SYDE 372 – Lab 1

Pattern Recognition

Technical Lab Report

by

Jason Dunham - 20317166

Brian Wildsmith - 20354178

Majd Al-shihabi - 20245905

A technical lab report

presented to the University of Waterloo

SYDE 372 – Lab 1

Waterloo, Ontario, Canada, 2014

**Table of Contents**

Table of Figures 3

1.0 Introduction 4

2.0 Implemention 4

2.1 MATLAB Implementation 4

2.2 Generating Clusters 4

2.3 Classifiers 6

3.0 Error Analysis: 8

4.0 Conclusions: 9

5.0 Appendix – MATLAB Code 10

5.1 FeatureClass.m 10

5.2 Lab1.m 10

5.3 Functions.m 16

5.4 plot\_ellipse.m 29

# Table of Figures

Figure 1 – Case 1 Distributions and Unit Contours 5

Figure 2 - Case 2 Distributions and Unit Contours 5

Figure 3 - Decision Boundaries for Case 1 6

Figure 4 - Decision Boundaries for Case 2 7

# Table of Tables

Table 1: probability distributions for all algorithms 8

Lab 1 – Introduction

# 1.0 Introduction

In this MATLAB laboratory, several pattern recognition functions will be explored and the error functions of each will be compared. The following functions will be evaluated:

* MED
* GED
* MaP
* NN
* kNN (with a k value of 5)

The following components were characterized during this laboratory:

# 2.0 Implemention

## 2.1 MATLAB Implementation

Each of the functions were compared using scripts written in MATLAB. The scripts utilized can be found in Appendix A. A general script called lab1.m was written and used to control the general flow of the application. A separate Functions class was utilized to store all of the functions required to calculated Euclidean distances, and perform all boundary analysis.

A class called FeatureClass.m was written to represent the concept of a feature space. This allowed each feature to be easily passed around between functions. Finally, a function called plot\_ellipse.m, which was provided was modified to perform the required ellipse drawing functionality.

Each of the four .m files described above can be found in the appendix of this report.

## 2.2 Generating Clusters

For each of the two cases, random clusters were generated in MATLAB. Unit Standard Deviation contours were also generated. The plots below demonstrate these cases:

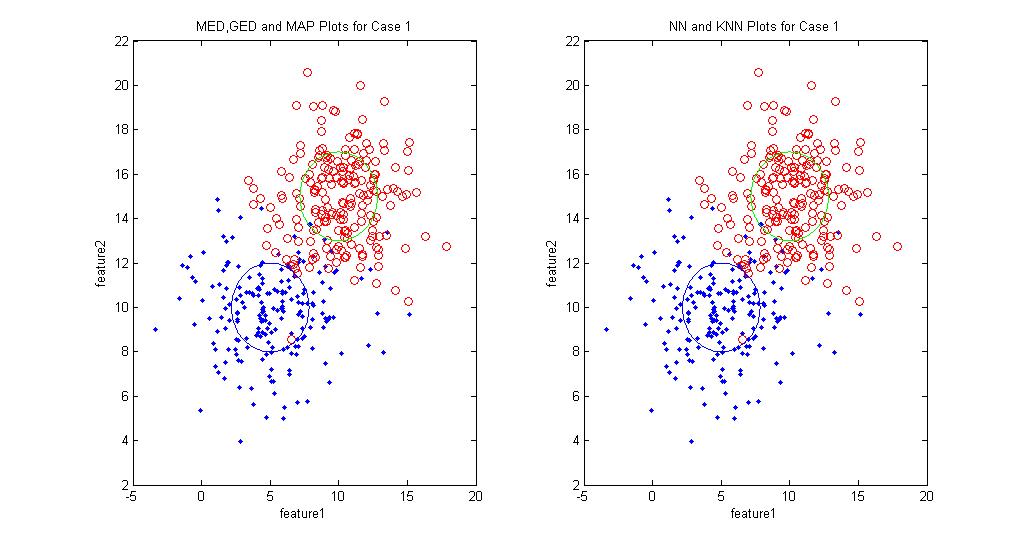


Figure – Case 1 Distributions and Unit Contours

In Figure 1, the graphs demonstrate Case 1. The blue dots represent the distribution for Class 1, while the red dots represent class 2.

