EE 516
Pattern Recognition
Prof. Jane Zhang

Project 5

May 31, 2023

Nicholas Brunet
Nathan Jaggers
Jordan Perlas

Introduction

For this project, we were tasked with working with two separate data sets. The first is the Iris Plant classification data set. This set consists of 50 samples, containing four features, from 3 species of Iris. The other data set is almost 600 samples of benign and malignant breast masses with 30 logged features. Our objective is to parse through the data and extract relevant and robust features to develop and train a Bayes and Naive Bayes classifier for both data sets. We are also tasked with evaluating the classifier and reporting results.

Iris Plant

Data

The Iris Data Set used is a popular data set and built into Matlab so it can be called using a single load instruction. The data set is hosted by UC Irvine and can be found at `http://archive.ics.uci.edu/ml/datasets/Iris`.

As stated before it is a dataset of three types of Iris plants (Setosa, Versicolour and Virginica) with 50 samples each, totaling 150 samples. Each sample is described by four features: sepal length, sepal width, petal length and petal width.

Procedure

To start, we loaded the data into Matlab using the load command and that gave us access to the 4 features mentioned previously and the associated class for each sample. Using this data set we created histograms of the different plants' features (Figures 1 – 4 in the Result section) which allowed us to analyze the separability of the different plant classes. The features that showed high separability were chosen and would later be used when creating the Bayes and Naive Bayes classifier.

Listing 1: Separating Iris Data into Classes

```
load("fisheriris.mat")

setosa_X = meas(1:50,:); % w1
versicolor_X = meas(51:100,:); % w2
virginica_X = meas(101:150,:); % w3

setosa_Y = ones(50, 1);
versicolor_Y = ones(50, 1) * 2;
virginica_Y = ones(50, 1) * 3;
Y = [setosa_Y; versicolor_Y; virginica_Y];
```

The separable features were used in training the Bayes Classifier. Training consisted of finding the mean and covariance matrices of the features and using them to develop discriminant functions. The challenge with this data set is that there are not a lot of samples

which makes it difficult to separate the data into a training and test set. To remedy this we employ a Leave-One-Out (LOO) technique which allows us to use the entire data set for training and testing by rotating out samples used for either set. This is accomplished by doing 150 iterations of training (the same amount of samples we have) where each iteration, we leave a single sample out and then test if our classifier correctly distinguishes the sample. To evaluate the effectiveness of the classifier we determine its accuracy when classifying samples in the LOO testing and also construct a confusion matrix to display the performance per class.

Listing 2: Selection of Test Sample

```
for i = 1:150
    % create train/test split
    setosa_X_train = setosa_X;
    versicolor_X_train = versicolor_X;
    virginica_X_train = virginica_X;
    if i <= 50
        x = setosa_X(i,:);
        setosa_X_train(i,:) = [];
    elseif i <= 100
        x = versicolor_X(i-50,:);
        versicolor_X_train(i-50,:) = [];
    else
        x = virginica_X(i-100,:);
        virginica_X_train(i-100,:) = [];
    end
    y = Y(i);
    ...
end
```

Lastly we explore the effectiveness of using a Naive Bayes Classifier for this data set. The procedure stays relatively constant when comparing to the Bayes Classifier. The only significant difference is during training. The assumption of the Naive Bayes Classifier is that the features are independent of each other. This means we are able to independently find and multiply the different features probability distributions when determining what class a sample belongs to. So instead of finding a mean and covariance for the whole class, we end up finding them for the features of the different classes. Using this training method, we gather results the same as described for the Bayes Classifier.

Results

Figures 1 through 4 show the histograms for the dataset. Here we can see that the best features for classification are petal length and petal width because of their high degree of separability.

Table 1 shows the Bayes and Naive Bayes error for the classifier. Figures 5 and 6 show the confusion matrix for the Leave-One-Out process using Bayes and Naive Bayes Classifiers.

