Stoch Project 4: Detection

Rebecca Gartenberg and Sophie Jaro

April 17, 2020

1 Matlab Part 1 Radar Detection

1.1 Procedure

The purpose of this part of the project is to determine if a target is present or absent. The MAP (maximum aposteriori probability) decision rule is used to make this determination given the detected signal. Implementation of the MAP decision rule requires comparing posterior probabilities P(H1|y) and P(H0|y). The hypothesis whose posterior probability is greater is selected.

In Part A of the project, the observation signal is described by Y = A + X if the target is present. A is a known constant and X is a zero mean Gaussian distribution. The signal is Y = X if no target is present, with X being the same zero mean Gaussian distribution. Thus, this part of the project compared hypotheses with different means and the same variances. This affected our choice of gamma, which defines the threshold for the decision of how to classify the state of the target. SNR for Part A was defined as the ratio of A to the variance of distribution X. The variance of X was changed for this simulation to demonstrate the effectiveness of the MAP detection rule at various SNR values.

In Part E of the project, the observation signal is described by Y = A + X when the target is present. A is a known constant and X is a zero mean Gaussian distribution. The signal is Y = A + Z when no target is present. A is the same known constant and Z is a zero mean Gaussian distribution. Thus, this part of the project compared hypotheses with the same means and different variances. This affected our choice of gamma. The SNR values for Part E were defined as the ratio of the variance of distribution Z to the variance of distribution X. The variances of X and Z were changed for this simulation to demonstrate the effectiveness of the MAP detection rule at various SNR values.

For parts A and E, s vector of 1000 0s and 1s, which represent target not present and target present respectively, is generated randomly with a prior of 0.8 on target not present and 0.2 on target present. Based on the observed signal Y, we predict whether or not the target is present at each sample y. For each observation a decision needs to be made about how to classify the state of the target. Using the MAP rule, this can be done either by computing and comparing the conditional probabilities of the observations as we did in part A, or it can be done by comparing the observations to a threshold as we did in part E.

1.2 Results

The plot in Figure 1 shows ROC curves for five different SNR values for two hypotheses with the same variance and different means. The SNR values, which are defined by mean divided by variance, are 10, 4, 2, 0.5, 0.05. It can be seen in the plot that a higher SNR value gets an ROC curve that is closer to perfect than a curve with a low SNR value. The ROC curve based on an SNR value of 10 is the best curve in the plot. The ROC curve with an SNR value of 0.05 is close to a 45 degree line, the worst ROC possible. This shows that a higher SNR produces better detection results. This makes sense since a high SNR value characterizes a clearer signal with less noise. More noise would make it more difficult to properly detect targets present and absent. The stars on each of the ROC curves mark the spot where the threshold eta is equal to P0/10*P1. This threshold indicates that missing a target is 10 times worse than falsely detecting a target. The star indicates the probability of detection versus probability of false alarm when that threshold is used.

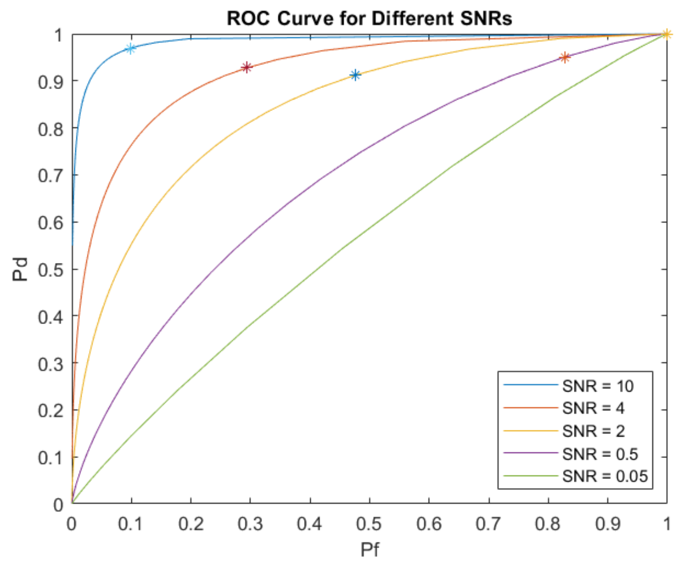

Figure 1: ROC for Part A: Different Mean, Same Variances. The * marks the minimal conditional risk.

