PR Project-1

– Report –

submitted by

Group 18 - M.Navyasri,P.Sruti,Y.Vishnu sreya
S20180020222, S20180020233, S20180020262

November 1, 2020

1 Introduction

The main objective of this project is to perform classification on Given data
Qns-1 : Non-linear transformation is used to convert multi-dimensional feature data into one-dimensional data using a suitable distance metric so that we can use a simple linear regression model to fit the data
Qns-2 : In Direct non-linear classification, we can fit multi-dimensional data into a classification model without even changing the data into one-dimensional data

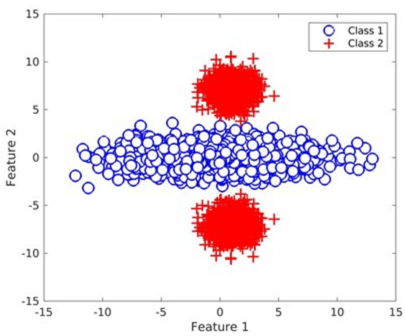
2 problem Description

2.1 Qns1

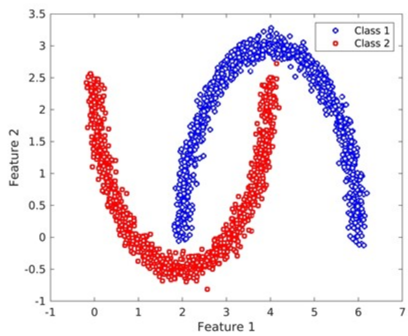
Given the following parameters we had to generate a 2D Data set consisting of 2 classes
For class 1 : given co-variance matrix = [15 0 ; 0 1]
For class 2 : given co-variance matrix = [1 0;0 1]
We have to transform the data into 1D feature data and build a suitable model to predict the classes of data.

2.2 Qns2

Given the following parameters we had to generate a 2D Data set consisting of 2 classes
For class 1 : co-variance matrix = 0.1 ,mean vector = = [h1 +acost,bsint]T
For class 2 : mean vector = [h2 + acost,k bsint)]T and co-variance vector = 0.1
The data is fitted in linear discriminant,Quadratic discriminant,SVM with Gaussian kernel,SVM with polynomial kernel , KNN classifier



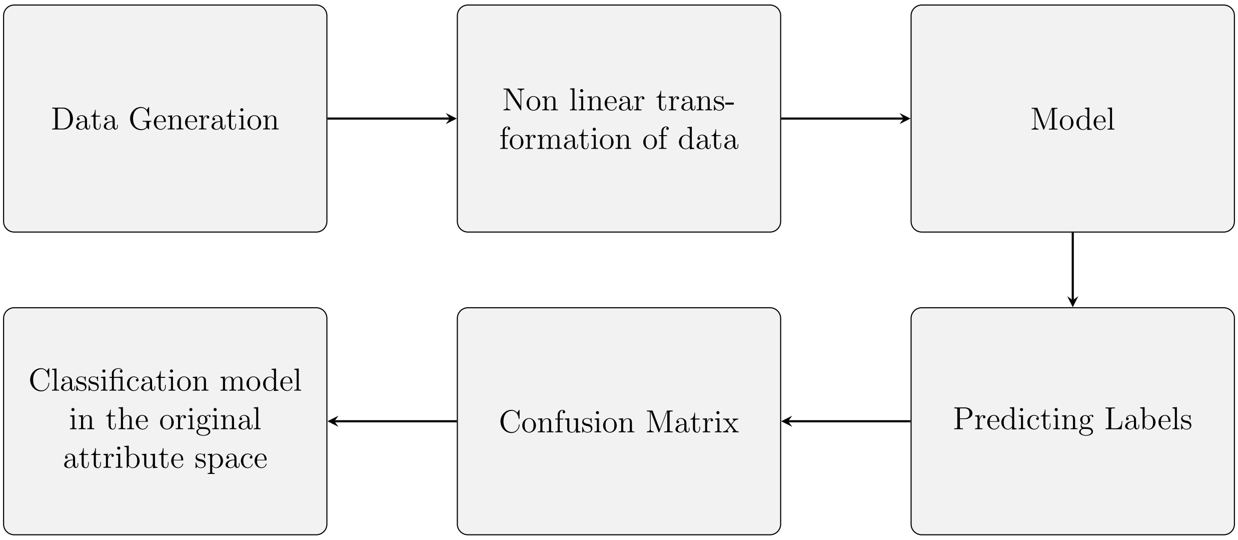
Divider Patterns - Question-1.



