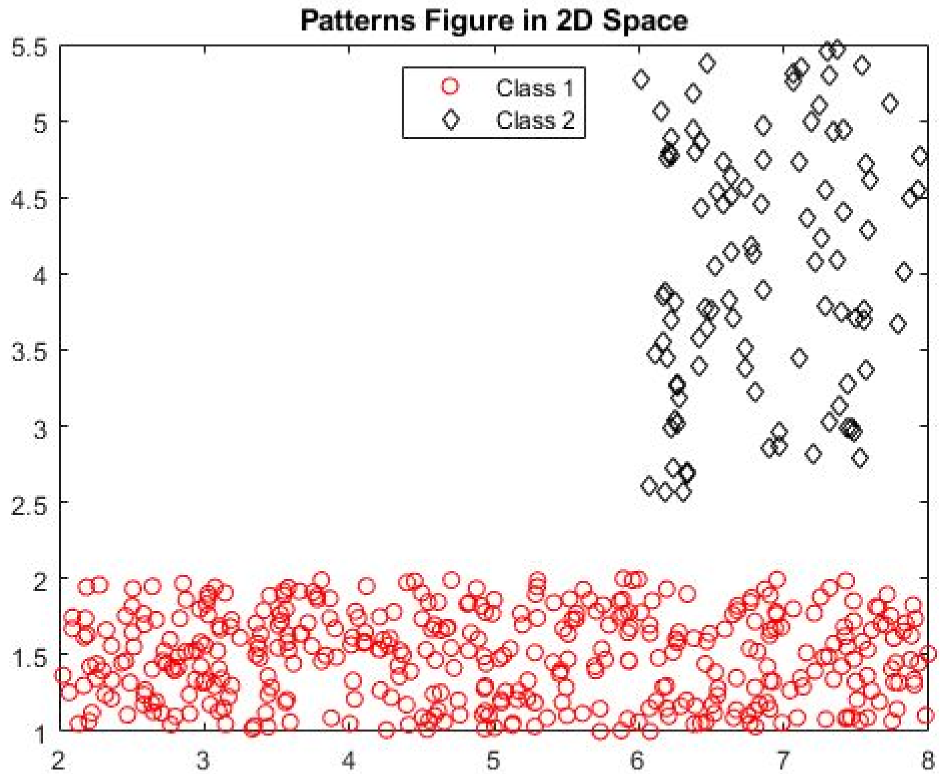
CE345 - Pattern Recognition Project

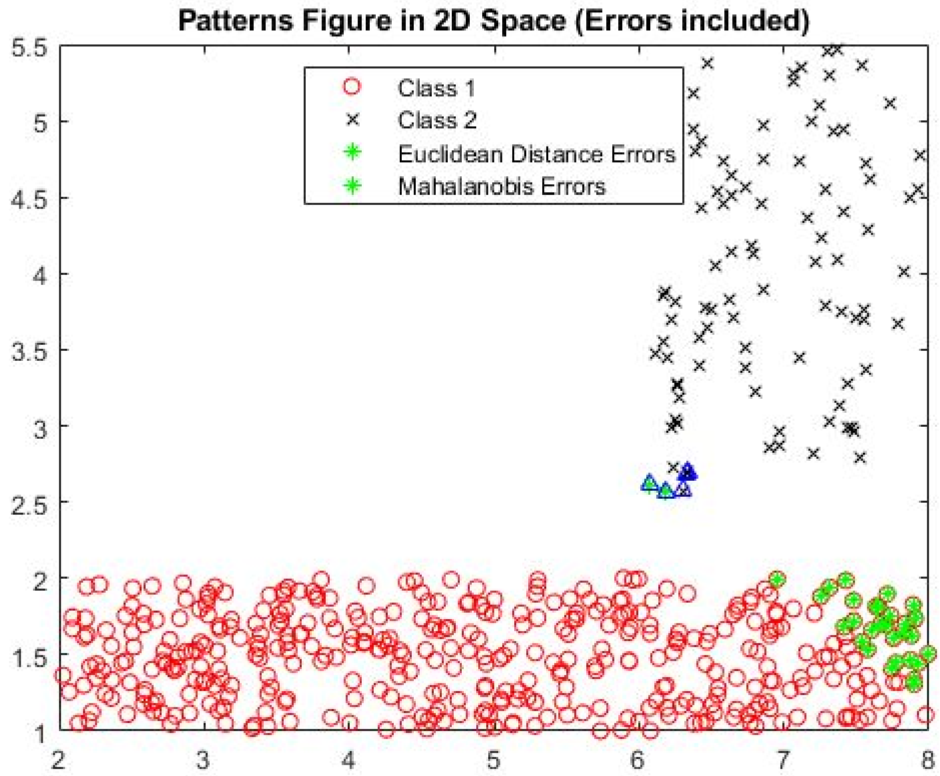
Eleftherios P. Loukas - AEM 2029

Results & Figures:

Part A: Dataset Generation



Part B: Bayesian Classification



