EE 516
Pattern Recognition
Prof. Jane Zhang

Project 4

May 10, 2023

Nicholas Brunet
Nathan Jaggers
Jordan Perlas

Introduction

For this lab, we were given data from the textbook to design dichotomizers and test their accuracy when the number of dimensions change. We also were tasked with taking two distributions and creating a random distribution. With the two distributions, we used two methods to get different classifiers and tested their accuracy.

Data

Problem 1

Listing 1 is the dataset for Problem 1 in full, as represented in MATLAB. This dataset include 10 samples for three classes, ω_1, ω_2, and ω_3, with three variables for each sample, x_1, x_2, x_3.

Listing 1: Dataset

```
dataset = [
%   --------w1--------   --------w2--------   --------w3--------
%    x1      x2     x3     x1      x2     x3     x1      x2     x3
   -5.01  -8.12  -3.68  -0.91  -0.18  -0.05   5.35   2.26   8.13;
   -5.43  -3.48  -3.54   1.30   2.06  -3.53   5.12   3.22  -2.66;
    1.08  -5.52   1.66  -7.75  -4.54  -0.95   1.34  -5.31  -9.87;
    0.86  -3.78  -4.11  -5.47   0.50   3.92   4.48   3.42   5.19;
   -2.67   0.63   7.39   6.14   5.72  -4.85   7.11   2.39   9.21;
    4.94   3.29   2.08   3.60   1.26   4.36   7.17   4.33  -0.98;
   -2.51   2.09  -2.59   5.37  -4.63  -3.65   5.75   3.97   6.65;
   -2.25  -2.13  -6.94   7.18   1.46  -6.66   0.77   0.27   2.41;
    5.56   2.86  -2.26  -7.39   1.17   6.30   0.90  -0.43  -8.71;
    1.03  -3.33   4.33  -7.50  -6.32  -0.31   3.52  -0.36   6.43;
];
```

Problem 2

The data given in Problem 2 was not defined explicitly. Instead, we were given two random classes modeled by Gaussian distributions.

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mu_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}$$

Procedure

Problem 1

The following assumptions were made:

- The variables in the dataset followed a Gaussian distribution.
- The prior probabilities were $P(\omega_1) = P(\omega_2) = 0.5$ and $P(\omega_3) = 0$.
- A zero-one loss function was appropriate.

The task was to calculate the empirical training error of Bayesian classifiers for ω_1 and ω_2. The classifiers would use one (x_1), two (x_1, x_2), and three (x_1, x_2, x_3) variables or features. To create a classifier, we needed to calculate the mean (`*_mean`) and covariance (`*_cov`) matrices for each class, as shown in Listing 2.

Listing 2: Dataset

```
w1_mean = mean(w1_x)';
w2_mean = mean(w2_x)';
w1_cov = cov(w1_x);
w2_cov = cov(w2_x);
```

In MATLAB we defined a function `g(x, mean, cov, P)` (Listing 3), which would be equivalent to $g_i(\mathbf{x})$ if the `mean`, `cov`, and `P` values were for ω_i.

Listing 3: Discriminant

```
function result = g(x, mean, cov, P)
    cov_i = inv(cov);
    W = -0.5 * cov_i;
    w = cov_i * mean;
    w0 = -0.5 * (mean' * cov_i * mean) - 0.5 * log(det(cov)) + log(P);
    result = x'*W*x + w'*x + w0;
end
```

To calculate the empirical error, we looped through the samples and counted the number of misclassified samples (Listing 4).

At this point, we also calculated the Bhattacharyya bound (Listing 5). The Bhattacharyya is an approximation of the Chernoff error bound where the β term is assumed to be 0.5. Here we give up some accuracy for simpler computation. The Bhattacharyya error bound is shown below:

$$P(error) \leq \sqrt{P(\omega_1)P(\omega_2)}e^{-k(\frac{1}{2})}$$

$$k\left(\frac{1}{2}\right) = \frac{1}{8}(\mu_2 - \mu_1)^T \left[\frac{\Sigma_1 + \Sigma_2}{2}\right]^{-1} (\mu_2 - \mu 1) + \frac{1}{2} ln \frac{|\frac{\Sigma_1+\Sigma_2}{2}|}{\sqrt{|\Sigma_1||\Sigma_2|}}$$