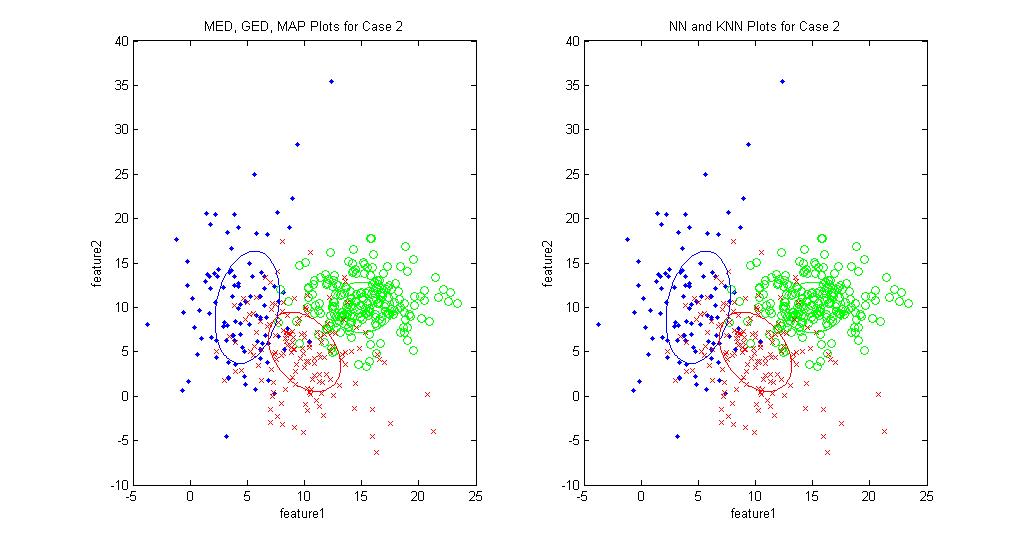


Figure - Case 2 Distributions and Unit Contours

Figure 2 demonstrates a sample distribution for Case 2, along with the Unit Standard Deviation contours. In both cases, the unit contours relate to the clustered data by visually showing the general shape of the data. In Case 1 and Case 2, there is no correlation between features, and as a result the unit contours are circular (on some plots they appear oval because the x and y scales do not match). In Cases 3, 4 and 5, the unit standard deviation contours are ovals with varying shape, orientation and volume, depending on the statistical properties of it’s underlying cluster data. Given that certain classes contain more sample points then other classes, the number of points for each class varies.

## 2.3 Classifiers

Figures 3 and 4 show the MED, GED, MAP, NN and KNN decision boundaries for each of the cases. Cases 1 and 2 are plotted on the same plots, while Cases 3, 4 and 5 are plotted on a separate set of plots.