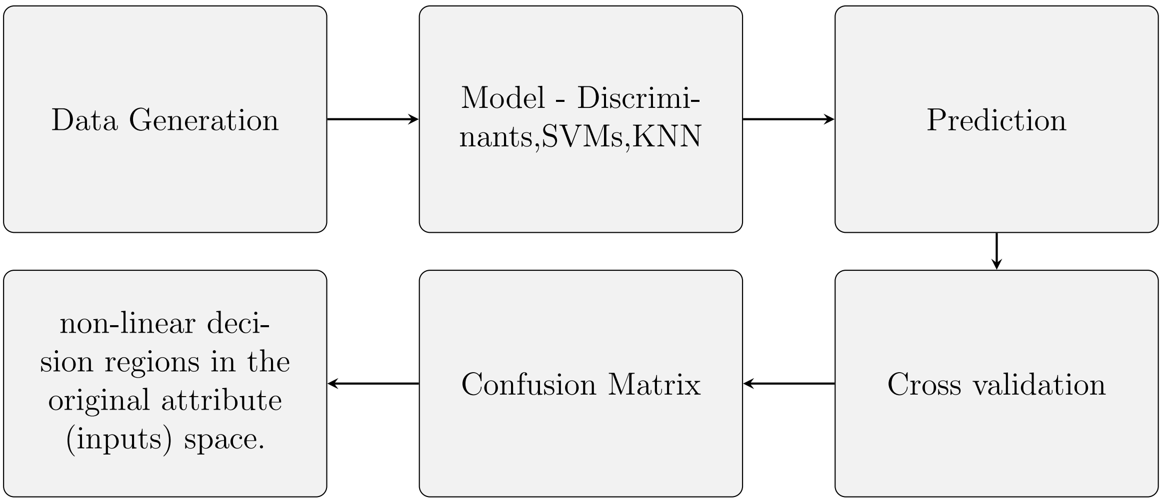
Interlocking Sinusoids - Question-2.

3 Methodology

3.1 Qns1

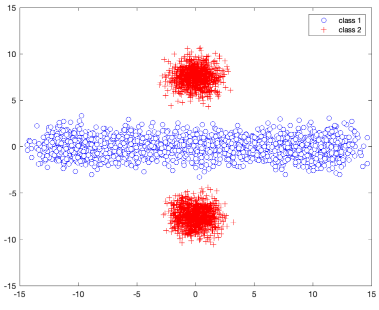


3.2 Qns2

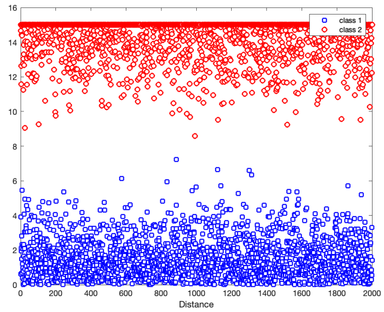


4 Implementation and Results

4.1 Qns1



Initial data generation.



data after non-linear transform

- The non linear transform is,

$$lx1 = \sum |x - [0, 7.5]| \tag{1}$$

$$lx2 = \sum |(x - [0, -7.5]| \tag{2}$$

$$lx = |lx2 - lx1| \tag{3}$$

$$ly1 = \sum |(y - [0, 7.5]| \tag{4}$$

$$ly2 = \sum |(y - [0, -7.5]| \tag{5}$$

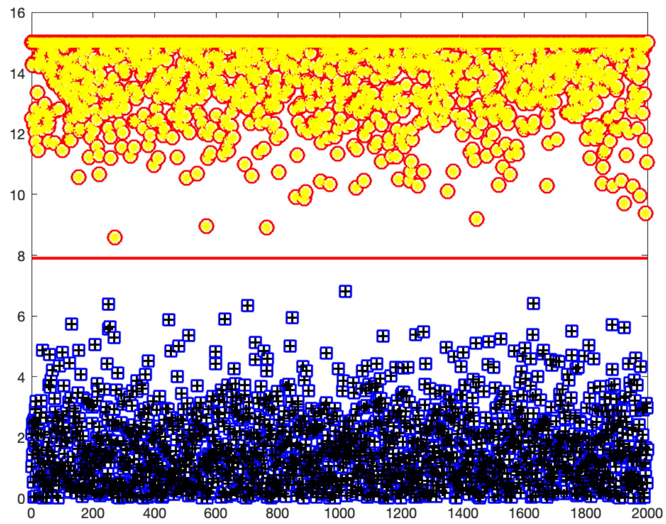
$$ly = |ly1 - ly2| \tag{6}$$

- Classes of the new transformed model are predicted using Least squares linear regression.