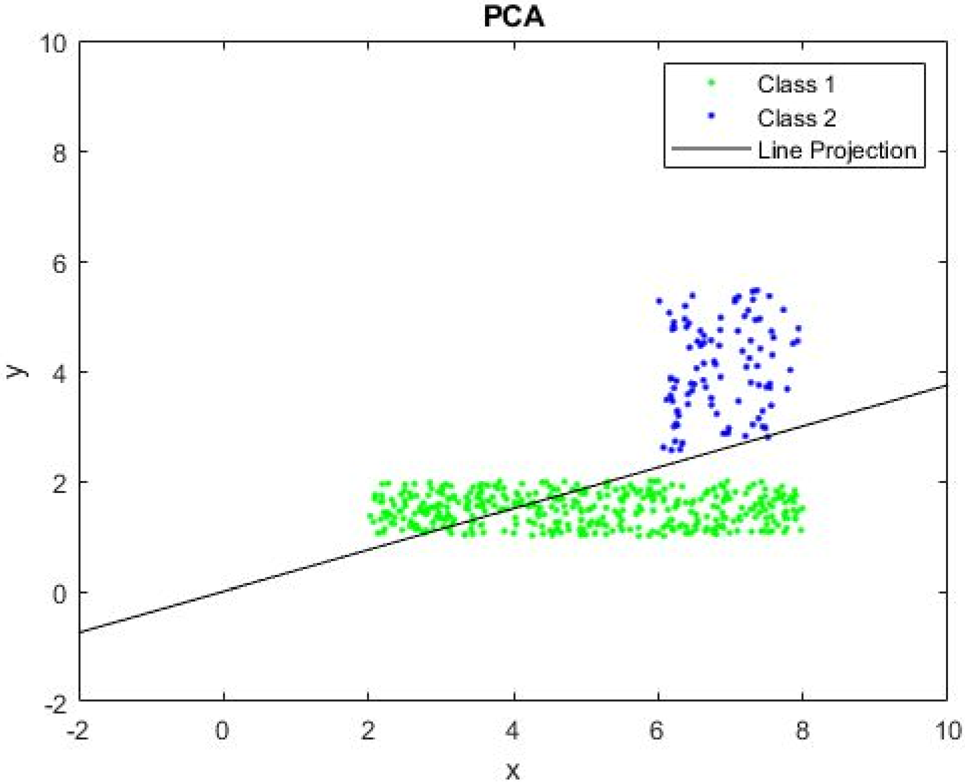
For some reason, the legend here won't show the correct signs. The errors based on the Euclidean Distance are the green ones, while the cyan signs denote the Mahalanobis errors.