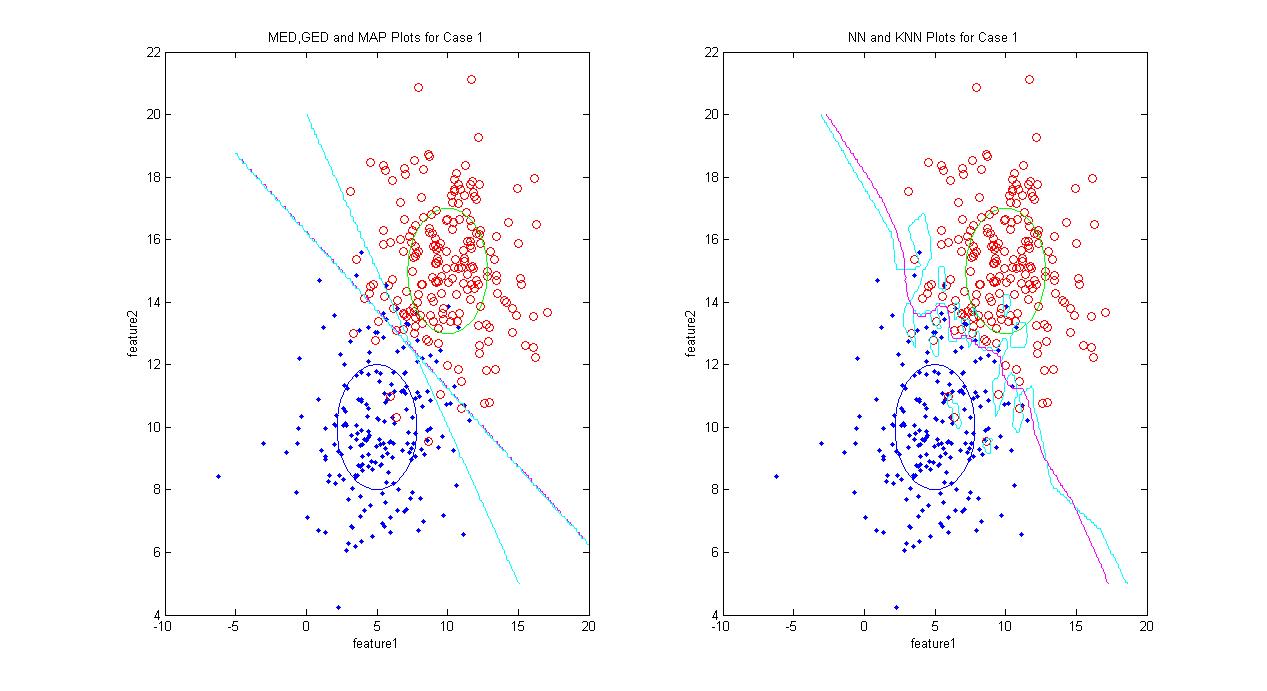


Figure - Decision Boundaries for Case 1

In the MED, GED, MAP plot in figure 3, the MED decision boundary is coloured Cyan, the GED plot is coloured in Magenta, and the MAP plot is coloured in Black. On the NN, KNN plots, the NN boundary is coloured in Cyan while the KNN plot is coloured in Magenta.

These boundaries each display as expected. In the MED, GED, MAP plot, it can be observed that the MED curve is least able to correctly fit the data. This is consistent with theory, as it does not take the correlation matrix into account. The GED and MAP boundaries are identical for Case 1. This is expected, as the a priori information is equal (50% previous probably of each class), and thus we would not expect that the MAP boundary would appear any different from the GED boundary.

In the NN/KNN Boundary, we can observe that while both boundaries follow a similar pattern, the NN boundary (Cyan) weaves around every single point, while the KNN Boundary (magenta) has an averaging effect. This is expected, as the KNN boundary takes into account the average of the closest 5 points for each distribution, making it less susceptible to noise.

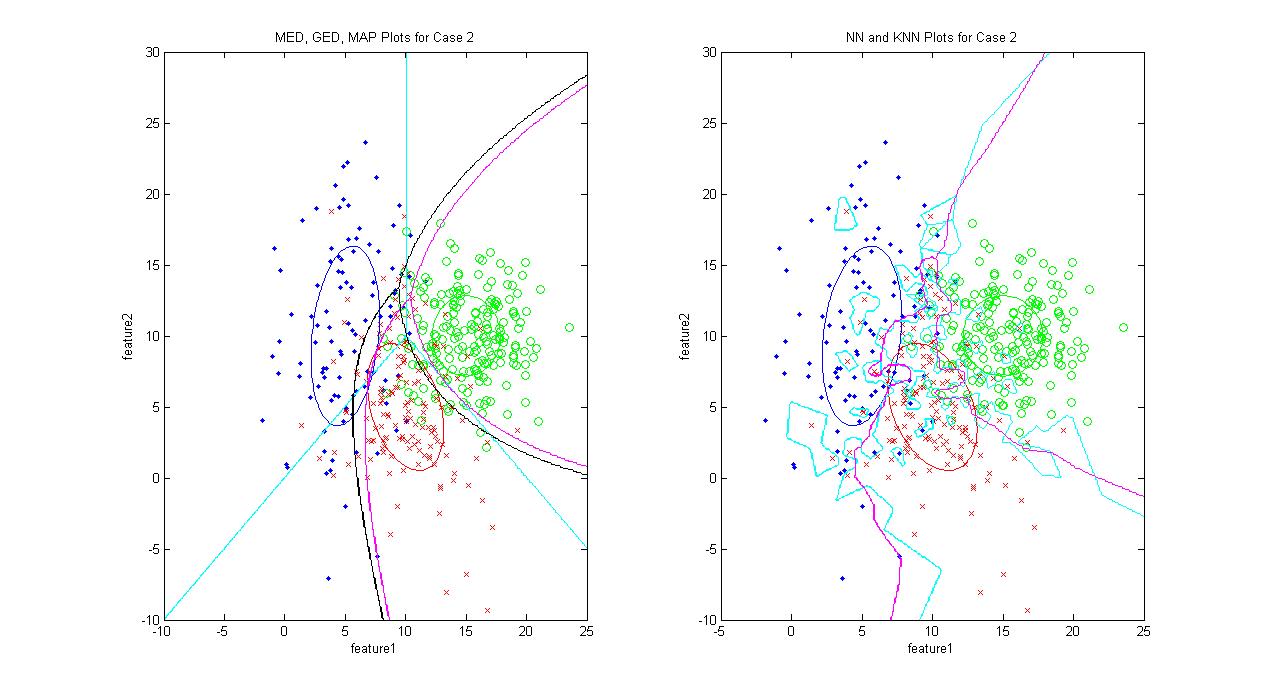


Figure - Decision Boundaries for Case 2

In the MED, GED, MAP plot in figure 4, the MED decision boundary is coloured Cyan, the GED plot is coloured in Magenta, and the MAP plot is coloured in Black. On the NN, KNN plots, the NN boundary is coloured in Cyan while the KNN plot is coloured in Magenta.

