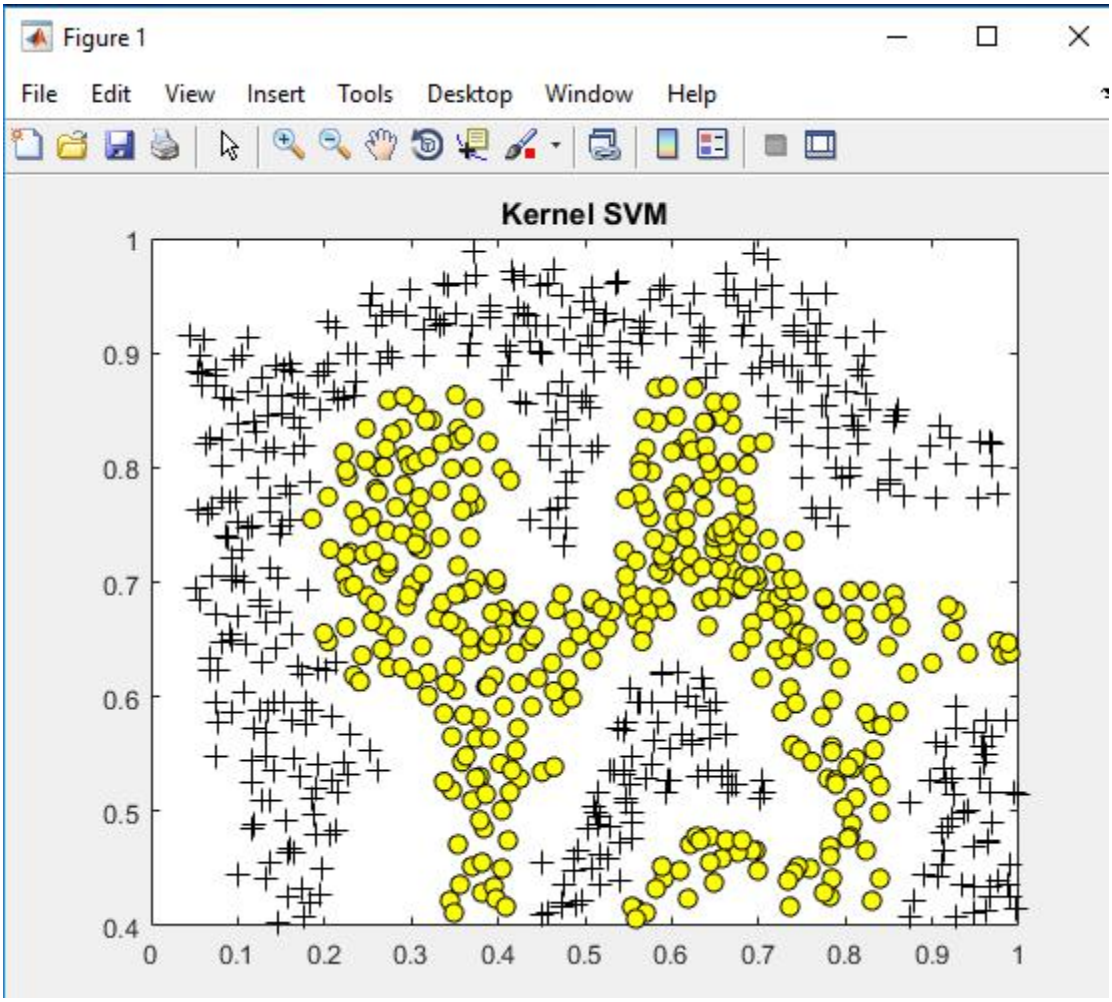