Listing 4: Discriminant

```
correct = 0;
Y = [ones(size(w1_x, 1), 1); zeros(size(w2_x, 1), 1)];
X = [w1_x; w2_x];
for i = 1:size(X, 1)
    x = X(i,:)';
    y = Y(i);
    g1_result = g(x, w1_mean, w1_cov, w1_P);
    g2_result = g(x, w2_mean, w2_cov, w2_P);
    correct = correct + ((g1_result - g2_result > 0) == y);
end
acc = correct / size(X, 1);
fprintf("Accuracy: %.2f\n", acc);
```

Listing 5: Bhattacharyya

```
mu2_mu1 = w2_mean - w1_mean;
cov2_cov1 = w1_cov+w2_cov;
ln_num = det(cov2_cov1/2);
ln_den = sqrt(det(w1_cov)*det(w2_cov));

k_bha = (1/8)*mu2_mu1'*inv(cov2_cov1/2)*mu2_mu1+(1/2)*log(ln_num/ln_den);

P_error_bound = sqrt((w1_P)*(w2_P))*exp(-k_bha);
disp(P_error_bound);
```

Problem 2

The first step of this portion required us to form and plot data set X. A set that consisted of 500 points from ω_1 and 500 points from ω_2. We are assuming equal prior probabilities for both classes with $P(\omega_1) = P(\omega_2) = 0.5$ To create the distributions we used the `mvnrnd()` Matlab function. With a given mean, covariance, and the number of samples, the function can return points that fit the given parameters, as shown in Listing 6. The plots of the two distributions are shown in Figure 1.

Listing 6: Random Distribution

```
mu1 = [0 0]';
mu2 = [2 2]';
sigma12 = [1 0.25; 0.25 1];
isig12 = inv(sigma12);
samplenum = 500;

%part 2.a
r1 = mvnrnd(mu1, sigma12, samplenum); %for w1
r2 = mvnrnd(mu2, sigma12, samplenum); %for w2
```