Similar to Case 1, these boundaries each display as expected. In the MED, GED, MAP plot, it can be observed that the MED curve is least able to correctly fit the data. This is consistent with theory, as it does not take the correlation matrix into account. Unlike in Case 1, the GED and MAP boundaries are not identical, although they do share a similar form. Since the a priori information is not equivalent, the MPA classifier favours classes which have previously appeared more frequently, thus skewing the decision boundary to the left .

Just like in Case 1, in the NN/KNN Boundary, we can observe that while both boundaries follow a similar pattern, the NN boundary (Cyan) weaves around every single point, while the KNN Boundary (magenta) has an averaging effect. This is expected, as the KNN boundary takes into account the average of the closest 5 points for each distribution, making it less susceptible to noise.

Error analysis for each of the boundaries will be evaluated in the following section.

# 3.0 Error Analysis:

Each algorithm was tested against the datasets provided in order to find the experimental error rate, as well as the confusion matrix. The results are summarized in table 1.

It should be noted that in the 3 distributions algorithms, the confusion matrix was reduced to the form:

This is done because of the way that the error algorithm was implemented. The error is found by passing all the points in a distribution to the algorithm. The points that are rejected are falsely rejected. Otherwise, they are correctly identified. This reduces has the drawback of not distinguishing between the two distributions that are incorrectly rejected.

Table : probability distributions for all algorithms

|  |  |  |
| --- | --- | --- |
| Algorithm |  | Confusion matrix |
| MED (2 distributions) | 0.05, 0.085 |  |
| MAP (2 distributions) | 0.0, 0.02 |  |
| GED (2 distributions) | 0.0, 0.02 |  |
| 1NN (2 distributions) | 0.06, 0.02 |  |
| 5NN (2 distributions) | 0.055, 0.07 |  |
| Algorithm |  | Confusion matrix |
| MED (3 distributions) | 0.19, 0.08, 0.29 |  |
| MAP (3 distributions) | 0.48, 0, 1 |  |
| GED (3 distributions) | 0.0, 0.03, 1.0 |  |
| 1NN (3 distributions) | 0.18, 0.1, 0.41 |  |
| 5NN (3 distributions) | 0, 0.005, 0.04 |  |

The smallest error of all the algorithms is MAP and GED. The confusion matrices in CASE 2 are either extremely inaccurate or spectacularly accurate. This might be due to an error in implementation.

# 4.0 Conclusions:

There is a fundamental design trade-off in speed between algorithms that require training (e.g. Nearest neighbour) and those that do not (e.g MED, GED, MAP). The choice in algorithm must be made in context of the problem provided and the form of the data given, which might include the distance between class means and size of each class variance.

# 5.0 Appendix – MATLAB Code

## 5.1 FeatureClass.m

classdef featureclass

properties

mu

sigma

prob

end

methods

function FC = featureclass(mu,sigma,prob)

FC.mu = mu;

FC.sigma = sigma;

FC.prob = prob;

end

end

end

## 5.2 Lab1.m

% SYDE Lab 0 - Matlab Introduction

% Feb 2nd 2013

clear all

close all

%n\_points = 200;

xDim1 = -5:1:20;

yDim1 = 20:-1:5;

xDim2 = -10:1:25;

yDim2 = 30:-1:-10;

%grid = Functions.ConstructGrid(xDim,yDim)

[X1, Y1] = meshgrid(xDim1,yDim1);

[X2, Y2] = meshgrid(xDim2,yDim2);

%Case 1 Training Data

NA = 200;

NB = 200;

class\_A = featureclass([5 10]',[8 0; 0 4],NA/(NA+NB));

class\_B = featureclass([10 15]',[8 0;0 4],NA/(NA+NB));

rA = Functions.GenerateDist(class\_A,NA);

rB = Functions.GenerateDist(class\_B,NB);

MED\_BoundaryCase1 = Functions.MEDBoundary2(X1,Y1,class\_A,class\_B);

GED\_BoundaryCase1 = Functions.GEDBoundary2(X1,Y1,class\_A,class\_B);

MAP\_BoundaryCase1 = Functions.MAPBoundary2(X1,Y1,class\_A,class\_B);

%K=1 therefore just NN

NN\_BoundaryCase1 = Functions.KNNBoundary2(X1,Y1,rA,rB,1);

k5NN\_BoundaryCase1 = Functions.KNNBoundary2(X1,Y1,rA,rB,5);

%Case 2 Training Data

NC = 100;

ND = 200;