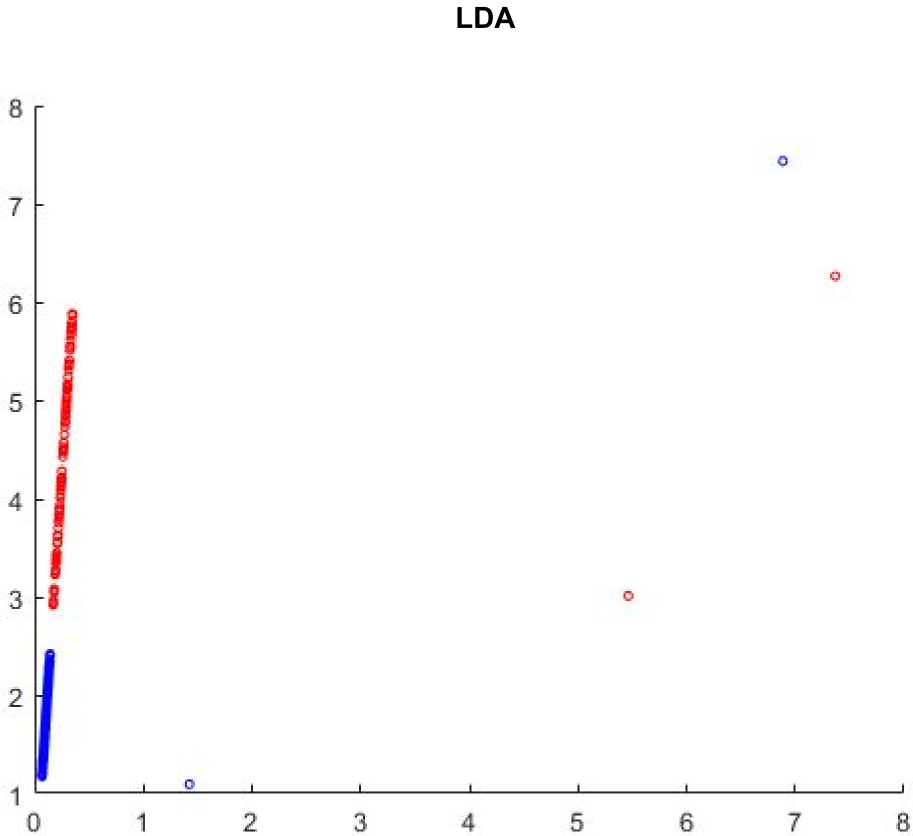
MLE:
*m1 = [**4.9672**99894532121, **1.4917**68678389496]*
*m2 = [**6.8619**04140477367, **4.0362**98245503686]*

*Σ1 = [**2.9883**54224903508,**-0.0134**16853169550;**-0.0134**16853169550,**0.0798**27083871080]*
*Σ2 = [**0.29140**6998056413,**0.0771**35776712221;**0.0771**35776712221, **0.6868**94833599279]*

Euclidean Distance error : 6.4%
Mahalanobis Distance Error : 1%
Bayesian Error : 0%

