4.

For question **(b)** in all parts is implemented in `MAP` and `classify` functions in the Matlab code.

`MAP` function takes the mean vector matrix U and covariance matrix Σ then compute the values of w and x_0 of the discriminant function:

$$g(\underline{x}) = w^T \cdot (\underline{x} - x_0) \tag{1}$$

`classify` function takes the w, x_0 and *dataset* then classify the data, calculate $g(\underline{X}_n)$, compute and auto-tune the decision boundary.

For question **(c)**, the classification result is printed to the console window when executing the program (`assignment_2.m`).

Part I:

(d). Plot the data samples in **2DGaussianDataset1.txt** and decision boundary:

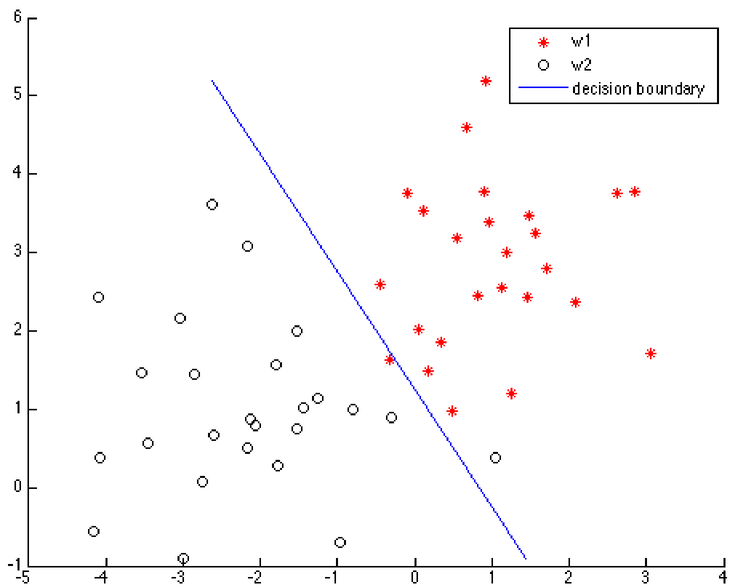

Figure 1: **2DGaussianDataset1.txt** and decision boundary (Classifier 1)

(e). Plot $g(\underline{X}_n)$ values:

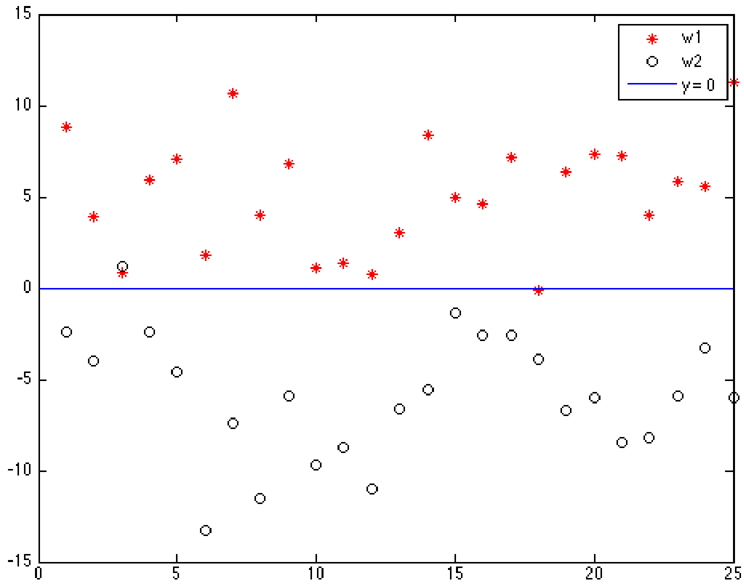

Figure 2: **2DGaussianDataset1.txt** and $g(\underline{X}_n)$ values (Classifier 1)

(f). The classification error rate for classifier 1 with **2DGaussianDataset1.txt** $= 4\%$.