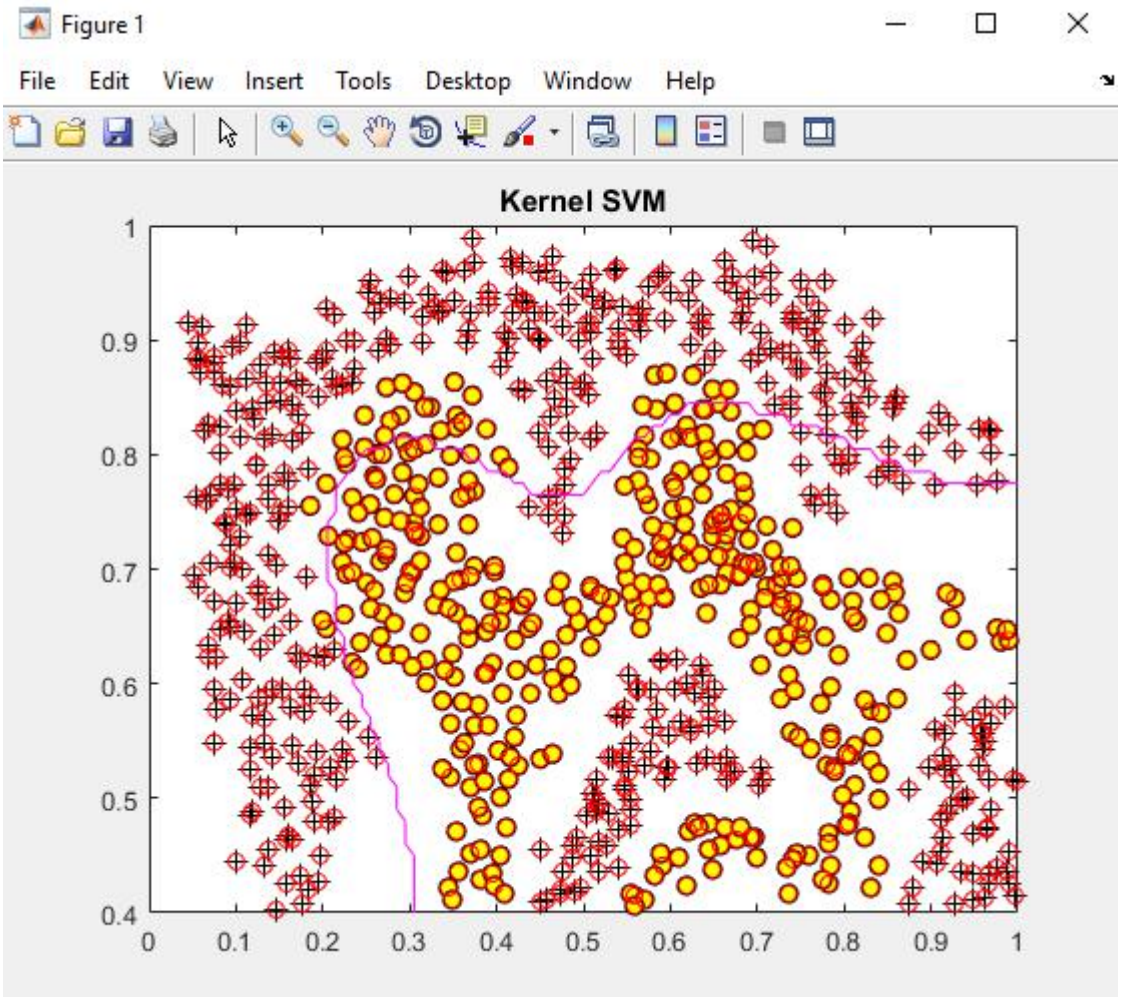