Part C: Feature Dimensionality Reduction

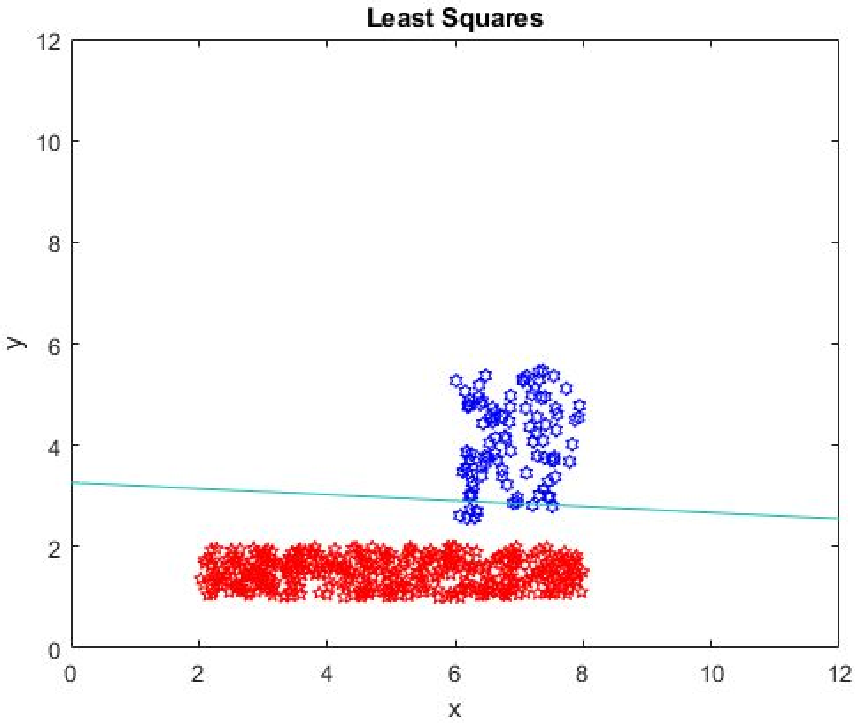


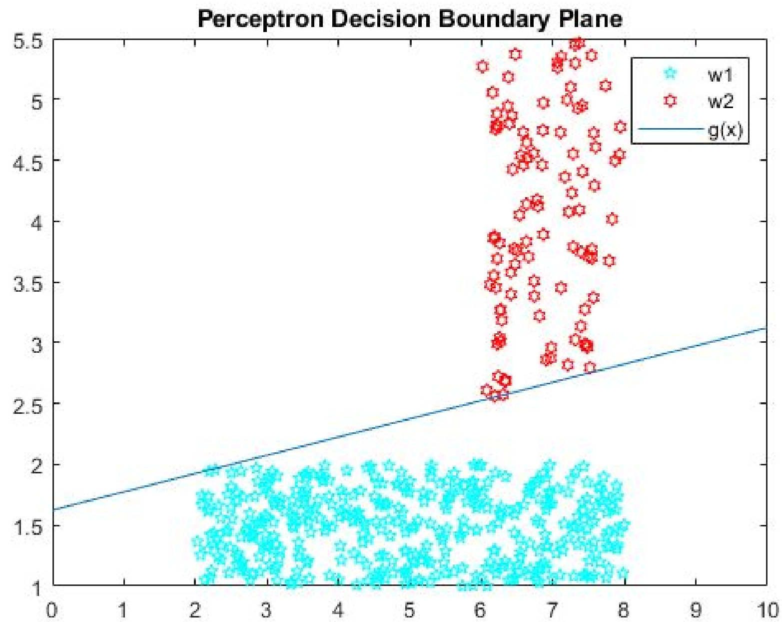


Classification Error based on Euclidean Distance After PCA : 0.4%
Classification Error based on Euclidean Distance After LDA: 0.2%

After reducing our dimensions with the PCA, we achieve a really good classification. This gets even better on LDA. We can see the “1-axis” line in the left of our diagram, while there are some ‘outliers’ on the top right, probably determining the 0.2% error that we get.

Part D: Linear Classification





The least squares methodology is performing well with an error of 349.09 while we can validate it visually.

The perceptron decision boundary looks even better on the classification problem. The parameters that were used for this experiment consist of three weights initialized to
w = [-11; -1; -11];
The learning rate is equal to 0.1;

The code for all the results is provided below, produced in the MATLAB environment. It also provides information on the numerical computation methods that were used.