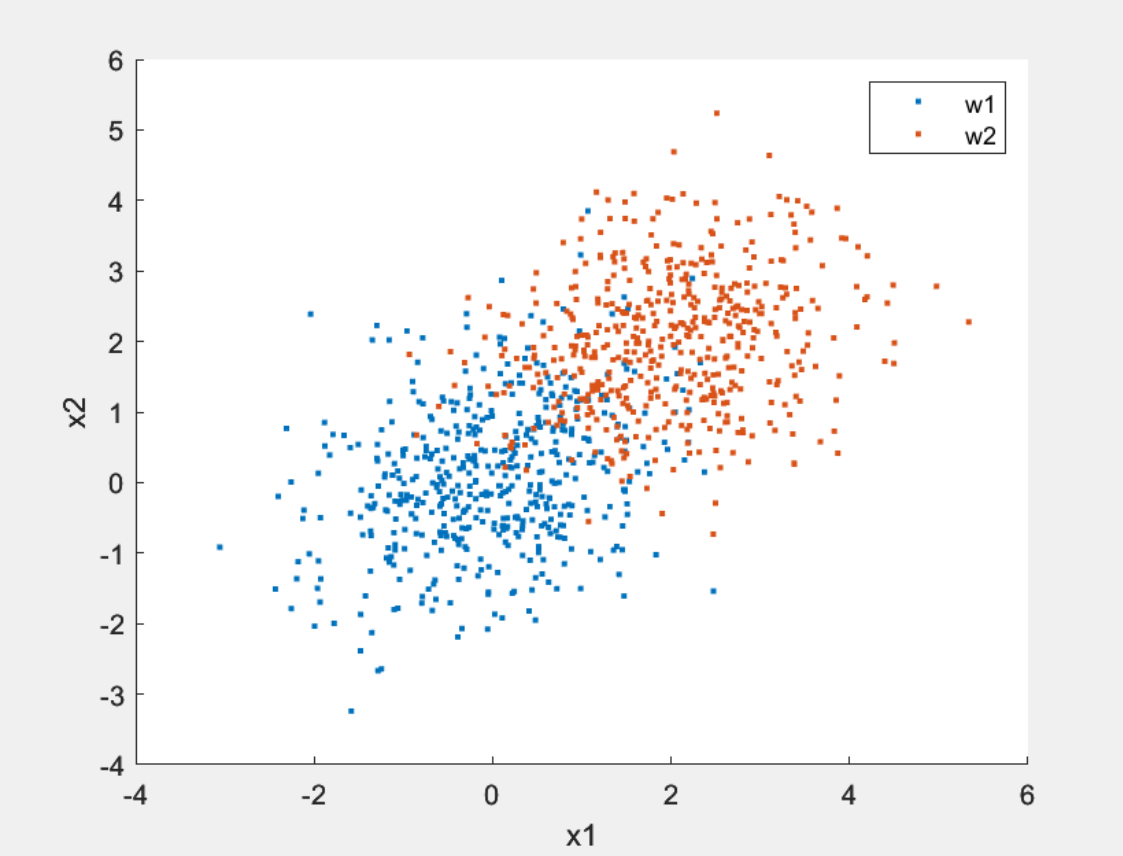

Figure 1: Plot for Random Gaussian Distributions

Next, a classifier was created based on the Bayes Decision Rule. To simplify the Bayes rule we can identify this problem as a Case II situation because $\Sigma_i = \Sigma$. Because the covariance matrices are equal, our decision boundary equation simplifies to $\mathbf{w}^t(\mathbf{x} - \mathbf{x}_0) = 0$. We use this equation and its definitions from our lecture notes to create the decision boundary equation shown in Listing 7. The general Bayes classifier can be shown in Figure 8.

To compute the error probability of the general Bayes classifier, we iterated through all points and checked to see if the class of each point aligned with the region it was on based on the decision boundary. If the point was on the wrong side of the decision boundary it was

Listing 7: Bayes Decision Boundary

```
%part 2.b
%classifier
%W' * (X - xo) = 0
mudiff = mu1-mu2;
w = isig12*(mudiff);
xo = 0.5*(mu1+mu2);

%bayes decision rule
range = -3:0.1:5;
dec_bound = ((w(1)*(range-xo(1)))/(-w(2)))+xo(2);
```

counted as a misclassified sample. Basically, it counted any red samples upper right corner and any green samples in the lower left corner.

Next, we were given loss functions and asked to create a new classifier that minimized the overall risk. The loss functions are shown below.

$$\lambda_{11} = \lambda_{22} = 0$$

$$\lambda_{12} = 1$$

$$\lambda_{21} = 0.005$$

The loss functions dictate how heavily a misclassification is weighed. In this case, λ_{12} is significantly greater than λ_{21} meaning that it is more costly to wrongly classify a sample as ω_1 than it is to wrongly classify a sample as ω_2. To minimize the overall risk, the general Bayes classifier was manipulated to incorporate the loss functions giving us a new classifier. To create the new classifier, the bias term of the decision boundary equation was recalculated and shown in Listing 8. The weighted classifier can be seen in Figure 9.

Listing 8: Loss Function Decision Boundary

```
%bayes rules with lambdas
%everything else stays the same
xo_new = xo - (log(lam21/lam12)*mudiff)/(mudiff'*isig12*mudiff);
dec_bound_new = ((w(1)*(range-xo_new(1)))/(-w(2)))+xo_new(2);
```

After, the samples were iterated through, and the overall risk was calculated. Every time a misclassified sample was found, it was multiplied by its lambda value and added to the misclassified total.

Results

Problem 1

Table 1 shows the actual error and Bhattacharyya maximum error bound for each classifier. As expected, the actual errors lie below the Bhattacharyya bounds.

TABLE 1: Error Result

Classifier	Empirical Error (%)	Bhattacharyya Bound (%)
x_1	30	47.40
x_1, x_2	40	45.63
x_1, x_2, x_3	15	41.18

Figures 2 through 7 show the histograms for the dataset. While most roughly follow a Gaussian distribution, Figures 3 and 4 appear to follow a linear distribution (or at the very least a non-Gaussian distribution). These likely contribute to the disproportionately large number of misclassifications of ω_2 and the generally poor performance with the x_1, x_2 classifier.