- Confusion Matrix and accuracy:

$$\begin{bmatrix} 1999 & 1 \\ 0 & 2000 \end{bmatrix}$$

accuracy=99.98

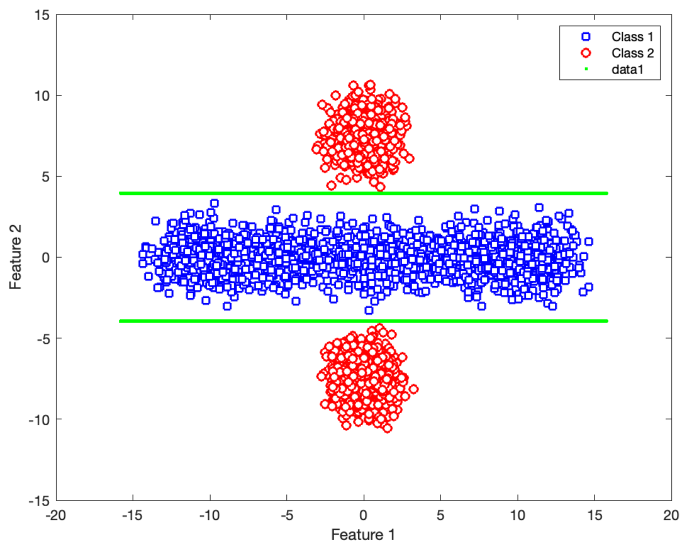


Predicted Classes.

- Plotting in Original Feature Space

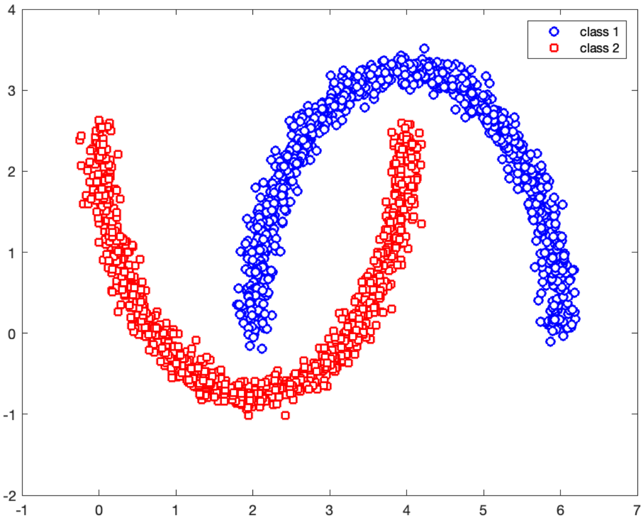
$$||z1| + |z2 - 7.5| - (|z1| + |z2 + 7.5|)| = thr \tag{7}$$

$$Z2 = +/-(thr/2) \tag{8}$$



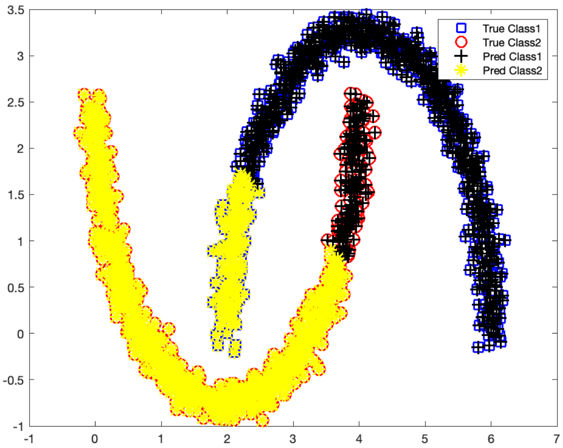
Predicted Classes.

4.2 Qns2

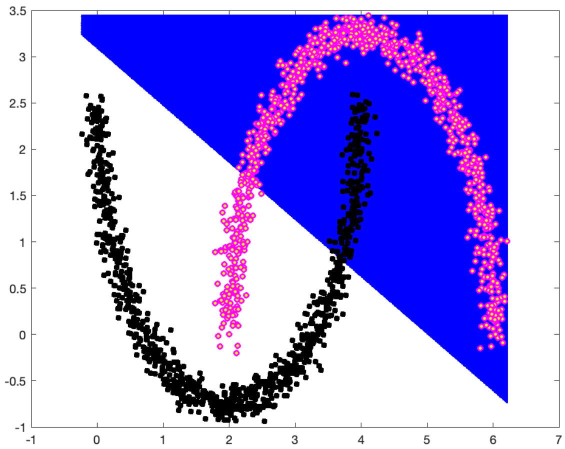


Initial data generation.