Figure 1
File Edit View Insert Tools Desktop Window Help
Kernel SVM
1
0.9
0.8
0.7
0.6
0.5
0.4
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1



The Matlab code is shown as follows:

“kernelSVM.m”

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
clc
clear all;
close all

dataLength = 2;
dataNumber = [50, 50];

x1 = randn(dataLength, dataNumber(1));
y1 = ones(1, dataNumber(1));

x2 = 3.5 + randn(dataLength, dataNumber(2));
y2 = -ones(1, dataNumber(2));
```

```
load('data1.mat');
figure(1);
pos = find(y == 1); neg = find(y == 0);
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 1, 'MarkerSize', 7)
hold on;
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7)
hold off;

hold on
%axis([-3 8 -3 8]);
title('Kernel SVM')


system('pause');

kernelType = 'gauss';

X=X';
Y=y';
negIdx=find(Y==0);
Y(:,negIdx)=-1;

[alpha, b] = kernelSvmTrainMine(X, Y, kernelType);

epsilon = 1e-5;
indexSV = find(alpha > epsilon);

plot(X(1,indexSV), X(2,indexSV), 'ro');
hold on


 maxX1 = max(X(1,:)); minX1 = min(X(1,:));
maxX2 = max(X(2,:)); minX2 = min(X(2,:));
%axis([minX1 maxX1 minX2 maxX2]);
[x1, x2] = meshgrid(0:0.01:1.0, 0.4:0.01:1.0);
%[x1, x2] = meshgrid(-1:0.1:4, -1:0.1:4);

[rows, cols] = size(x1);
nt = rows * cols;
Xt = [reshape(x1,1,nt); reshape(x2,1,nt)];

Yd = sign(sum(bsxfun(@times, alpha(indexSV) .* Y(indexSV)', kernel(X(:,
indexSV), Xt, kernelType))) + b);

Yd = reshape(Yd, rows, cols);
contour(x1,x2,Yd,[0 0],'m');
hold off;
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

“kernel.m”

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```
function out = kernel(X, Y, Type)

switch Type
    case 'liner'
        out = X' * Y;
    case 'poly'
        out = (1 + X' * Y).^2;
    case 'gauss'
        numberX = size(X,2);
        numberY = size(Y,2);
        tmp = zeros(numberX, numberY);
        for i = 1:numberX
            for j = 1:numberY
                tmp(i,j) = norm(X(:,i)-Y(:,j));
            end
        end
        out = exp(-0.5*(tmp).^2);
    otherwise
        out = 0;
end
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

(3)From what have been studied and figure shown above, it is easy for us to find that the performance of SVM is not good by using it to classify this multicircular or different target mixed regions. So the classification error rate is still very high even though the box constraint C is increased or the gaussian kernel function with smooth capability of separation for nonlinear data. Therefore, as the classes to be classified increase, the more difficult for SVM to classify.

Problem 2:

An LVQ network is trained to classify points on the 2-D grid. The MATLAB code is

```
%% Learning Vector Quantization
% An LVQ network is trained to classify input vectors according to given
% targets.
% Let X be 64 2-element example input vectors and C be the classes these
vectors
% fall into.  These classes can be transformed into vectors to be used as
% targets, T, with IND2VEC.
clear all;
close all;
clc;

x1 =-4:4;
x2 =-4:4;
[X1,X2] = meshgrid(x1,x2);
x_train= [reshape(X1,[1,81]);reshape(X2,[1,81])];
c_train = [3 3 3 3 2 2 1 1 1 3 3 3 3 3 2 1 1 1 3 3 2 1 3 3 1 1 1 3 2 2 1 3 3
1 1 1 3 2 2 2 1 1 3 3 1 3 2 2 2 1 3 2 2 1 3 2 2 1 1 3 2 2 1 1 2 2 3 3 3 2 2 1
1 2 2 3 3 3 2 2 1];
t_train = ind2vec(c_train);


% Here the data points are plotted.  Red = class 3, Blue = class 2; Cyan =
class 1.  The LVQ
% network represents clusters of vectors with hidden neurons, and groups the
% clusters with output neurons to form the desired classes.
for i=1:1:length(c_train)
    XX1 = x_train(1,i);
    XX2 = x_train(2,i);
    if (c_train(i) ==1)
        h(1)= plot(XX1,XX2,'c+','LineWidth',2); % Class 1
        hold on;
    elseif (c_train(i) ==2)
        h(2)= plot(XX1,XX2,'b+','LineWidth',2); % Class 2
        hold on;
    elseif (c_train(i) ==3)
        h(3)= plot(XX1,XX2,'r+','LineWidth',2); % Class 3
        hold on;
    end
end


% Here LVQNET creates an LVQ layer with twelve hidden neurons and a
% learning rate of 0.1.
net = lvqnet(9,0.02);
net = configure(net,x_train,t_train);


% To train the network, first override the default number of epochs, and then
```

```
% train the network.  When it is finished, replot the input vectors '+' and
the
% competitive neurons' weight vectors 'o'. Red = class 3, Blue = class 2;
Cyan = class 1


net.trainParam.epochs=200;
net=train(net,x_train,t_train);

hold on;
Weights = net.IW{1}';
Weigh_c = vec2ind(net.LW{2});
for i=1:1:length(Weigh_c)
    Weights_x1 = Weights(1,i);
    Weights_x2 = Weights(2,i);
    if (Weigh_c(i) ==1)
        h(4)= plot(Weights_x1,Weights_x2,'co','LineWidth',2); % Class 1
        hold on;
    elseif (Weigh_c(i) ==2)
        h(5)= plot(Weights_x1,Weights_x2,'bo','LineWidth',2); % Class 2
        hold on;
    elseif (Weigh_c(i) ==3)
        h(6)= plot(Weights_x1,Weights_x2,'ro','LineWidth',2); % Class 3
        hold on;
    end
end


%% Testing
% Now use the LVQ network as a classifier, where each neuron corresponds to a
% different category.  Present the input vector [0.2; 1].  Blue = class 2;
Cyan = class 1
x1_test =-4:0.5:4;
x2_test =-4:0.5:4;
[X1_test,X2_test] = meshgrid(x1_test,x2_test);
x_test= [reshape(X1_test,[1,289]);reshape(X2_test,[1,289])];
y = net(x_test);
classes = vec2ind(y);
% plot the predicted classes
figure
for i=1:1:length(classes)
    XX1_test = x_test(1,i);
    XX2_test = x_test(2,i);
    if (classes(i) ==1)
        h(7)= plot(XX1_test,XX2_test,'c+','LineWidth',2); % Class 1
        hold on;
    elseif (classes(i) ==2)
        h(8)= plot(XX1_test,XX2_test,'b+','LineWidth',2); % Class 2
        hold on;
    elseif (classes(i) ==3)
        h(9)= plot(XX1_test,XX2_test,'r+','LineWidth',2); % Class 3
        hold on;
    end
end
```