We believe it is a reasonable assumption to make that the more or dimensions you have when classifying data, the better your results should turn out. This is under the assumption that the data is Gaussianly distributed and that the dimensions are independent from one another. In our experiment, moving from using one feature to using two does not follow this pattern, however moving from two to three features does. We attribute these results to the assumption that our data is Gaussian when that isn't true for each feature. Therefore in a set of finite data there are cases where increasing dimensions can lead to greater amounts of empirical error. Particularly when the data being added dilutes the Gaussian nature of the prior data.

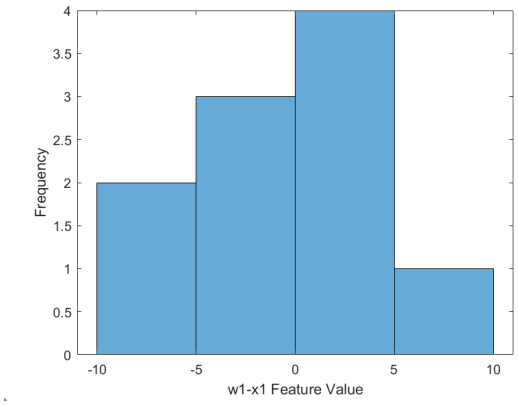

FIGURE 2: ω_1, x_1

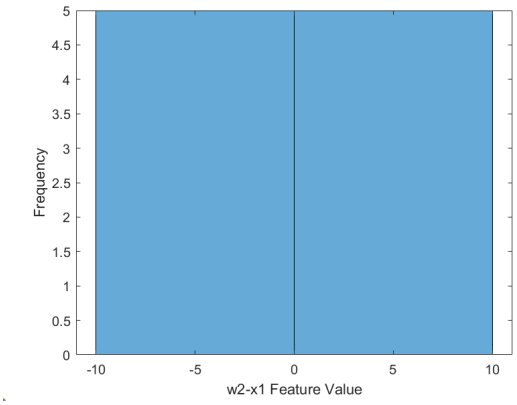

FIGURE 3: ω_2, x_1

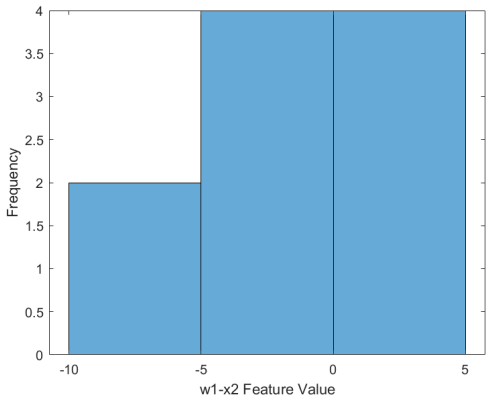

FIGURE 4: ω_1, x_2

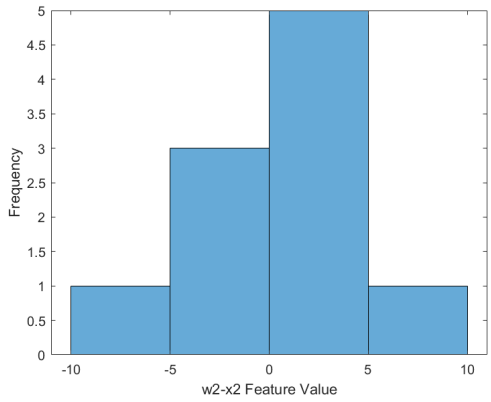

FIGURE 5: ω_2, x_2

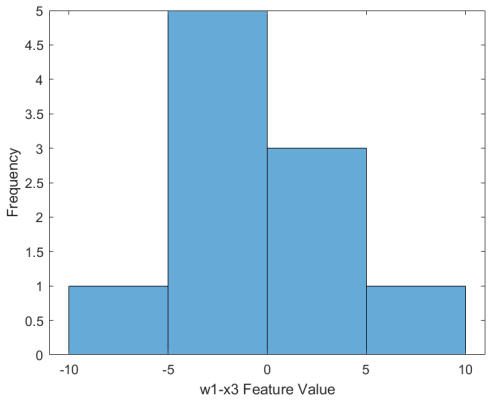