4.2.1 linear discriminant kernel

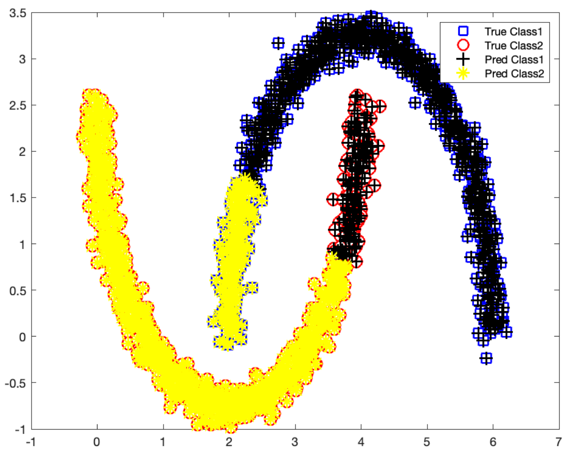


Predicted classes for Linear Discriminant.

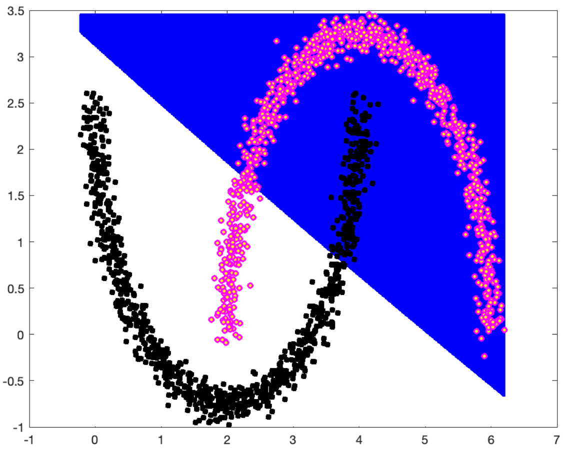


Non-linear Decision region.

4.2.2 Quadratic discriminant kernel

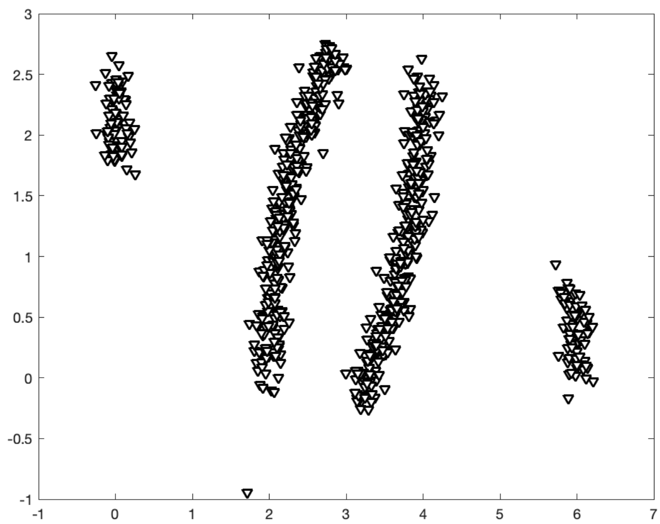


Predicted classes for Quadratic Discriminant kernel

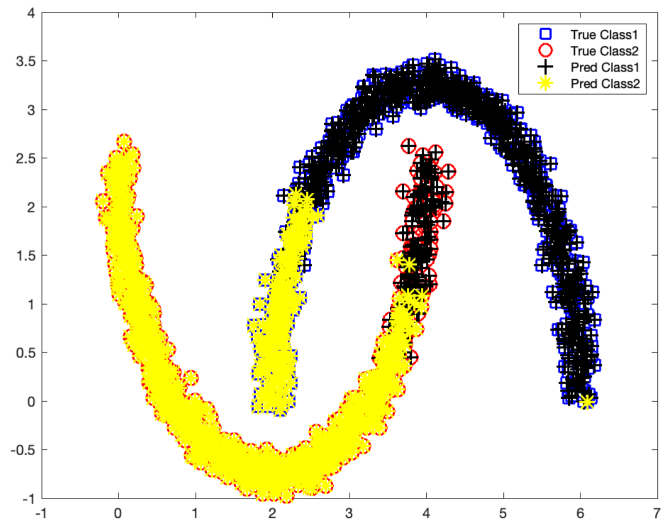


Non-linear decision region.