The code will be hosted available online at github.com/eloukas/uth-pattern-recognition

```
close all; clc; clear; clf;

%For Octave
%pkg load statistics
%pkg load symbolic


% Section A
% Generate Dataset

% Create patterns for class 1 in a parallelogram way;
% Define variables for the dataset generation
a = 2; b = 8; c = 1; d = 2;
N1 = 400;

% Generate randomly the dataset
x1 = a + (b-a).*rand(N1, 1);
x2 = c + (d-c).*rand(N1, 1);

% Assign patterns to class 1
w1 = [x1 x2];

% Create patterns for class 2 in a parallelogram way;
% Define variables for the dataset generation
a = 6; b = 8; c = 2.5; d = 5.5;
N2 = 100;

% Generate randomly the datasets
x1 = a + (b-a).*rand(N2, 1);
x2 = c + (d-c).*rand(N2, 1);

% Assign patterns to class 2
w2 = [x1 x2];

% Plot all patterns of class 1 & 2
```

```matlab
plot(w1(:,1),w1(:,2),'ro',w2(:,1),w2(:,2),'kd');
title('Patterns Figure in 2D Space');
legend('Class 1','Class 2','Location','north');

%-----------------------------------------------------------------------

% Section B
% Bayesian Classification in 2D Space


% Section B.1
% Find the mean value and the covariances of the 2 PDFs using MLE

% Find the mean values
m1 = mean(w1);
m2 = mean(w2);
disp(['Class 1 mean is [',num2str(m1),']']);
disp(['Class 2 mean is [',num2str(m2),']',]);
% Find the covariance values
s1 = cov(w1);
s2 = cov(w2);

% Perform MLE for Class 1
p11 = [];
p12 = [];
for i = 1:N1
    % According to 2.27 equation from Theodoridis & Koutroumbas' book
    p11 = [p11,(1/(2*pi*sqrt(abs(det(s1))))) * exp(-(1/2)*(w1(i,:) -
m1)*inv(s1)*(w1(i,:) - m1)')];
    p12 = [p12,(1/(2*pi*sqrt(abs(det(s2))))) * exp(-(1/2)*(w1(i,:) -
m2)*inv(s2)*(w1(i,:) - m2)')];

end

% Perform MLE for Class 2
p21 = [];
p22 = [];
for i = 1:N2
  % According to 2.27 equation from Theodoridis & Koutroumbas' book
    p22 = [p22,(1/(2*pi*sqrt(abs(det(s2))))) * exp(-(1/2)*(w2(i,:) -
m2)*inv(s2)*(w2(i,:) - m2)')];
    p21 = [p21,(1/(2*pi*sqrt(abs(det(s1))))) * exp(-(1/2)*(w2(i,:) -
m1)*inv(s1)*(w2(i,:) - m1)')];
end
```

```matlab
% Section B.2
% Initialize vectors for counting the error statistics
class1_correct = []; class2_right = []; class1_false = []; class2_false
= [];

% For the patterns of class 1
for i = 1:N1

    d1 = norm(w1(i,:) - m1); %  Find Distance from class 1 mean
    d2 = norm(w1(i,:) - m2); % Find Distance from class 2 mean

    if d1<=d2  % If distance1 < distance2
        class1_correct = [class1_correct; w1(i,:)]; % It's correctly
assigned to class 1
    else
        class1_false = [class1_false; w1(i,:)]; % It's falsely assigned
to class 1
    end

end

% Apply the same for the patterns of class2
for i = 1:N2

    % Find the distances
    d1 = norm(w2(i,:) - m1);
    d2 = norm(w2(i,:) - m2);

    % Assign to vectors
    if d1<d2
        class2_false = [class2_false; w2(i,:)];
    else
        class2_right = [class2_right; w2(i,:)];
    end

end

error = (size(class1_false, 1) + size(class2_false, 1)) / (N1+N2);
disp(['Euclidean Distance error : ', num2str(error*100),'%']);

% Plot the Euclidean Distance classifier
figure
plot(w1(:,1),w1(:,2),'ro',w2(:,1),w2(:,2),'kx');
```

```
if size(class1_false,1) ~= 0
    hold on;
    plot(class1_false(:,1),class1_false(:,2),'g*');
end

if size(class2_false,1) ~= 0
    hold on;
    plot(class2_false(:,1),class2_false(:,2),'g*');
end

% Red = Euclidean Distance Errors

% Section B.3
% Mahalanobis Distance Classifier

% Find the shared covariance matrix for the 2 classes
s = (s1+s2)/2;

% Initialize the vectors again
class1_correct = []; class2_right = []; class1_false = []; class2_false
= [];

% For class 1
for i = 1:N1

    % Calculate distances based on Mahalanobis
    d1 = sqrt((w1(i,:) - m1)*inv(s)*(w1(i,:) - m1)');