FIGURE 6: ω_1, x_3

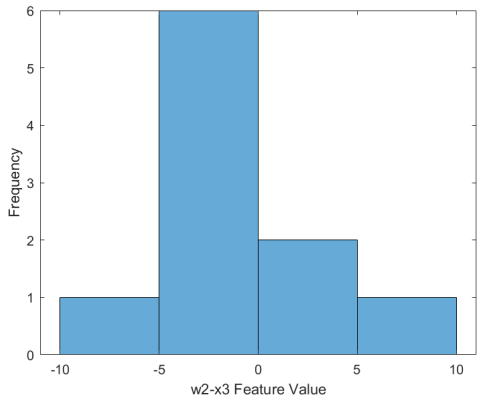

FIGURE 7: ω_2, x_3

Problem 2

Once the samples were plotted, it was easy to visualize a simple linear boundary that would separate the two classes fairly well. There would also be a small number of samples misclassified because the classifier should split them evenly between the centers. This can be seen in Figure 8.

The Bayes error rate was computed through Matlab:

- Misclassified samples: 107
- Error Rate = 10.7%

Using the loss functions, we presumed that the classifier would shift to minimize ω_2 being wrongly classified. The new computed decision boundary does exactly that. As shown in Figure 9, the new line moves almost entirely through the ω_1. This ensures that most of the samples from the second class are correctly classified.

The calculated overall risk:

- Misclassified samples: 2.035
- Error Rate = 0.2035%

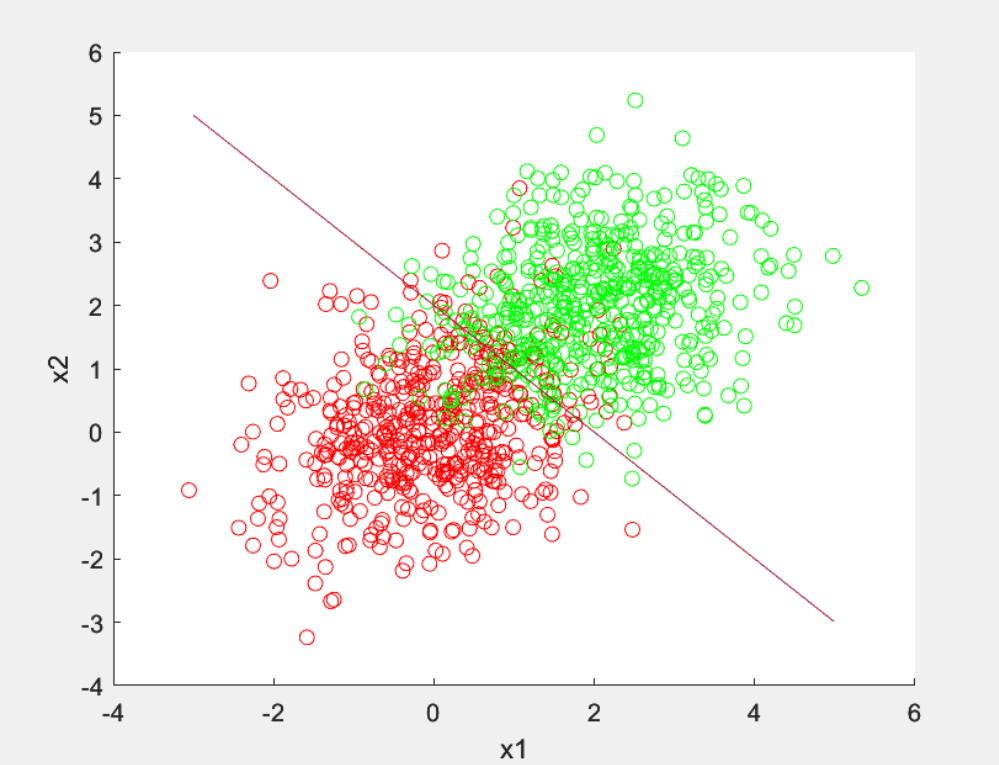

FIGURE 8: Bayes Decision Rule: ω_1(red) ω_2(green)