The plot in Figure 2 shows ROC curves for five different SNR values for two hypotheses with different variances and the same means. The SNR values, which are defined by the ratio of the variances of the two hypotheses' distributions, are 1.01, 2, 10, 50, 100. It can be seen that higher SNR values have a better detection versus false positive rate. The curve with the lowest SNR of 0.05 is close to a 45 degree line, which is the worst possible ROC curve. This again makes sense because more noise will lead to more false positives.

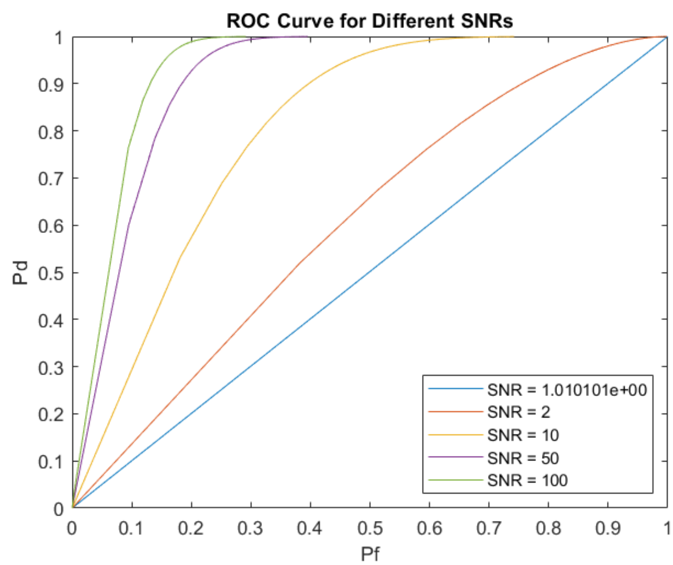

Figure 2: ROC for Part E: Same Mean, Different Variances

The plot in Figure 3 shows the expected cost for a SNR value of 4 when mean is 1 and variance is 0.25 when it is 10 times more costly to miss a target than to falsely detect a target. The cost is maximized at approximately equal prior probabilities of target present and target absent because the decision is relatively arbitrary for equiprobable states. This leads to more misses. When the prior probability of target present is low, the expected cost is lower because there are fewer targets to miss. When the probability of target present is high, the expected cost is lower again because with a higher prior probability, there will be fewer misses. The plot is skewed towards the left because the cost of a miss is higher than the cost of a false alarm. The model is more likely to miss a detection when the probability that the target is present is low.

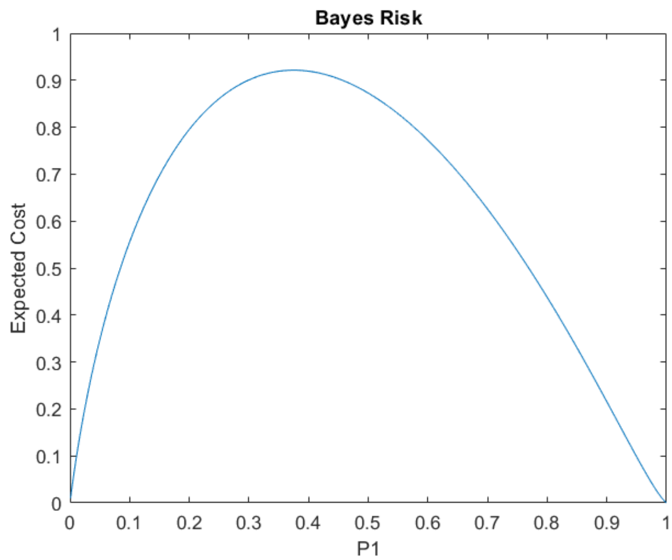

Figure 3: Bayes Risk aka Expected Cost

2 Matlab Part 2 Introduction to pattern classification and machine learning

2.1 Procedure

The purpose of Part 2 is to use machine learning in order to classify the data in a dataset into 3 classes based on 4 features. The first step is to divide the data of each class into equally-sized training and testing sets. In this simple case, training the data involves computing the sample means for each feature and the covariance matrix of the training data for each class. The next step is to compute the likelihood that the data in the training set belongs to each of the 3 classes. The likelihood with the highest value is the label that is assigned to that data point. The likelihood were computed using the MVNPDF function with the inputs corresponding to the testing data, the mean vector for a given class and the covariance matrix for that class. Although the labels of the testing class were not used when labeling the testing data, we can use those labels in order to figure out how well our model classified the data. A confusion matrix is displayed in Figure 4. The confusion matrix shows the true class versus the predicted class. In our case the classifier performed very well. All of the data points in class 1 were correctly predicted. All of the data points in class 3 were also correctly predicted. 3 of the data points in class 2 were incorrectly classified as class 3. The probability of error was 0.0133. The percent error was 1.33 % which means our classifier performed well, being incorrect only 1.33 % of the time.