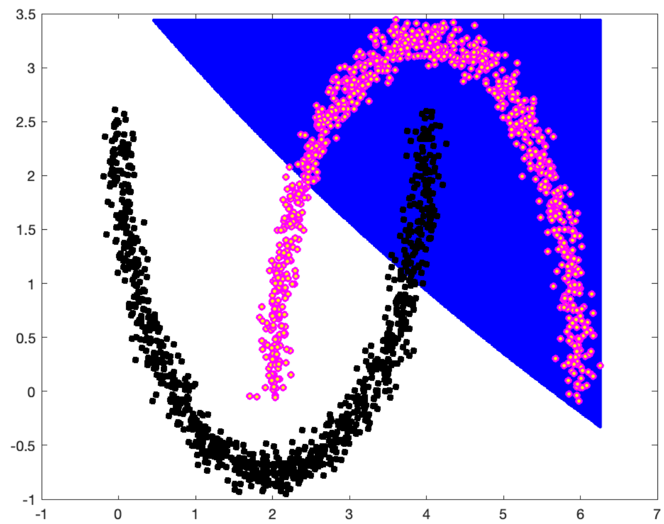
4.2.3 SVM with Ploynomial (Quadratic) Kernel



support vectors for Quadratic kernel

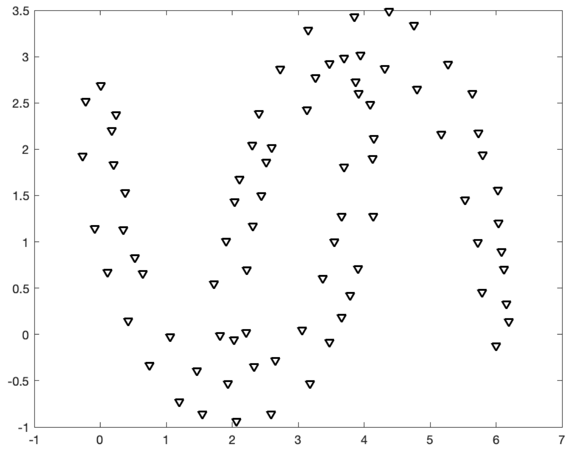


Predicted Classes for Quadratic kernel.

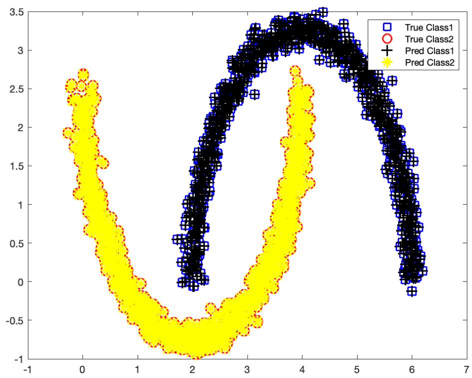


Non-linear Decision space.

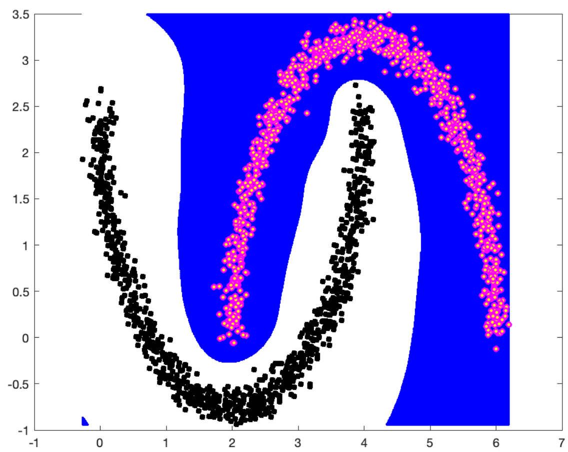
4.2.4 SVM with a Gaussian Kernel



support vectors for Gaussian(fine) kernel.

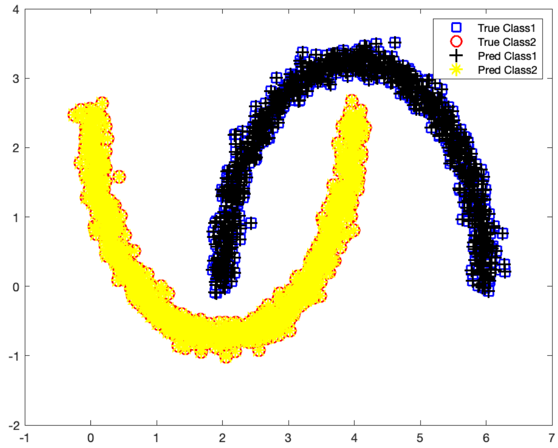


predicted classes for Gaussian(fine) kernel.