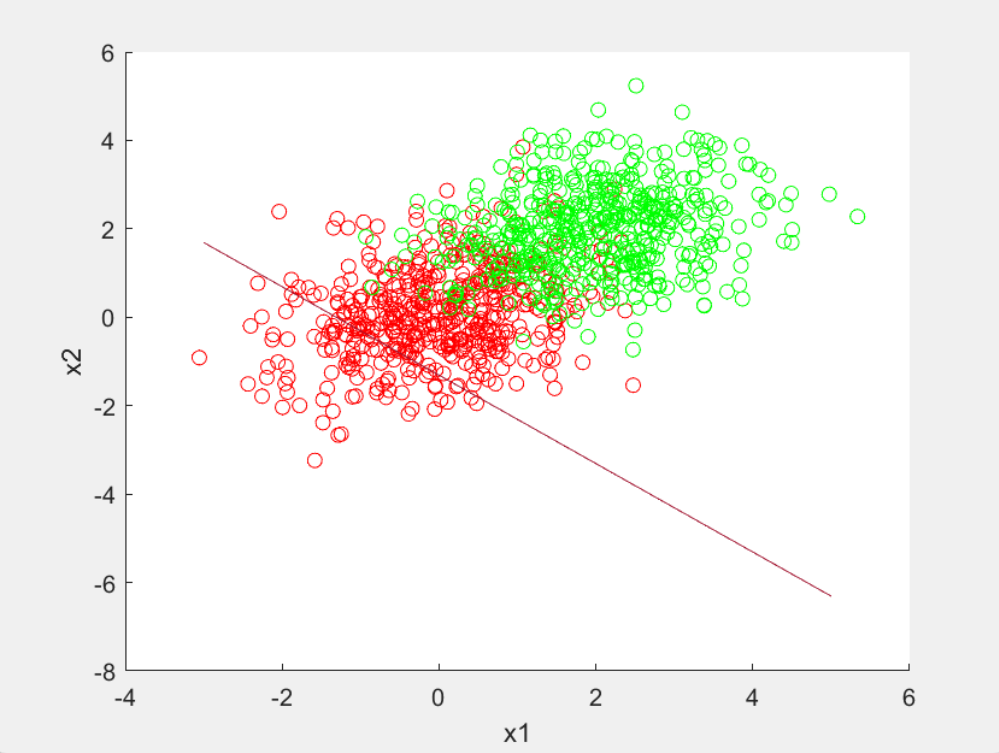

FIGURE 9: Loss Function Classifier

The calculated risk for the loss function is very very low compared to the general Bayes classifier. This can be attributed to the loss functions. Through inspection, we can see that the green samples are all on the correct side so all of class 2 is classified correctly. This is good because it is more costly to misclassify ω_2. Furthermore, even though a majority of red samples are misclassified, the loss functions weigh their error so lightly that they only amount to approximately 2 wrong samples.

Conclusion

Nathan Jaggers

This project was very valuable because it allowed us to put into practice what we have been discussing in lecture. Learning about the cases for Multivariate Gaussian Density and using them led to very different levels of understanding. It was interesting to see how error rates change with with different dimensions in part one and how the quality of the data can affect the final results. Also the exercise of learning how to incorporate loss functions into classification decisions was very valuable. I did the Battacharyya section of part 1 and created histograms to help explain why our accuracy was lower than expected. I also helped write up some of the discussion of our results for part 1.

Nicholas Brunet

My work with Problem 1 helped solidify my understanding of the Bayes classifier and the Battacharyya error bound. As with the other projects, I found that implementing this classifier from scratch helped me understand the concepts more than using an existing model. It was also interesting that more features did not consistently give us better results, but this also underscored the need for a larger dataset. And, compared to prior projects, I'm finally becoming comfortable with MATLAB syntax and can start to appreciate the tool's power.

Jordan Perlas

This project really helped me apply what we are learning in the lecture and helped me visualize the theory that we discuss in class. Observing the accuracy change with the different dimensions was really intriguing and it made sense to see our accuracy decrease because we were assuming Gaussian when graphically we could see that it wasn't. I believe it offers a great transition into our next unit. I also really enjoyed the second part of the project and seeing how to mathematically manipulate the boundary to optimize a specific goal. It also helped me visualize the significance of the loss functions and how each sample is weighted. My contribution to the project was initial work on Problem 1 and a bulk of Part 2. I also contributed to the analysis and write-up of the report.