2.2 Results

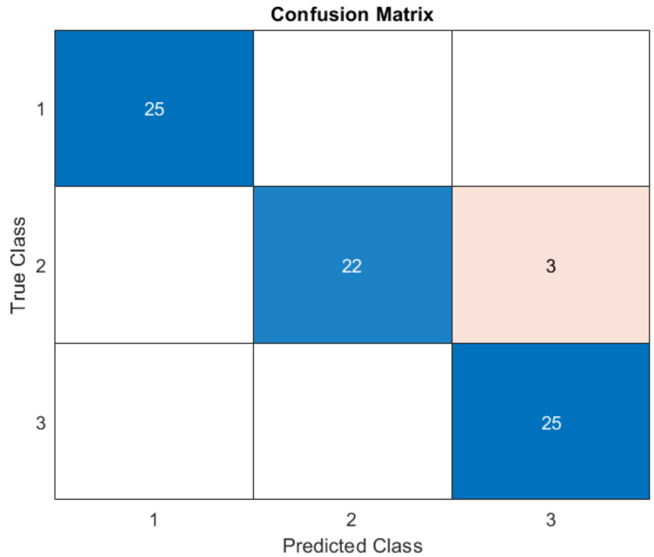

Figure 4: Confusion Matrix

3 Appendix: MATLAB code

```
% Part 1: Radar Detection

A = 1; % a known constant
var_vec = [.1 .25 0.5 2 20]; % variances of X for part A (different mean, same variance)
var_vec_e = [.99 .25 .5 0.01 0.01]; % variances of X for part E (same mean, different ...
    variances)
varz_vec_e = [1 .5 5 .5 1]; % variances of Z for part E (same mean, different variances)


for j = 1:length(var_vec) % This loops through variances to test the rule for several SNRs

    variance = var_vec(j);
    variance_e = var_vec_e(j);
    variancez_e = varz_vec_e(j);

    sigma = sqrt(variance);
    sigma_e = sqrt(variance_e);
    sigmaz_e = sqrt(variancez_e);

    SNR(j) = A/variance; % SNR for Part A is the ratio of the mean to the variance
    SNR_e(j) = variancez_e/variance_e; % SNR for Part E is the ratio of the variance of ...
        dist. X to the variance of dist. Z

    P0 = 0.8; % The a priori probability the target is not present is 0.8
    P1 = 0.2; % The a priori probability the target is present is 0.2
    eta = P0/P1; % threshhold, assuming MPE cost assignment

    N = 1000; % Number of iterations
    for k = 1:N
        X = sigma.*randn(1,N); % zero mean Gaussian noise with variance sigma
        X_e = sigma_e.*randn(1,N); % zero mean Gaussian noise with variance sigma
        Z_e = sigmaz_e.*randn(1,N); % zero mean Gaussian noise with variance sigmaz
```

```
        target = zeros(1,N);
        target_e = zeros(1,N);

        % Generate data for Part A
        x = rand(1, N);
        target(find(x ≤ P0)) = 0; % 80% of the time the target is not present
        target(find(x > P0)) = 1; % 20% of the time the target is present
        Y(find(x ≤ P0)) = X(find(x ≤ 0.8)); % Y = A when target not present
        Y(find(x > P0)) = A + X(find(x > 0.8)); % Y = A + X when target not present

        % Generate data for Part E
        y = rand(1, N);
        target_e(find(y ≤ P0)) = 0; % 80% of the time the target is not present
        target_e(find(y > P0)) = 1; % 20% of the time the target is present
        Y_e(find(y ≤ P0)) = A + Z_e(find(y ≤ 0.8)); % Y = A + Z when target not present
        Y_e(find(y > P0)) = A + X_e(find(y > 0.8)); % Y = A + X when target present

        % Compute Conditional Probabilities P(H1|y) and P(H0|y) for MAP Rule for Part A
        P_y_given_H1 = (1/sqrt(2*pi*variance))*exp((-(1/2)*(Y-A).^2)/variance); % Equation ...
            (8.19) in "Detection Theory"
        P_H1_given_y = P_y_given_H1*P1; % P(H1|y) = P(y|H1)*P1
        P_y_given_H0 = (1/sqrt(2*pi*variance))*exp((-(1/2)*(Y).^2)/variance); %(8.19)
        P_H0_given_y = P_y_given_H0*P0; % P(H0|y) = P(y|H0)*P0

        % Compute Conditional Probabilities P(H1|y) and P(H0|y) for MAP Rule for Part E
        P_y_given_H1_e = (1/sqrt(2*pi*variance_e))*exp((-(1/2)*(Y_e).^2)/variance_e); %(8.19)
        P_H1_given_y_e = P_y_given_H1_e*P1; % P(H1|y) = P(y|H1)*P1
        P_y_given_H0_e = (1/sqrt(2*pi*variancez_e))*exp((-(1/2)*(Y_e).^2)/variancez_e); ...
            %(8.19)
        P_H0_given_y_e = P_y_given_H0_e*P0; % P(H0|y) = P(y|H0)*P0

        % MAP Rule for Part A (8.17)
        target_hat = zeros(1,length(target)); % Predicted hypothesis for Part A