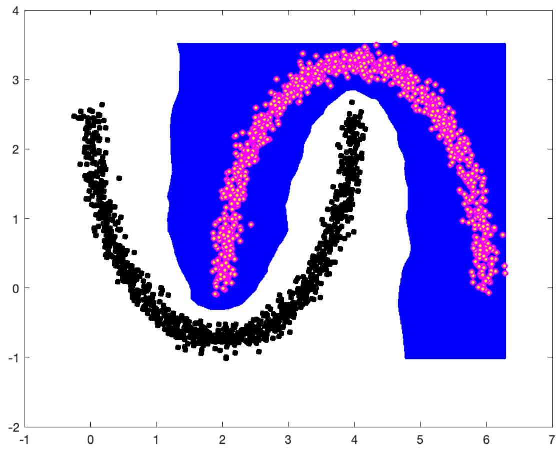


Non-Linear decision region.

4.2.5 K-nearest Neighbour kernel



Predicted classes for K-Nearest Neighbours.



Non-linear Decision Region.

5 Analysis

5.1 Qns2

Here is the Model accuracy analysis of the above 5 Models,

classification Model	Confusion Matrix	Accuracy(%)
Linear Discriminant	$\begin{bmatrix} 828 & 172 \\ 183 & 817 \end{bmatrix}$	82.25
Quadratic Discriminant	$\begin{bmatrix} 828 & 172 \\ 174 & 826 \end{bmatrix}$	82.7
SVM-Gaussian(fine)	$\begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$	100
SVM-Quadratic	$\begin{bmatrix} 819 & 181 \\ 185 & 815 \end{bmatrix}$	81.7
K-Nearest neighbours	$\begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$	100

From the above table we can observe that SVM with Gaussian kernel ad K-Nearest neighbours gives you the best accuracy of 100 that is model classifies the data into 2 classes with 100percent accuracy

If we keep the best model for data with given parameters, the order would be like

1. SVM-Gaussian - 100
2. K-Nearest neighbours - 100
3. Quadratic discriminant - 82.7
4. Linear discriminant - 82.25
5. SVM-Quadratic - 81.7

6 MATLAB CODES

6.1 Qns 1

6.1.1 Data Generation

```
clear;
clc;
close all;
N = 2000;
h = 0;k = 0;a=12;b = 0;r = 2.5;
th = (1:N)*2*pi/N;
th = th(:);
xunit = h + a*cos(th);
yunit = k + b*sin(th);
plot(xunit,yunit);
figure;
%class 1
%mu1 is [h+a*cos(th),k+b*sin(th)]
sigma1 = [15 0;0 1];
x = [randn(N,1)+h+a*cos(th),randn(N,1)+k+b*sin(th)];
plot(x(:,1),x(:,2),'bo','MarkerFaceColor','w');
hold on;
%class 2
%mu2 is [0,+/-7.5]
sigma2 = [1 0 ;0 1];
y1 = [0 + randn(N/2,1),7.5 + randn(N/2,1)];
y2 = [0 + randn(N/2,1),-7.5 + randn(N/2,1)];
y = [y1;y2];
plot(y(:,1),y(:,2),'r+','MarkerFaceColor','w');
legend('class 1','class 2');
%nonlinear transformation
lx1 = (sum(abs(x-[0,7.5])'));
lx2 = (sum(abs(x-[0,-7.5])'));
lx = abs(lx2 - lx1);
ly1 = (sum(abs(y-[0,7.5])'));
ly2 = (sum(abs(y-[0,-7.5])'));
```

```
ly = abs(ly1 - ly2);
figure;
plot(lx,'bs','LineWidth',1.5,'MarkerFaceColor','W');
hold on;
plot(ly,'ro','LineWidth',1.5,'MarkerFaceColor','W');
hold on;
xlabel('Distance');
legend('class 1','class 2');
```