NE = 150;

class\_C = featureclass([5 10]',[8 4;4 40],NC/(NC+ND+NE));

class\_D = featureclass([15 10]',[8 0;0 8],ND/(NC+ND+NE));

class\_E = featureclass([10 5]',[10 -5;-5 20],NE/(NC+ND+NE));

rC = Functions.GenerateDist(class\_C,NC);

rD = Functions.GenerateDist(class\_D,ND);

rE = Functions.GenerateDist(class\_E,NE);

MED\_BoundaryCase2 = Functions.MEDBoundary3(X2,Y2,class\_C,class\_D,class\_E);

GED\_BoundaryCase2 = Functions.GEDBoundary3(X2,Y2,class\_C,class\_D,class\_E);

MAP\_BoundaryCase2 = Functions.MAPBoundary3(X2,Y2,class\_C,class\_D,class\_E);

NN\_BoundaryCase2 = Functions.KNNBoundary3(X2,Y2,rC,rD,rE,1);

k5NN\_BoundaryCase2 = Functions.KNNBoundary3(X2,Y2,rC,rD,rE,5);

%Case 1 Testing Data

tA = Functions.GenerateDist(class\_A,NA);

tB = Functions.GenerateDist(class\_B,NB);

%Case 2 Testing Data

tC = Functions.GenerateDist(class\_C,NC);

tD = Functions.GenerateDist(class\_D,ND);

tE = Functions.GenerateDist(class\_E,NE);

figure

subplot(1,2,1)

plot(rA(:,1),rA(:,2),'b.');

hold on;

plot(rB(:,1),rB(:,2),'ro');

hold on;

plot\_ellipse(class\_A,'b');

hold on;

plot\_ellipse(class\_B,'g');

hold on;

xlabel('feature1');

ylabel('feature2');

title('MED,GED and MAP Plots for Case 1');

[C6,h6]=contour(xDim1,yDim1,MED\_BoundaryCase1,1);

set(h6,'EdgeColor','c');

[C7,h7]=contour(xDim1,yDim1,GED\_BoundaryCase1,1);

set(h7,'EdgeColor','m');

[C10,h10]=contour(xDim1,yDim1,MAP\_BoundaryCase1,1);

set(h10,'EdgeColor','c');

subplot(1,2,2)

plot(rA(:,1),rA(:,2),'b.');

hold on;

plot(rB(:,1),rB(:,2),'ro');

hold on;

plot\_ellipse(class\_A,'b');

hold on;

plot\_ellipse(class\_B,'g');

hold on;

xlabel('feature1');

ylabel('feature2');

title('NN and KNN Plots for Case 1');

[C8,h8]=contour(xDim1,yDim1,NN\_BoundaryCase1,1);

set(h8,'EdgeColor','c');

hold on;

[C9,h9]=contour(xDim1,yDim1,k5NN\_BoundaryCase1,10);

set(h9,'EdgeColor','m');

figure

subplot(1,2,1)

plot(rC(:,1),rC(:,2),'b.');

hold on;

plot(rD(:,1),rD(:,2),'go');

hold on;

plot(rE(:,1),rE(:,2),'rx');

hold on;

plot\_ellipse(class\_C,'b');

hold on;

plot\_ellipse(class\_D,'g');

hold on;

plot\_ellipse(class\_E,'r');

hold on;

xlabel('feature1');

ylabel('feature2');

title('MED, GED, MAP Plots for Case 2')

[C3,h3]=contour(xDim2,yDim2,MED\_BoundaryCase2,2);

set(h3,'EdgeColor','c');

[C4,h4]=contour(xDim2,yDim2,GED\_BoundaryCase2,2);

set(h4,'EdgeColor','m');

[C5,h5]=contour(xDim2,yDim2,MAP\_BoundaryCase2,2);

set(h5,'EdgeColor','k');

subplot(1,2,2)

plot(rC(:,1),rC(:,2),'b.');

hold on;

plot(rD(:,1),rD(:,2),'go');

hold on;

plot(rE(:,1),rE(:,2),'rx');

hold on;

plot\_ellipse(class\_C,'b');

hold on;

plot\_ellipse(class\_D,'g');

hold on;

plot\_ellipse(class\_E,'r');

hold on;

xlabel('feature1');

ylabel('feature2');

title('NN and KNN Plots for Case 2')

[C1,h1] = contour(xDim2,yDim2,NN\_BoundaryCase2,2);

set(h1,'EdgeColor','c');

hold on;