        target_hat(find(P_H1_given_y ≤ P_H0_given_y)) = 0; % Choose H0 when P(H0|y) > P(H1|y)
        target_hat(find(P_H1_given_y > P_H0_given_y)) = 1; % Choose H1 when P(H0|y) < P(H1|y)

        % MAP Rule for Part E (8.17)
        target_hat_e = zeros(1,length(target_e)); % Predicted hypothesis for Part E
        gamma_e = ...
            2*((variance_e*variancez_e)/(variancez_e-variance_e))*log((sigmaz_e/sigma_e)*eta) ...
            ; %(8.24)
        target_hat_e(find(Y_e.^2 < gamma_e)) = 0; % Choose H0 when y^2 < gamma, the ...
            decision threshold
        target_hat_e(find(Y_e.^2 > gamma_e)) = 1; % Choose H1 when y^2 > gamma, the ...
            decision threshold

        % Pd: If target is 1 and target_hat is 1 then detection
        % Pm: If target is 1 and target_hat is 0 then missed
        % Pf: If target is 0 and target_hat is 1 then false positive

        % Count number of detections, misses, and false alarms for Part A
        detection = length(intersect(find(target == 1), find(target_hat == 1)));
        missed = length(intersect(find(target == 1), find(target_hat == 0)));
        falsealarm = length(intersect(find(target == 0), find(target_hat == 1)));

        % Count number of detections, misses, and false alarms for Part E
        detection_e = length(intersect(find(target_e == 1), find(target_hat_e == 1)));
        missed_e = length(intersect(find(target_e == 1), find(target_hat_e == 0)));
        falsealarm_e = length(intersect(find(target_e == 0), find(target_hat_e == 1)));

        % Compute probability of detections, misses, and false alarms for Part A
        % with Prob(H1 guessed | H0 True) = P(H1 guessed and H0 True)/P(H0 True)
        Pd = detection/length(find(target == 1));
        Pm = missed/length(find(target == 1));
```

```
        Pf = falsealarm/length(find(target == 0));

        % Compute probability of detections, misses, and false alarms for Part A
        % with Prob(H1 guessed | H0 True) = P(H1 guessed and H0 True)/P(H0 True)
        Pd_e = detection_e/length(find(target_e == 1));
        Pm_e = missed_e/length(find(target_e == 1));
        Pf_e = falsealarm_e/length(find(target_e == 0));

        % Compute probability of error for each iteration
        Perror_exp_vec(k) = (1-Pd)*P1 + Pf*P0; %(8.36)
        Perror_exp_vec_e(k) = (1-Pd_e)*P1 + Pf_e*P0; %(8.36)

    end % end of iterations loop

    % Average probability of error (experimental) for Part A
    Perror_exp(j) = mean(Perror_exp_vec)
    % Average probability of error (experimental) for Part E
    Perror_exp_e(j) = mean(Perror_exp_vec_e)

    % Compute threshhold eta for Part A (MPE) and Part C (Weighted Costs)
    eta = P0/P1; %(8.11), assuming MPE cost assigment
    eta_partc = eta/10; % Cost of missed C01 = 10 and false alarm C10 = 1

    % Compute gamma for Part A, Part C
    gamma = (A/2) + (variance*log(eta))/A; %(8.27)
    gamma_partc = (A/2) + (variance*log(eta_partc))/A;

    % Calculate Pd for Part A, Part C
    Pd_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l-A).^2)/(2*variance)); %(8.46)
    Pd = integral(Pd_int, gamma, inf); % area under A + X Gaussian from gamma to infinity