    d2 = sqrt((w1(i,:) - m2)*inv(s)*(w1(i,:) - m2)');

    % Assign patterns to appropriate vectors
    if d1<=d2
        class1_correct = [class1_correct;w1(i,:)];
    else
        class1_false = [class1_false;w1(i,:)];
    end
end

% For class 2
for i = 1:N2

    % Calculate distances based on Mahalanobis
    d1 = sqrt((w2(i,:) - m1)*inv(s)*(w2(i,:) - m1)');
    d2 = sqrt((w2(i,:) - m2)*inv(s)*(w2(i,:) - m2)');

    % Assign patterns to appropriate vectors
```

```
    if d1<d2
        class2_false = [class2_false;w2(i,:)];
    else
        class2_right = [class2_right;w2(i,:)];
    end
end

% Calculate statistics
error = (size(class1_false, 1) + size(class2_false, 1)) / (N1+N2);
disp(['Mahalanobis Distance Error : ',num2str(error*100),'%']);

if size(class1_false, 1) ~= 0
    hold on;
    plot(class1_false(:, 1),class1_false(:, 2),'b^');
end
if size(class2_false, 1) ~= 0
    hold on;
    plot(class2_false(:, 1),class2_false(:, 2),'b^');
end

title('Patterns Figure in 2D Space (Errors included)');
legend('Class 1', 'Class 2', 'Euclidean Distance Errors', 'Mahalanobis
Errors', 'Location', 'north');

% Section B.4

% Find the shared covariance matrix for the 2 classes
s = (s1+s2)/2;
% Initialize vectors for statistics
class1_correct = []; class2_right = []; class1_false = []; class2_false
= [];

% For class 1
for i = 1:N1

    % Apply Bayesian Classifier & Assign to Correct or False Class
    if p11>=p12
        class1_correct = [class1_correct;w1(i,:)];
    else
        class1_false = [class1_false;w1(i,:)];
    end

end

% For class 2
```

```matlab
for i = 1:N2

    % Apply Bayesian Classifier & Assign to Correct or False Class
    if p21>p22
        class2_false = [class2_false;w2(i,:)];
    else
        class2_right = [class2_right;w2(i,:)];
    end

end

% Print Error
error = (size(class1_false, 2) + size(class2_false,2)) / (N1+N2);
disp(['Bayesian Error : ', num2str(error*100),'%']);

% Section C
% Feature Dimensionality Reduction

% Section C.1
% Principal Compomnent Analysis

figure;
x = [w1;w2];

% Apply PCA
[coeff,score] = pca(x);
G = [w1;w2] * coeff;
plot(w1(:,1),w1(:,2),'g.');
hold on;

plot(w2(:,1),w2(:,2),'b.');
hold on;
syms x y;
hold on;

% Show data in the reduced space
f2(x,y) = coeff(1,2)*x + coeff(2,2)*y;
h = ezplot(f2,[-2, 10]); hold on;
set(h, 'Color', 'k');
title('PCA');
legend('Class 1', 'Class 2', 'Line Projection');
pause

% Section C.2
% Euclidean Distance Classifier after PCA
```

```matlab
% Initialize vectors for statistics
class1_false = []; class1_correct = []; class2_false = []; class2_right
= [];

% For transformed Class 1 features
for i = 1:N1

   % Find distance to mean of class 1 and 2
   d1 = norm(G(i,:) - m1);
   d2 = norm(G(i,:) - m2);

   % Assign correctly
   if d1<=d2
        class1_correct = [class1_correct;G(i,:)];
   else
        class1_false = [class1_false;G(i,:)];
   end
end

% For transformed Class 2 features
for i = (N2+1):(N1+N2)

    % Find distance to mean of class 1 and 2
   d1 = norm(G(i,:) - m1);
   d2 = norm(G(i,:) - m2);

   % Assign correctly
   if d1<d2
       class2_false = [class2_false;G(i,:)];
   else
       class2_right = [class2_right;G(i,:)];
   end
end

figure(3)
title('PCA Errors');
if size(class1_false,1) ~= 0
    hold on;
    plot(class1_false(:,1),class1_false(:,2),'k^');
end
if size(class2_false,1) ~= 0
    hold on;
    plot(class2_false(:,1),class2_false(:,2),'r^');
end
```

```
error = (size(class1_false, 2) + size(class2_false,2)) / (N1+N2);
disp(['Euclidean Distance on PCA Error : ', num2str(error*100),'%']);

% Section G.3
% Linear Discriminant Analysis
figure(4)
title('LDA')

% Calculate the possibilities (400/500, 100/500)
p1 = 4/5; p2 = 1/5;
% Compute Sw and w
Sw = p1*s1 + p2*s2;
w = inv(Sw) *(m1-m2)';