[C2,h2] = contour(xDim2,yDim2,k5NN\_BoundaryCase2,2);

set(h2,'EdgeColor','m');

med\_error = Functions.error('Functions.MEDBoundary2', rA, rB, class\_A, class\_B);

map\_error = Functions.error('Functions.MAPBoundary2', rA, rB, class\_A, class\_B);

ged\_error = Functions.error('Functions.GEDBoundary2', rA, rB, class\_A, class\_B);

nn\_error = Functions.error('Functions.KNNBoundary2', tA, tB, rA, rB);

knn\_error = Functions.error\_knn('Functions.KNNBoundary2', tA, tB, rA, rB, 5);

med\_error3 = Functions.error('Functions.MEDBoundary2', rC, rD, rE, class\_C, class\_D, class\_E);

map\_error3 = Functions.error('Functions.MAPBoundary2', rC, rD, rE, class\_C, class\_D, class\_E);

ged\_error3 = Functions.error('Functions.GEDBoundary2', rC, rD, rE, class\_C, class\_D, class\_E);

nn\_error3 = Functions.error('Functions.KNNBoundary2', tC, tD, rE, rC, rD, rE);

knn\_error3 = Functions.error\_knn('Functions.KNNBoundary2', tC, tD, rE, rC, rD, rE, 5);

## 5.3 Functions.m

classdef Functions

methods (Static = true)

function GD = GenerateDist(class, n\_points)

GD = mvnrnd(class.mu, class.sigma, n\_points);

end

function MED\_Distance = MEDDistance(data\_point,class)

MED\_Distance = sqrt((data\_point - class.mu)'\*(point - class.mu));

end

function MEDDist = MEDDist(X,Y,class)

MEDDist = sqrt((X - class.mu(1)).^2+(Y - class.mu(2)).^2);

end

function GEDDist = GEDDist(x,y,class)

temp = [x y]' - class.mu;

GEDDist = temp'\*inv(class.sigma)\*temp;

end

function MAPDist = MAPDist(x,y,class)

MAPDist = class.prob\*exp(-0.5\*Functions.GEDDist(x,y,class))/(det(class.sigma))^(0.5);

end

function MED\_Boundary = MEDBoundary2(X,Y,class\_1,class\_2)

MED\_Boundary = zeros(size(X));

MEDDist1 = Functions.MEDDist(X,Y,class\_1);

MEDDist2 = Functions.MEDDist(X,Y,class\_2);

[h,w] = size(MEDDist1);

for i = 1:w

for j = 1:h

MED\_Boundary(j,i) = Functions.GetSmallestValue2(MEDDist1(j,i),MEDDist2(j,i));

end

end

end

function MED\_Boundary = MEDBoundary3(X,Y,class\_1,class\_2,class\_3)

MED\_Boundary = zeros(size(X));

MEDDist1 = Functions.MEDDist(X,Y,class\_1);

MEDDist2 = Functions.MEDDist(X,Y,class\_2);

MEDDist3 = Functions.MEDDist(X,Y,class\_3);

[h,w] = size(MEDDist1);

for i = 1:w

for j = 1:h

MED\_Boundary(j,i) = Functions.GetSmallestValue3(MEDDist1(j,i),MEDDist2(j,i),MEDDist3(j,i));

end

end

end

function SV = GetSmallestValue3(x,y,z)

if x <= y

SV = 0;

else

SV = 1;

end

if z <= y && z <= x

SV = 2;

end

end

function SV = GetSmallestValue3MAP(x,y,z)

if x >= y

SV = 0;

else

SV = 1;

end

if z >= y && z >= x

SV = 2;

end

end

function SV = GetSmallestValue2(x,y)

if x <= y

SV = 0;

else

SV = 1;

end

end

function GED\_Boundary = GEDBoundary2(X,Y,class\_1,class\_2)

GED\_Boundary = zeros(size(X));

[h,w] = size(GED\_Boundary);

for i = 1:w

for j = 1:h

GEDDist1 = Functions.GEDDist(X(1,i),Y(j,1), class\_1);

GEDDist2 = Functions.GEDDist(X(1,i),Y(j,1), class\_2);

GED\_Boundary(j,i) = Functions.GetSmallestValue2(GEDDist1,GEDDist2);

end

end

end

function GED\_Boundary = GEDBoundary3(X,Y,class\_1,class\_2,class\_3)

GED\_Boundary = zeros(size(X));

[h,w] = size(GED\_Boundary);

for i = 1:w

for j = 1:h

GEDDist1 = Functions.GEDDist(X(1,i),Y(j,1), class\_1);

GEDDist2 = Functions.GEDDist(X(1,i),Y(j,1), class\_2);

GEDDist3 = Functions.GEDDist(X(1,i),Y(j,1), class\_3);

GED\_Boundary(j,i) = Functions.GetSmallestValue3(GEDDist1,GEDDist2,GEDDist3);

end

end

end

function MAP\_Boundary = MAPBoundary2(X,Y,class\_1,class\_2)

MAP\_Boundary = zeros(size(X));