Part II:

(d). Plot the data samples in **2DGaussianDataset2.txt** and decision boundary:

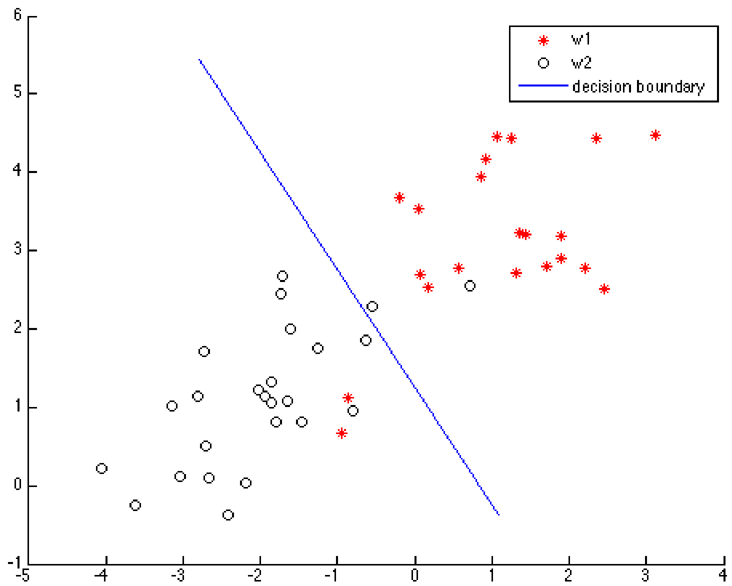

Figure 3: **2DGaussianDataset2.txt** and decision boundary (Classifier 1)

(e). Plot $g(\underline{X}_n)$ values:

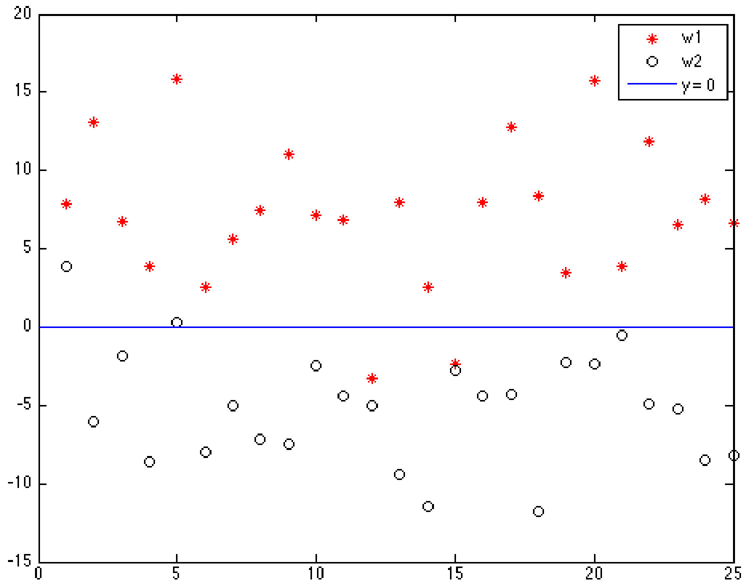

Figure 4: **2DGaussianDataset2.txt** and $g(\underline{X}_n)$ values (Classifier 1)

(f). The classification error rate for classifier 1 with **2DGaussianDataset2.txt** $= 8\%$.

Part III:

(d). Plot the data samples in **2DGaussianDataset2.txt** and decision boundary:

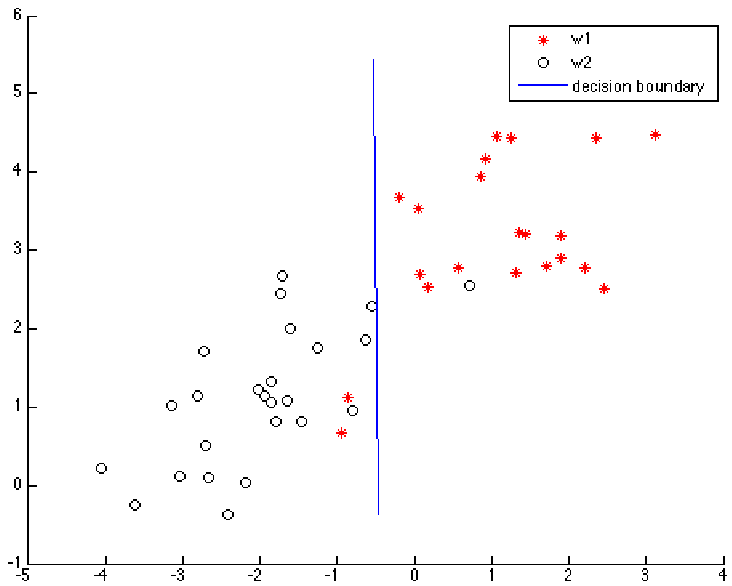

Figure 5: **2DGaussianDataset2.txt** and decision boundary (Classifier 2)

(e). Plot $g(\underline{X}_n)$ values:

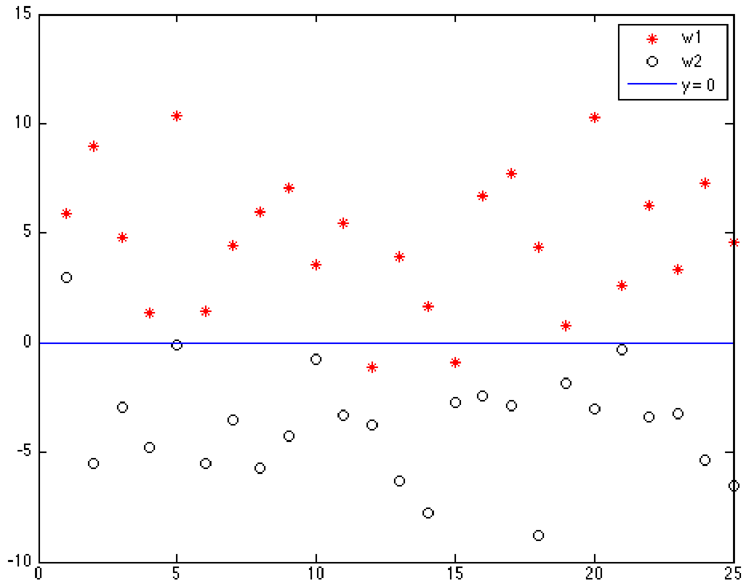

Figure 6: **2DGaussianDataset2.txt** and $g(\underline{X}_n)$ values (Classifier 2)

(f). The classification error rate for classifier 2 with **2DGaussianDataset2.txt** = 6%.

The difference between the error rates in **Part II** and **Part III** on **2DGaussianDataset2.txt** is due to the classifiers being used to perform the classification task. Classifier 2 is more accurate than classifier 1 since there is additional information available in term of correlation in the given covariance matrix in classifier 2.

$$\Sigma_{classifier\_1} = \begin{bmatrix} 1.21 & 0 \\ 0 & 1.21 \end{bmatrix}$$

$$\Sigma_{classifier\_2} = \begin{bmatrix} 1.21 & 0.8 \\ 0.8 & 1.21 \end{bmatrix}$$

Appendix:

assignment_2.m

```
%
% CS7720 Spring 2015
% Introduction to Machine Learning and Pattern Recognition
% University of Missouri-Columbia
%
% Author: Chanmann Lim
% email: cl9p8@mail.missouri.edu
%
% Homework Assignment 2
% Problem 4
%
clc; clear; close all;
dataset1 = load('2DGaussianDataset1.txt');
dataset2 = load('2DGaussianDataset2.txt');

%
% Part I
%
u1 = [1; 3]; u2 = [-2; 1];
C = [1.21 0; 0 1.21];

%
% report c, d, e, f
%
disp('-------------------------- Part I --------------------------');
report([u1 u2], C, dataset1);

%
% Part II
%   report c, d, e, f
%
disp(' '); disp('-------------------------- Part II --------------------------');
report([u1 u2], C, dataset2);

%
% Part III
%
disp(' '); disp('-------------------------- Part II --------------------------');
D = [1.21 .8; .8 1.21];
report([u1 u2], D, dataset2);
```