    Pd_c(j) = integral(Pd_int, gamma_partc, inf); % area under A + X Gaussian from gamma_c ...
        to infinity

    % Calculate Pf for Part A, Part C
    Pf_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l).^2)/(2*variance)); %(8.47)
    Pf = integral(Pf_int, gamma, inf); % area under X Gaussian from gamma to infinity
    Pf_c(j) = integral(Pf_int, gamma_partc, inf); % area under X Gaussian from gamma_c to ...
        infinity

    % Calculate Theoretical Probability for Part A
    Perror_theo(j) = (1-Pd)*P1 + Pf*P0 %(8.36)

    % Compute gamma for Part E
    gamma_e = ...
        2*((variance_e*variancez_e)/(variancez_e-variance_e))*log((sigmaz_e/sigma_e)*eta) ...
        ; %(8.24)

    % Compute Pd for Part E
    Pd_int_e = @(l) (1/sqrt(2*pi*variance_e))*exp(-((l-A).^2)/(2*variance_e)); %(8.46)
    Pd_e = integral(Pd_int_e, -inf, -sqrt(gamma_e)) + integral(Pd_int_e, sqrt(gamma_e), inf);

    % Compute Pf for Part E
    Pf_int_e = @(l) (1/sqrt(2*pi*variancez_e))*exp(-((l-A).^2)/(2*variancez_e)); %(8.47)
    Pf_e = integral(Pf_int_e, -inf, -sqrt(gamma_e)) + integral(Pf_int_e, sqrt(gamma_e), inf);

    % Calculate Theoretical Probability for Part E
    Perror_theo_e(j) = (1-Pd_e)*P1 + Pf_e*P0 %(8.36)

    % Part B: Plot ROC for various SNR values
    eta_vec = linspace(0,100,1000);
    gamma_vec = (A/2) + (variance*log(eta_vec))/A; %(8.27)

    for i = 1:length(gamma_vec)
        % Compute Pd for different threshholds
        Pd_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l-A).^2)/(2*variance)); %(8.46)
```

```
        Pd(i) = integral(Pd_int, gamma_vec(i), inf);

        % Compute Pf for different threshholds
        Pf_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l).^2)/(2*variance)); %(8.47)
        Pf(i) = integral(Pf_int, gamma_vec(i), inf);
    end

    % Plot ROC Curve for Part A (Different Mean, Same Variance)
    figure(1)
    plot(Pf,Pd)
    xlim([0,1])
    ylim([0,1])
    xlabel('Pf')
    ylabel('Pd')
    hold on

    % Plot ROC Curve for Part B (Same Mean, Difference Variances)
    eta_vec_e = linspace(0,100,1000);
    gamma_vec_e = ...
        2*((variance_e*variancez_e)/(variancez_e-variance_e))*log((sigmaz_e/sigma_e)*eta_vec_e); ...
        % (8.24)

    for i = 1:length(gamma_vec_e)
        Pd_int_e = @(l) (1/sqrt(2*pi*variance_e))*exp(-((l).^2)/(2*variance_e)); %(8.46)
        Pd_e(i) = 2*integral(Pd_int_e, 0, real(sqrt(gamma_vec_e(i))));

        Pf_int_e = @(l) (1/sqrt(2*pi*variancez_e))*exp(-((l).^2)/(2*variancez_e)); %(8.47)
        Pf_e(i) = 2*integral(Pf_int_e, 0, real(sqrt(gamma_vec_e(i))));
    end

    figure(2)
    plot(Pf_e,Pd_e)
    xlim([0,1])
    ylim([0,1])
    xlabel('Pf')
    ylabel('Pd')
    hold on

end % end of variances loop

figure(1)
legend(sprintf('SNR = %d', SNR(1)),sprintf('SNR = %d', SNR(2)),...
    sprintf('SNR = %d', SNR(3)),sprintf('SNR = %g', SNR(4)),sprintf('SNR = %g', ...
        SNR(5)),'Location','southeast')
title('ROC Curve for Different SNRs')
hold on;

figure(2)
legend(sprintf('SNR = %d', SNR_e(1)),sprintf('SNR = %d', SNR_e(2)),...
    sprintf('SNR = %d', SNR_e(3)),  sprintf('SNR = %d', SNR_e(4)), sprintf('SNR = %g', ...
        SNR_e(5)), 'Location' ,'southeast');%,sprintf('SNR = %g', SNR(4)),sprintf('SNR = ...
        %g', SNR(5)),'Location','southeast')