[h,w] = size(MAP\_Boundary);

for i = 1:w

for j = 1:h

MAPDist1 = Functions.MAPDist(X(1,i),Y(j,1), class\_1);

MAPDist2 = Functions.MAPDist(X(1,i),Y(j,1), class\_2);

MAP\_Boundary(j,i) = Functions.GetSmallestValue2(MAPDist1,MAPDist2);

end

end

end

function MAP\_Boundary = MAPBoundary3(X,Y,class\_1,class\_2,class\_3)

MAP\_Boundary = zeros(size(X));

[h,w] = size(MAP\_Boundary);

for i = 1:w

for j = 1:h

MAPDist1 = Functions.MAPDist(X(1,i),Y(j,1), class\_1);

MAPDist2 = Functions.MAPDist(X(1,i),Y(j,1), class\_2);

MAPDist3 = Functions.MAPDist(X(1,i),Y(j,1), class\_3);

MAP\_Boundary(j,i) = Functions.GetSmallestValue3MAP(MAPDist1,MAPDist2,MAPDist3);

end

end

end

%sorting function from Mathworks

function [smallestNElements smallestNIdx] = getNElements(A, n)

[ASorted AIdx] = sort(A);

smallestNElements = ASorted(1:n);

smallestNIdx = AIdx(1:n);

end

function EucledianDistance = EucledeanDistance2(X,Y,r1)

EucledianDistance = sqrt((X - r1(:,1)).^2+(Y - r1(:,2)).^2);

end

function KNN\_Boundary = KNNBoundary2(X,Y,r1,r2,K)

if ~exist('K', 'var') || isempty(K)

K = 1;

end

KNN\_Boundary = zeros(size(X));

EucDist1 = 50;

EucDist2 = 50;

[h,w] = size(KNN\_Boundary);

for i = 1:w

for j = 1:h

temp1 = Functions.EucledeanDistance2(X(1,i),Y(j,1),r1);

EucDist1 = Functions.getNElements(temp1,K);

temp2 = Functions.EucledeanDistance2(X(1,i),Y(j,1),r2);

EucDist2 = Functions.getNElements(temp2,K);

%if K == 1

class1\_mean = mean(EucDist1);

class2\_mean = mean(EucDist2);

KNN\_Boundary(j,i) = Functions.GetSmallestValue2(class1\_mean,class2\_mean);

EucDist1 = 50;

EucDist2 = 50;

%end

end

end

end

function KNN\_Boundary = KNNBoundary3(X,Y,r1,r2,r3,K)

if ~exist('K', 'var') || isempty(K)

K = 1;

end

KNN\_Boundary = zeros(size(X));

EucDist1 = 50;

EucDist2 = 50;

EucDist3 = 50;

[h,w] = size(KNN\_Boundary);

for i = 1:w

for j = 1:h

temp1 = Functions.EucledeanDistance2(X(1,i),Y(j,1),r1);

EucDist1 = Functions.getNElements(temp1,K);

temp2 = Functions.EucledeanDistance2(X(1,i),Y(j,1),r2);

EucDist2 = Functions.getNElements(temp2,K);

temp3 = Functions.EucledeanDistance2(X(1,i),Y(j,1),r3);

EucDist3 = Functions.getNElements(temp3,K);

class1\_mean = mean(EucDist1);

class2\_mean = mean(EucDist2);

class3\_mean = mean(EucDist3);

KNN\_Boundary(j,i) = Functions.GetSmallestValue3(class1\_mean,class2\_mean,class3\_mean);

EucDist1 = 50;

EucDist2 = 50;

EucDist3 = 50;

end

end

end

function error = error(classifier, distribution\_1, distribution\_2, class\_A, class\_B)

[x\_1, y\_1] = size(distribution\_1);

[x\_2, y\_2] = size(distribution\_2);

fn = str2func(classifier);

error\_1 = fn(distribution\_1(:,1), distribution\_1(:,2), class\_A, class\_B);

error\_2 = fn(distribution\_2(:,1), distribution\_2(:,2), class\_A, class\_B);

true\_positives\_1 = sum(error\_1(:)==0);

false\_rejections\_1 = x\_1-true\_positives\_1;

true\_positives\_2 = sum(error\_2(:)==1);

false\_rejections\_2 = x\_2-true\_positives\_2;

confusion = [true\_positives\_1 false\_rejections\_2;

false\_rejections\_1 true\_positives\_2

];

%disp(false\_rejections\_1/x\_1);

%disp(false\_rejections\_2/x\_2);

error = confusion;

end

function error = error\_knn(classifier, distribution\_1, distribution\_2, class\_A, class\_B, k)

[x\_1, y\_1] = size(distribution\_1);