The original 2-D grid is shown below

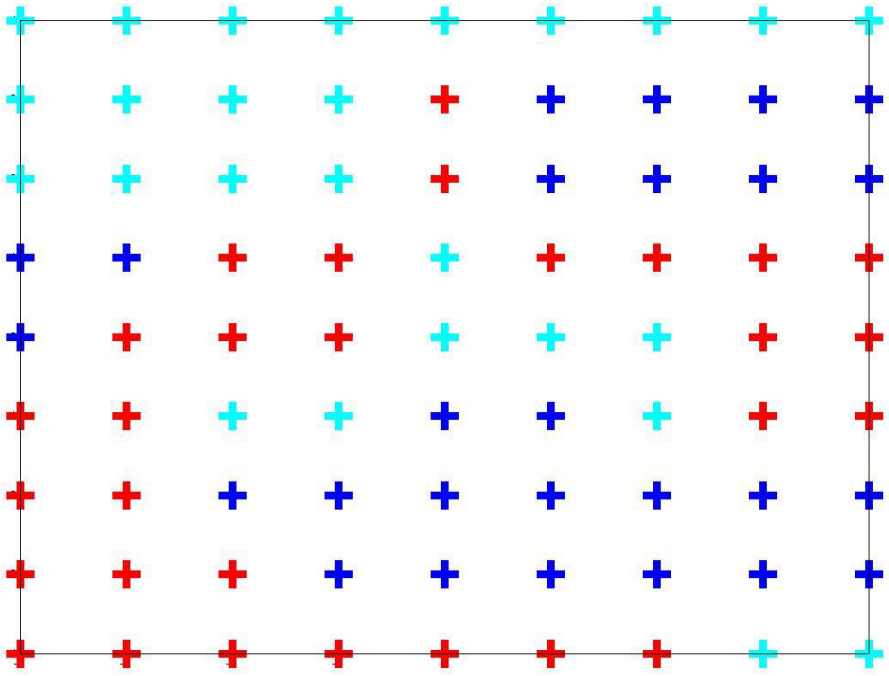


Figure 1. Original 2-D grid

After training the LVQ network. The competitive neurons 'weight vectors' are found to be