title('ROC Curve for Different SNRs')
hold on;

% Part C: Plot point with minimum conditional risk for each SNR
figure(1)
plot(Pf_c(1),Pd_c(1),'*','HandleVisibility','Off')
plot(Pf_c(2),Pd_c(2),'*','HandleVisibility','Off')
plot(Pf_c(3),Pd_c(3),'*','HandleVisibility','Off')
plot(Pf_c(4),Pd_c(4),'*','HandleVisibility','Off')
plot(Pf_c(5),Pd_c(5),'*','HandleVisibility','Off')

% Part D: Plot a priori probabilities versus Bayes Risk
C00 = 0;
```

```matlab
C01 = 10;
C10 = 1;
C11 = 0;
variance = 0.25;
A = 1;
SNR = A/variance;

P0 = linspace(0,1,1000);
P1 = 1 - P0;

Pd_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l-A).^2)/(2*variance)); %(8.46)
Pf_int = @(l) (1/sqrt(2*pi*variance))*exp(-((l).^2)/(2*variance)); %(8.47)

for i = 1:1000
    eta = P0(i)/P1(i); %(8.11)
    gamma = (A/2) + (variance*log(eta))/A; %(8.27)

    Pd = integral(Pd_int, gamma, inf);
    Pf = integral(Pf_int, gamma, inf);

    ExpCost(i) = C00*P0(i) + C01*P1(i) + (C10-C00)*P0(i)*Pf - (C01-C11)*P1(i)*Pd;
end

figure
plot(P1,ExpCost)
xlabel('P1')
ylabel('Expected Cost')
title('Bayes Risk')


%% Part 2 --- Introduction to pattern classification and machine learning

% Iris data has 3 classes and 4 features
load('Iris.mat','features','labels')

% labels are appended onto features for later validatiion of our classifier
data = [labels features];

% Split into 3 classes
dataClass1 = data(1:50,:);
dataClass2 = data(51:100,:);
dataClass3 = data(101:150,:);

% Randomize each class
rnddata1 = dataClass1(randperm(size(dataClass1, 1)), :);
rnddata2 = dataClass2(randperm(size(dataClass2, 1)), :);
rnddata3 = dataClass3(randperm(size(dataClass3, 1)), :);

% Take half of each class for training
training1 = rnddata1(1:25,:);
training2 = rnddata2(1:25,:);
training3 = rnddata3(1:25,:);

% Take half of each class for testing
testing1 = rnddata1(26:50,:);
testing2 = rnddata2(26:50,:);
testing3 = rnddata3(26:50,:);
testing = [testing1; testing2; testing3];

% randomize testing data
testingrnd = testing(randperm(size(testing, 1)),:);

% Mean of each feature by class
for i = 2:5
    meanvec1(i-1) = mean(training1(:,i));
```

```
    meanvec2(i-1) = mean(training2(:,i));
    meanvec3(i-1) = mean(training3(:,i));
end

% Covariance by class
cov1 = cov(training1(:,2:5));
cov2 = cov(training2(:,2:5));
cov3 = cov(training3(:,2:5));

% Get the liklihood that the training data belongs to each class
for i = 1:length(testing)
    lkhood_class1(i) = mvnpdf(testingrnd(i,2:5),meanvec1,cov1);
    lkhood_class2(i) = mvnpdf(testingrnd(i,2:5),meanvec2,cov2);
    lkhood_class3(i) = mvnpdf(testingrnd(i,2:5),meanvec3,cov3);
end

% Assign labels to training data based on liklihood
for i = 1:length(lkhood_class1)
    if (lkhood_class1(i) > lkhood_class2(i)) || (lkhood_class1(i) > lkhood_class3(i))
        classes(i,1) = 1;
    else if lkhood_class2(i) > lkhood_class3(i)
            classes(i,1) = 2;
        else
            classes(i,1) = 3;
        end
    end

end

% Compare labels to known labels
compare = (classes == testingrnd(:,1));

% Confusion matrix
figure
C = confusionmat(testingrnd(:,1),classes);
confusionchart(C);
title('Confusion Matrix')

% Probability of error
Perror = (1/3)*sum(compare == 0)/length(classes) % (8.115)
```