[x\_2, y\_2] = size(distribution\_2);

fn = str2func(classifier);

error\_1 = fn(distribution\_1(:,1), distribution\_1(:,2), class\_A, class\_B, k);

error\_2 = fn(distribution\_2(:,1), distribution\_2(:,2), class\_A, class\_B, k);

true\_positives\_1 = sum(error\_1(:)==0);

false\_rejections\_1 = x\_1-true\_positives\_1;

true\_positives\_2 = sum(error\_2(:)==1);

false\_rejections\_2 = x\_2-true\_positives\_2;

confusion = [true\_positives\_1 false\_rejections\_2;

false\_rejections\_1 true\_positives\_2

];

%disp(false\_rejections\_1/x\_1);

%disp(false\_rejections\_2/x\_2);

error = confusion;

end

function error = error3(classifier, distribution\_1, ...

distribution\_2, distribution\_3, class\_A, class\_B, class\_C)

[x\_1, y\_1] = size(distribution\_1);

[x\_2, y\_2] = size(distribution\_2);

[x\_3, y\_3] = size(distribution\_3);

fn = str2func(classifier);

error\_1 = fn(distribution\_1(:,1), distribution\_1(:,2), class\_A, class\_B, class\_C);

error\_2 = fn(distribution\_2(:,1), distribution\_2(:,2), class\_A, class\_B, class\_C);

error\_3 = fn(distribution\_3(:,1), distribution\_3(:,2), class\_A, class\_B, class\_C);

true\_positives\_1 = sum(error\_1(:)==0);

false\_rejections\_1 = x\_1-true\_positives\_1;

true\_positives\_2 = sum(error\_2(:)==1);

false\_rejections\_2 = x\_2-true\_positives\_2;

true\_positives\_3 = sum(error\_3(:)==2);

false\_rejections\_3 = x\_3-true\_positives\_3;

confusion = [true\_positives\_1 true\_positives\_2 true\_positives\_3;

false\_rejections\_1 false\_rejections\_2 false\_rejections\_3;

];

disp(false\_rejections\_1/x\_1);

disp(false\_rejections\_2/x\_2);

disp(false\_rejections\_3/x\_3);

error = confusion;

end

function error = error\_knn3(classifier, distribution\_1, ...

distribution\_2, distribution\_3, class\_A, class\_B, class\_C, k)

[x\_1, y\_1] = size(distribution\_1);

[x\_2, y\_2] = size(distribution\_2);

[x\_3, y\_3] = size(distribution\_3);

fn = str2func(classifier);

error\_1 = fn(distribution\_1(:,1), distribution\_1(:,2), class\_A, class\_B, class\_C, k);

error\_2 = fn(distribution\_2(:,1), distribution\_2(:,2), class\_A, class\_B, class\_C, k);

error\_3 = fn(distribution\_3(:,1), distribution\_3(:,2), class\_A, class\_B, class\_C, k);

true\_positives\_1 = sum(error\_1(:)==0);

false\_rejections\_1 = x\_1-true\_positives\_1;

true\_positives\_2 = sum(error\_2(:)==1);

false\_rejections\_2 = x\_2-true\_positives\_2;

true\_positives\_3 = sum(error\_3(:)==2);

false\_rejections\_3 = x\_3-true\_positives\_3;

confusion = [true\_positives\_1 true\_positives\_2 true\_positives\_3;

false\_rejections\_1 false\_rejections\_2 false\_rejections\_3;

];

disp(false\_rejections\_1/x\_1);

disp(false\_rejections\_2/x\_2);

disp(false\_rejections\_3/x\_3);

error = confusion;

end

end

end

## 5.4 plot\_ellipse.m

%

% Plot\_Ellipse(x,y,theta,a,b)

%

% This routine plots an ellipse with centre (x,y), axis lengths a,b

% with major axis at an angle of theta radians from the horizontal.

%

%

%

% Author: P. Fieguth

% Jan. 98

%Modified by Jason Dunham

%function plot\_ellipse(x,y,theta,a,b)

function plot\_ellipse(featureclass,colour)

%if nargin<5, error('Too few arguments to Plot\_Ellipse.'); end;

x = featureclass.mu(1);

y = featureclass.mu(2);

[ eigvect , diag ] = eig(featureclass.sigma);

theta = atan( eigvect(2,1) / eigvect(1,1) );

a = sqrt(diag(1,1));

b = sqrt(diag(2,2));

np = 100;

ang = [0:np]\*2\*pi/np;

pts = [x;y]\*ones(size(ang)) + [cos(theta) -sin(theta); sin(theta) cos(theta)]\*[cos(ang)\*a; sin(ang)\*b];

plot( pts(1,:), pts(2,:),colour );