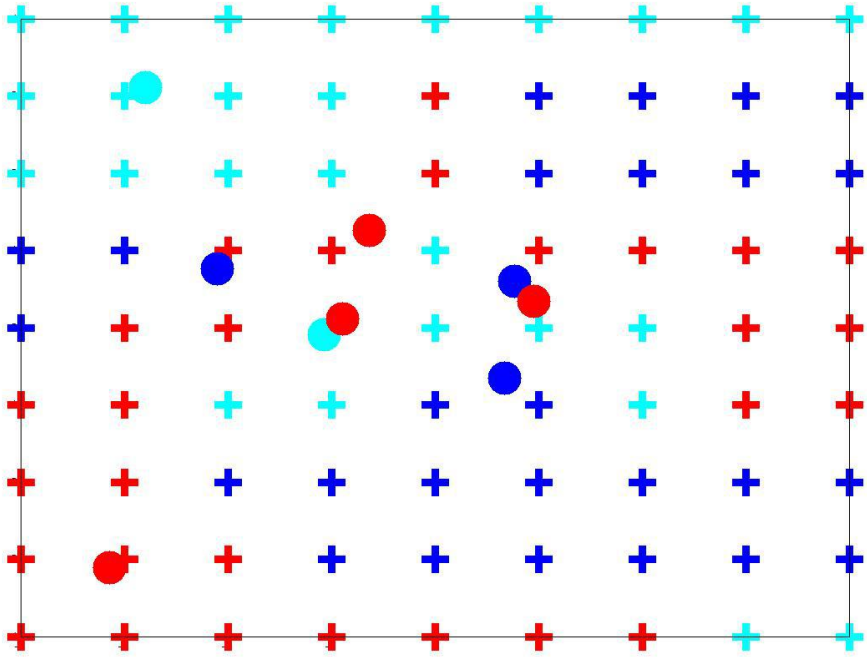


Figure 2. 2-D grid with weight vectors

A finer grid is chosen as the testing data. And the predicted classes using trained LVQ network are shown in the figure

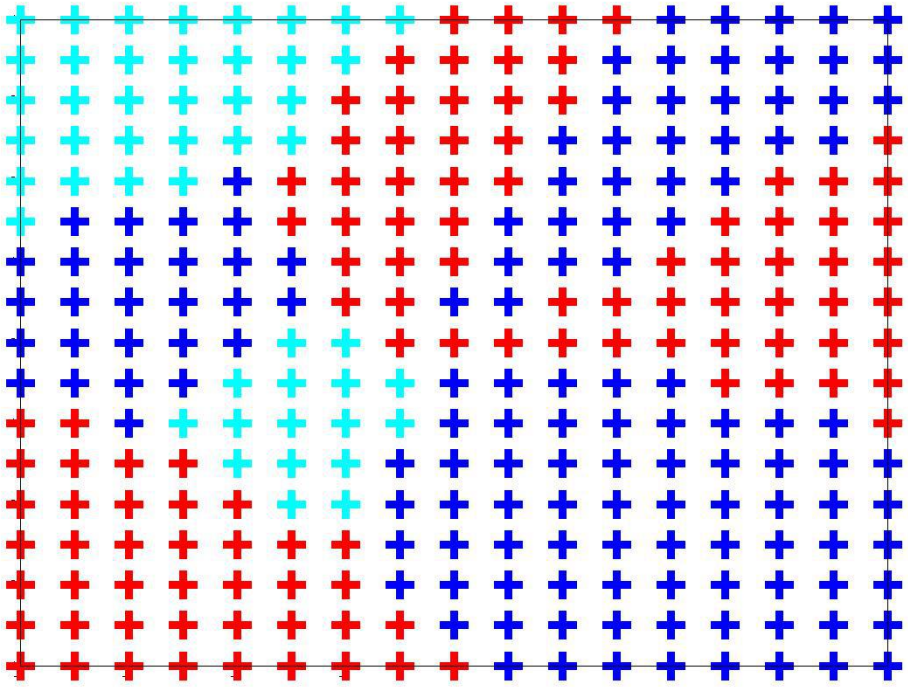


Figure 2. 2-D grid with weight vectors

The results are not good as we expect. The results should be improved if the parameters of LVQ network are adjusted.

<end>