Complete Code Listings

Part A

```
%% Part 1
close all;
clear;
clc;

dataset = [
   -5.01  -8.12  -3.68  -0.91  -0.18  -0.05   5.35   2.26   8.13;
   -5.43  -3.48  -3.54   1.30   2.06  -3.53   5.12   3.22  -2.66;
    1.08  -5.52   1.66  -7.75  -4.54  -0.95   1.34  -5.31  -9.87;
    0.86  -3.78  -4.11  -5.47   0.50   3.92   4.48   3.42   5.19;
   -2.67   0.63   7.39   6.14   5.72  -4.85   7.11   2.39   9.21;
    4.94   3.29   2.08   3.60   1.26   4.36   7.17   4.33  -0.98;
   -2.51   2.09  -2.59   5.37  -4.63  -3.65   5.75   3.97   6.65;
   -2.25  -2.13  -6.94   7.18   1.46  -6.66   0.77   0.27   2.41;
    5.56   2.86  -2.26  -7.39   1.17   6.30   0.90  -0.43  -8.71;
    1.03  -3.33   4.33  -7.50  -6.32  -0.31   3.52  -0.36   6.43;
];

w1_P = 0.5;
w2_P = 0.5;

d = 3; %dimensions

w1_x = dataset(:,1:d);
w2_x = dataset(:,4:d+3);

w1_mean = mean(w1_x)';
w2_mean = mean(w2_x)';

w1_cov = cov(w1_x);
w2_cov = cov(w2_x);

% Test
correct = 0;
Y = [ones(size(w1_x, 1), 1); zeros(size(w2_x, 1), 1)];
X = [w1_x; w2_x];
for i = 1:size(X, 1)
    x = X(i,:)';
    y = Y(i);
    g1_result = g(x, w1_mean, w1_cov, w1_P);
    g2_result = g(x, w2_mean, w2_cov, w2_P);
    correct = correct + ((g1_result - g2_result > 0) == y);
end
acc = correct / size(X, 1);
fprintf("Accuracy: %.2f\n", acc);

%%
% Bhattacharyya bound
```

```
49
50  %Assuming beta = 0.5 allows us to turn the chernoff equations into the
51  %following
52  mu2_mu1 = w2_mean - w1_mean;
53  cov2_cov1 = w1_cov+w2_cov;
54  ln_num = det(cov2_cov1/2);
55  ln_den = sqrt(det(w1_cov)*det(w2_cov));
56
57  k_bha = (1/8)*mu2_mu1'*inv(cov2_cov1/2)*mu2_mu1+(1/2)*log(ln_num/ln_den);
58
59  P_error_bound = sqrt((w1_P)*(w2_P))*exp(-k_bha);
60  disp(P_error_bound);
61
62  %%
63  %creating histograms for data
64
65  % 1 feature
66  figure;
67  histogram(dataset(:,1));
68  xlabel("w1-x1 Feature Value");
69  ylabel("Frequency");
70  figure;
71  histogram(dataset(:,4));
72  xlabel("w2-x1 Feature Value");
73  ylabel("Frequency");
74
75  % 2 feature
76  figure;
77  histogram(dataset(:,2));
78  xlabel("w1-x2 Feature Value");
79  ylabel("Frequency");
80  figure;
81  histogram(dataset(:,5));
82  xlabel("w2-x2 Feature Value");
83  ylabel("Frequency");
84
85  % 3 feature
86  figure;
87  histogram(dataset(:,3));
88  xlabel("w1-x3 Feature Value");
89  ylabel("Frequency");
90  figure;
91  histogram(dataset(:,6));
92  xlabel("w2-x3 Feature Value");
93  ylabel("Frequency");
94
95  %% Functions
96
97  % Calculate parameters once in separate function if this takes too long
98  function result = g(x, mean, cov, P)
99      cov_i = inv(cov);
100     W = -0.5 * cov_i;
101     w = cov_i * mean;
102     w0 = -0.5 * (mean' * cov_i * mean) - 0.5 * log(det(cov)) + log(P);
```

```
    result = x'*W*x + w'*x + w0;
end
```