6.1.2 Model Training and Predicting Labels

```
%model
Z1(1:2:2*N-1,:) = x;
Z1(2:2:2*N,:) = y;
 TestTarg1(1:2:2*N-1) = 1;
 TestTarg1(2:2:2*N) = -1;
 vx = var(x);
 vy = var(y);
T = TestTarg1;
%PhiZ1 = (sum(abs(Z1-[0,7.5])'));
%PhiZ2 = (sum(abs(Z1-[0,-7.5])'));
%PhiZ = abs(PhiZ1 - PhiZ2);
PhiZ = [lx;ly];
Xmat = [ones(size(Z1,1),1) PhiZ(:)];
W_ls = regress(T(:),Xmat);
Y_x = Xmat*W_ls;
thr = -W_ls(1)/W_ls(2);

%predicting labels
pred_labels = ones(size(T));
pred_labels(Y_x < 0) = 2;
T(T == -1) = 2;
figure;
 plot(lx,'bs','LineWidth',1.5,'MarkerSize',10,'
    MarkerFaceColor','w');
 hold on;
```

```
plot(ly,'ro','LineWidth',1.5,'MarkerSize',10,'MarkerFaceColor','w');
plot(PhiZ(pred_labels ==1),'k+','LineWidth',1.5,'MarkerFaceColor','w');
plot(PhiZ(pred_labels ==2),'y*','LineWidth',1.5,'MarkerFaceColor','w');
plot(thr*ones(N),'r','LineWidth',2);
hold off;

%Confusion matrix
ConfMat = confusionmat(T,pred_labels);
disp(ConfMat);
acc = sum(diag(ConfMat))/sum(sum(ConfMat));
disp(acc);
```

6.1.3 Plotting the decision boundaries in original attribute Space

```
%  phiz1 = |z1-0| + |z2-7.5| , phiz2 = |z1-0| + |z2-(-7.5)|
%  phiz = |phiz1 - phiz2| = ||z2 - 7.5| - |z2 + 7.5|| = thr
%by solving ||z2 - 7.5| - |z2 + 7.5|| = thr ,we get z2 = +/-thr/2
%so here z1 can be anything from the given equation since it is not there
%in the euation
mu = mean(Z1);
%z1vec = min(min([x(1,:),y(1,:)])):0.01:max(max([x(1,:),y(1,:)]));
z1vec = -2*thr:0.01:2*thr;
ix = 1;
for zx = 1:length(z1vec)
 z1 = z1vec(zx);
 z2 = thr/2;
 model(ix,:) = [z1,z2];
 ix = ix + 1;
 z2 = -thr/2;
 model(ix,:) = [z1,z2];
 ix = ix+1;
```

```
end
figure;
plot(x(:,1),x(:,2),'bs',y(:,1),y(:,2),'ro','LineWidth',1.5,'
    MarkerFaceColor','w');
xlabel('Feature 1');
ylabel('Feature 2');
legend('Class 1','Class 2');
hold on;
plot(model(:,1),model(:,2),'g.','LineWidth',2);
```

6.2 Qns 2

6.2.1 Generating and plotting data

```
clear
clc
close all
N = 1000;
h1 = 4;k = 2.5;a=2;b = 3.25;h2 = 2;
th = (1:N)*pi/N;
th = th(:);
xunit = h1 + a*cos(th);
yunit = b*sin(th);
plot(xunit,yunit);
hold on;
xunit = h2 + a*cos(th);
yunit = k-b*sin(th);
plot(xunit,yunit);
figure;
%class 1
x = [0.1*randn(N,1)+h1+a*cos(th),0.1*randn(N,1)+b*sin(th)];
plot(x(:,1),x(:,2),'bo','LineWidth',1.5,'MarkerFaceColor','w
    ');
hold on;
%class 2
y = [0.1*randn(N,1)+h2+a*cos(th),0.1*randn(N,1)+k-b*sin(th)
    ];
```

```
plot(y(:,1),y(:,2),'rs','LineWidth',1.5,'MarkerFaceColor','w
    ');
legend('class 1','class 2');

t1 = ones(N,1);
t2 = -ones(N,1);
T = [t1;t2];
rin=randperm(length(T));
T=T(rin);
one_vec = ones(2*N,1);
xmat1 = [x;y];
xmat = xmat1(rin,:);
X = xmat;

trainingData = [X T];

predictors = trainingData(:,1:end-1);
response = trainingData(:,end);
```

6.2.2 Linear Discriminant Model

```
inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3'});

predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];
% Train a classifier
classificationDiscriminant = fitcdiscr(...
    predictors, ...
    response, ...
    'DiscrimType', 'linear', ...
    'Gamma', 0, ...
    'FillCoeffs', 'off', ...
    'ClassNames', [-1; 1]);

```

```
% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
discriminantPredictFcn = @(x) predict(classificationDiscriminant, x);
trainedClassifier.predictFcn = @(x) discriminantPredictFcn(predictorExtractionFcn(x));
% Add additional fields to the result struct
trainedClassifier.ClassificationDiscriminant = classificationDiscriminant;
%model
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3'});

predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];
% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationDiscriminant, 'KFold', 5);
% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

6.2.3 Quadratic Discriminant Model