% Find the projections
w_p = w/norm(w, 2);
projections_w1= w_p*w_p'*w1';
projections_w2= w_p*w_p'*w2';

% Plot projections
scatter(w1(1, :), w1(2, :), 10, 'b');
hold on;
scatter(w2(1, :), w2(2, :), 10, 'r');
hold on;

scatter(projections_w1(1, :), projections_w1(2, :), 10, 'b');
hold on;
scatter(projections_w2(1, :), projections_w2(2, :), 10, 'r');

% % Section G.4
% Euclidean Distance Classifier after LDA

% Initialize vectors for statistics
class1_false = []; class1_correct = []; class2_false = []; class2_right
= [];

m1 = mean(projections_w1); m2 = mean(projections_w2);

% For transformed Class 1 features
for i = 1:N1

   % Find distance to mean of class 1 and 2
   d1 = norm(projections_w1(:,i) - m1);
   d2 = norm(projections_w1(:, i) - m2);
```

```
   % Assign correctly
   if d1<=d2
        class1_correct = [class1_correct;projections_w1(:,i)];
   else
        class1_false = [class1_false;projections_w1(:,i)];
   end
end

% For transformed Class 2 features
for i = 1:100

    % Find distance to mean of class 1 and 2
    d1 = norm(projections_w2(:,i) - m1);
    d2 = norm(projections_w2(:, i) - m2);

   % Assign correctly
   if d1<d2
       class2_false = [class2_false;projections_w1(:,i)];
   else
       class2_right = [class2_right;projections_w1(:,i)];
   end
end

error = (size(class1_false, 2) + size(class2_false,2)) / (N1+N2);
disp(['Euclidean Distance on LDA Error : ', num2str(error*100),'%']);

% Section D.1 - Classification
% Least Squares - Linear Classification

% Prepare y matrix for LS with labels 1 and -1
y = ones(500, 1); y(401: 500) = -y(401: 500);

% Prepare X matrix
X = ones(500, 3);
X(:,1:2) = [w1;w2];

% Calculate w
w = inv(X'*X)*X'*y; % Based on equation 3.45 from the book
disp(w);


% Plot W
figure;
plot(w1(:,1),w1(:,2),'rp',w2(:,1),w2(:,2),'bh');
```

```
hold on;

% Create a symbolic function based on weights the three weights
syms x y;
f(x,y) = w(1)*x + w(2)*y + w(3);
ezplot(f,[0,12]);
title('Least Squares');

% Calculate Least Square Error
J = [];
y = ones(500,1);
for i = 1:500
    J = [J, (y(i) - X(i,:)*w)^2];
end

lse = sum(J);

disp(['Minimum Least Square Error is: ', num2str(lse)]);

% Init errors
errors = 0;

% Compute Errors for class 1 and class 2
for i = 1:N1
    if w'*[w1(i,:),1]'<=0
        errors = errors +1;
    end
end

for  i = 1:N2
    if w'*[w2(i,:),1]' >= 0
        errors = errors +1;
    end
end


% Section D.2
% Perceptron

% Initialize weights and learning rate
w = [-11; -1; -11]; lr = 1/10;

% Prepare X, y just like previously
y = ones(500,1);
y(401:500) = -y(401:500);
```

```
X = [w1;w2];
X = [X, ones(500,1)];

Y = []; error = 1; % Used to bypass the first check in the while loop.
syms x y;

while length(Y)|| error ~= 0
    a = 0;
    Y = [];

    % For all patterns
    for i = 1:(N1+N2)
        if i<=400
            if X(i,:)*w<=0 % The first 400 patterns (class 1) should
have neuron output <=0
                Y = [Y;1*X(i,:)];
            end
        else
            if X(i,:)*w>=0  % The remaining 100 patterns (class 2)
should have neuron output >0
                Y = [Y;(-1)*X(i,:)];
            end
        end
    end

    % Calculate losses according to the cost function equation in 3.9
    deltas = [];
    for i = 1:size(Y,1)
      deltas = [deltas ;Y(i,:)];
    end
    error = sum(deltas);

    % Update weight accordingly to error and the learning error
    w = w +lr*error';

end

% Decision plane equation
f(x,y) = w(1)*x + w(2)*y+w(3);
figure;
plot(w1(:,1),w1(:,2),'cp',w2(:,1),w2(:,2),'rh'); hold on;

% Plot the decision boundary plot along with the data
N3 = 10; N4 = 1000; xx = linspace(0,N3, N4);
plot(xx,(xx*w(1)+w(3))/(-w(2)))
```

```
legend('w1','w2','g(x)'); title('Perceptron Decision Boundary Plane');
```