Part B

```
%Part 2
clear

mu1 = [0 0]';
mu2 = [2 2]';
sigma12 = [1 0.25; 0.25 1];
isig12 = inv(sigma12);
samplenum = 500;

%part 2.a
r1 = mvnrnd(mu1, sigma12, samplenum); %for w1
r2 = mvnrnd(mu2, sigma12, samplenum); %for w2

%lambdas
lam11=0; lam22=0; lam12=1; lam21=0.005;
%more costly to choose w1 when actually w2
%shrink R1 and make R2 bigger

%part 2.b
%classifier
%W' * (X - xo) = 0
mudiff = mu1-mu2;
w = isig12*(mudiff);
xo = 0.5*(mu1+mu2);

%bayes decision rule
range = -3:0.1:5;
dec_bound = ((w(1)*(range-xo(1)))/(-w(2)))+xo(2);

%bayes rules with lambdas
%everything else stays the same
xo_new = xo - (log(lam21/lam12)*mudiff)/(mudiff'*isig12*mudiff);
dec_bound_new = ((w(1)*(range-xo_new(1)))/(-w(2)))+xo_new(2);


%assign classes to plots
r1_class = [ones(samplenum,1) r1];
r2_class = [-1*ones(samplenum,1) r2];
datasetX = [r1_class; r2_class];
%%
%for part e
misclass_new = 0; %starting counter for misclassified samples

```

```
for i = 1:1000
    gx = (w(1)*(datasetX(i,2)-xo_new(1))) + (w(2)*(datasetX(i,3)-xo_new(2)));
    if gx >= 0 %want w1 (classified as 1)
        if datasetX(i, 1) ~= 1 %classified as w1 but is w2, thats lambda12
            misclass_new = misclass_new + 1*lam12;
        end
    else %want w2 (classified as -1)
        if datasetX(i, 1) ~= -1 %classified as w2 but is w1, thats lambda21
            misclass_new = misclass_new + 1*lam21;
        end
    end
end

disp(['misclassified samples: ' num2str(misclass_new)])
disp(['error rate = ' num2str((misclass_new/1000)*100) '%'])
%%
%plot for part d
for i = 1:1000
    %gx = (w(1)*(datasetX(i,2)-xo(1))) + (w(2)*(datasetX(i,3)-xo(2)));
    if datasetX(i,1) == 1
    %if gx >= 0 %depening on what she actually wants for this part the only thing that
    ↪  needs to change is this if
        hold on
        scatter(datasetX(i,2), datasetX(i,3), 'r')
    else
        hold on
        scatter(datasetX(i,2), datasetX(i,3), 'g')
    end
end
plot(range, dec_bound_new)
xlabel('x1')
ylabel('x2')
hold off

%%
%for part c
misclass = 0; %starting counter for misclassified samples

for i = 1:1000
    gx = (w(1)*(datasetX(i,2)-xo(1))) + (w(2)*(datasetX(i,3)-xo(2)));
    if gx >= 0 %want w1 (classified as 1)
        if datasetX(i, 1) ~= 1
            misclass = misclass + 1;
        end
    else %want w2 (classified as -1)
        if datasetX(i, 1) ~= -1
            misclass = misclass + 1;
        end
    end
end

disp(['Misclassified Samples: ' num2str(misclass)])
disp(['Error Rate = ' num2str((misclass/1000)*100) '%'])
%%
```

```
%plot for part 2
for i = 1:1000
    gx = (w(1)*(datasetX(i,2)-xo(1))) + (w(2)*(datasetX(i,3)-xo(2)));
    if datasetX(i,1) == 1
    %if gx >= 0 %depening on what she actually wants for this part the only thing that
    ↪  needs to change is this if
        hold on
        scatter(datasetX(i,2), datasetX(i,3), 'r')
    else
        hold on
        scatter(datasetX(i,2), datasetX(i,3), 'g')
    end
end
plot(range, dec_bound)
xlabel('x1')
ylabel('x2')
hold off

%%
%plot for part 1
figure
hold on
scatter(r1(:,1), r1(:,2), ".")
scatter(r2(:,1), r2(:,2), ".")
%plot(range, dec_bound)
xlabel('x1')
ylabel('x2')
legend("w1", "w2")
hold off
```