report.m

```
function report(U, C, dataset)
% REPORT - produce reports
% Input:
%   U - mean vector matrix
%   C - Covariance matrix
%   dataset - data sample
%
    [w, x0] = MAP(U, C);
    [classification, g_Xn, g_0] = classify(w, x0, dataset);

    %
    % c - classify dataset with minimum error classifier (MAP)
    %
    disp('Classification = '); disp([dataset classification']);

    %
    % d - plot data sample and decision boundary
    %
    figure;
    w1 = dataset(1:25, :);
    w2 = dataset(26:end, :);
    scatter(w1(:,1), w1(:,2), '*r'); hold on;
    scatter(w2(:,1), w2(:,2), 'ok'); hold on;
    plot(g_0(:,1), g_0(:,2)); hold off;
    legend('w1', 'w2', 'decision boundary');
```

```
    %
    % e - plot g(x_n)
    %
    figure;
    plot(g_Xn(1:25), '*r'); hold on;
    plot(g_Xn(26:end), 'ok'); hold on;
    x = 0:25; y = 0*x;
    plot(x, y); hold off;
    legend('w1', 'w2', 'y = 0');

    %
    % f - compute classification error rate
    %
    w1_misclassified = 25 - sum(classification(1:25));
    w2_misclassified = sum(classification(26:end));
    error_rate = (w1_misclassified + w2_misclassified) / size(dataset, 1) * 100;
    disp(['The classification error rate = ', num2str(error_rate), '%.']);
end
```

MAP.m

```
function [ w, x0 ] = MAP( U, C )
% MAP - Compute w and x0 values of g(x) minimum error classification
%   g(x) = w' * (x - x0)
%
% Input:
%   U - 2 mean vectors of the 2-D gaussian
%   C - Covariance of the 2-D gaussian (assume C1 = C2 = C)
%
% Output:
%   w - w vector
%   x0 - x0 vector

    u1 = U(:, 1); u2 = U(:, 2);
    w = C \ (u1 - u2); % = inv(C) * (u1 - u2)
    x0 = 1/2 * (u1 + u2);
end
```

classify.m

```
function [ classification, g_Xn, g_0 ] = classify( w, x0, dataset )
% CLASSIFY - Classify the dataset
%   Perform classification g(x) = w' * (x - x0)
%
% Input:
%   w - w vector in g(x) function
%   x0 - x0
%
% Output:
%   classification - 1 for class 1 and 0 for class 2.
%   g_Xn - value g(x) for all data samples.
%   g_0 - decision boundary
%
    % Auto-tune decision boundary to x2 bound
    [~, lower] = min(dataset(:, 2));
    [~, upper] = max(dataset(:, 2));
    g_0 = decision_boundary(w, x0, dataset([lower upper], 2));

    % NOTE: linear algebra can replace "for loop" here
    %       see ENHANCED section below
    g_Xn = [];
    for i=1:size(dataset, 1)
        x = dataset(i, :)';
        g = w' * (x - x0);
        g_Xn(i) = g;
    end

    classification = g_Xn > 0;
end

% ENHANCED:
%
%     scaled_x0 = x0 * ones(1, size(dataset, 1));
%     g_xn = w' * (dataset' - scaled_x0);
```

decision_boundary.m

```
function [ g_0 ] = decision_boundary( w, x0, y )
% DECISION_BOUNDARY - Derive decision boundary of 2-D Gaussian
%   Use discriminant function g(x) variables w and x0 to determine the
%   decision boundary. g(x) = 0.
%
% Input:
%   w - w of g(x)
%   x0 - x0 of g(x)
%   y - y data
%
    % ratio of w
    r = w(2)/w(1);
    % g_0 = [x1, computed_x2]
    %   the computed_x2 is derived from g(x) = 0
    g_0 = [ x0(1) - r*(y - x0(2)), y ];
end
```