```
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3'});

predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];
```

```
% Train a classifier
% This code specifies all the classifier options and trains
    the classifier.
classificationDiscriminant = fitcdiscr(...
     predictors, ...
     response, ...
     'DiscrimType', 'quadratic', ...
     'FillCoeffs', 'off', ...
     'ClassNames', [-1; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames'
    , predictorNames);
discriminantPredictFcn = @(x) predict(
    classificationDiscriminant, x);
trainedClassifier.predictFcn = @(x) discriminantPredictFcn(
    predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationDiscriminant =
    classificationDiscriminant;
% Convert input to table
inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3'});
predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];
% Perform cross-validation
partitionedModel = crossval(trainedClassifier.
    ClassificationDiscriminant, 'KFold', 5);
% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(
    partitionedModel);
% Compute validation accuracy
```

```
validationAccuracy = 1 - kfoldLoss(partitionedModel, '
    LossFun', 'ClassifError');
```

6.2.4 Support Vector Machines with Gaussian Kernel

```
%SVM with gaussian kernel
classificationSVM = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 0.35, ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', [-1; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames'
    , predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(
    predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationSVM = classificationSVM;

% model.
inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3'});

predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];

% Perform cross-validation
```

```
partitionedModel = crossval(trainedClassifier.
    ClassificationSVM, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(
    partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, '
    LossFun', 'ClassifError');
```

6.2.5 SVM with Polynomial Kernel

```
%SVM with polynomial kernal of degree 1
 classificationSVM = fitcsvm(...
     predictors, ...
     response, ...
     'KernelFunction', 'linear', ...
     'PolynomialOrder', [], ...
     'KernelScale', 'auto', ...
     'BoxConstraint', 1, ...
     'Standardize', true, ...
     'ClassNames', [-1; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames'
    , predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(
    predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationSVM = classificationSVM;

% model.
inputTable = array2table(trainingData, 'VariableNames', {'
    column_1', 'column_2', 'column_3'});
```

```
predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.
    ClassificationSVM, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(
    partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, '
    LossFun', 'ClassifError');
```

6.2.6 K-Nearest Neighbours

```
%Fine KNN
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Euclidean', ...
    'Exponent', [], ...
    'NumNeighbors', 1, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', [-1; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames'
    , predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x) knnPredictFcn(
    predictorExtractionFcn(x));
```

```
% Add additional fields to the result struct
trainedClassifier.ClassificationKNN = classificationKNN;
%model
inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3'});

predictorNames = {'column_1', 'column_2'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_3;
isCategoricalPredictor = [false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationKNN, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

6.2.7 Confusion Matrix and Accuracy

```
pred_labels(pred_labels == -1) = 2;
TestTarg = T;
TestTarg(T == -1) = 2;
%ConfMat = ConfusionMatrix2(pred_labels,TestTarg,2);
ConfMat = confusionmat(TestTarg,pred_labels);
disp(ConfMat);
acc = sum(diag(ConfMat))/sum(sum(ConfMat));
disp(acc);
```

6.2.8 support vectors plotting for SVM

```
figure;
```

```
plot(x(:,1),x(:,2),'bs','linewidth',1.5,'MarkerSize',10,'MarkerFaceColor','w');
hold on;
plot(y(:,1),y(:,2),'ro','linewidth',1.5,'MarkerSize',10,'MarkerFaceColor','w');

SVInx = trainedClassifier.ClassificationSVM.IsSupportVector;
%%

plot(X(SVInx == 1,1),X(SVInx == 1,2),'kv','linewidth',1.5,'MarkerFaceColor','w');
```

6.2.9 Decision region plotting for SVM, KNN,and Discriminant models

```
x1range = min(min([x(:,1),y(:,1)])):.01:max(max([x(:,1),y(:,1)]));
x2range = min(min([x(:,2),y(:,2)])):.01:max(max([x(:,2),y(:,2)]));
[xx1, xx2] = meshgrid(x1range,x2range);
XGrid = [xx1(:) xx2(:)];
predictedspecies = predict(trainedClassifier.ClassificationKNN,XGrid);
%in case of discriminant models,
%predict(trainedClassifier.ClassificationDiscriminant,XGrid);
%in case of SVM models,
%predict(trainedClassifier.ClassificationSvm,XGrid);
figure;
gscatter(xx1(:),xx2(:),predictedspecies,'wbr');
hold on;
plot(x(:,1),x(:,2),'mo','linewidth',1.5,'MarkerSize',4,'MarkerFaceColor','y');
hold on;
plot(y(:,1),y(:,2),'ks','linewidth',1.5,'MarkerSize',4,'MarkerFaceColor','k');
hold off;
legend off;
```

You can find all codes and output images HERE