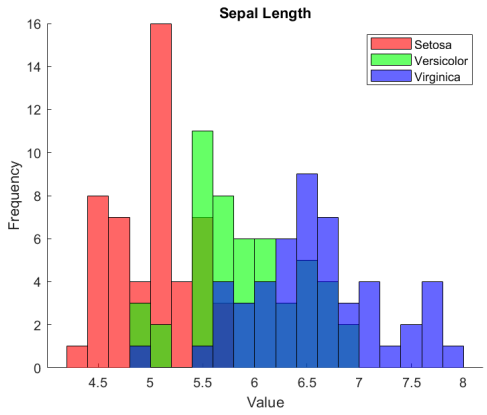

FIGURE 1: Sepal Length

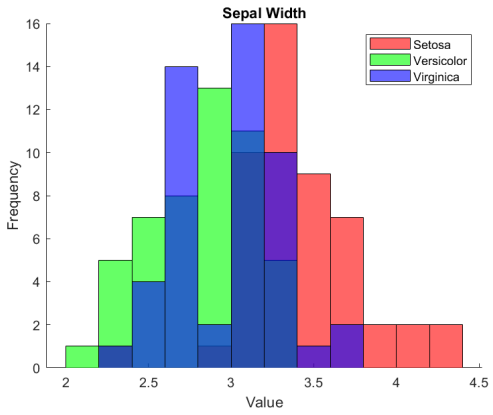

FIGURE 2: Sepal Width

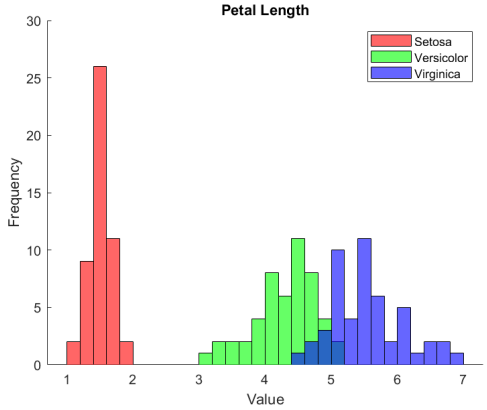

FIGURE 3: Petal Length

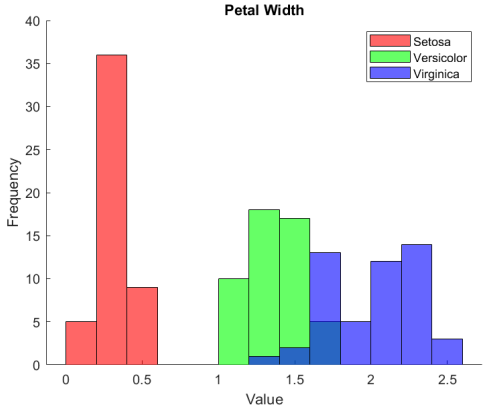

FIGURE 4: Petal Width

TABLE 1: Error Result

Classifier	Empirical Error (%)
Bayes	3.3
Naive Bayes	4

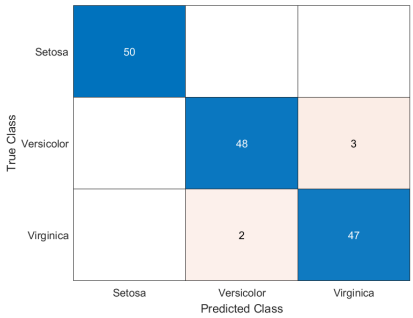

FIGURE 5: Bayes LOO Confusion Matrix

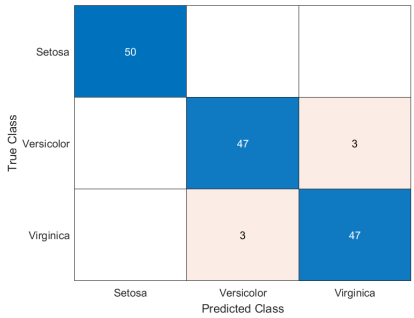

FIGURE 6: Naive Bayes LOO Confusion Matrix

In the confusion matrices, we see perfect classification for Setosa. If we reference the histograms of the features we are using, we can clearly see why. In figures 3 and 4 we see

that the features for Setosa are completely separated from the other two irises. We see some misclassification for Versicolour and Virginica because there is some overlap in the features distribution. When comparing the results between the Bayes and Naive classifier, we see that they perform similarly. The Naive classifier only misclassifies one more sample than the Bayes classifier.

Breast Cancer

Data

The data for this section was taken from the Breast Cancer Wisconsin (Diagnostic) data set. It includes 357 benign samples and 212 malignant samples totaling 569 samples, each with 30 features. This data set can be found at `http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/`.

Procedure

Once the data set was downloaded into Matlab, our first task was to split the data into training and test data sets. From the entire data set, 80% was set aside for training data and the other 20% was stored to be used for testing after the Bayes Classifier was trained. The steps we took to do this are outlined in Listing 3.

Listing 3: Breast Cancer Initial Processing

```
% load data set
wdbc = importdata("wdbc.data");

%separate benign and malignant
pos_data = wdbc.data((wdbc.textdata(:,2))=="M",:);
neg_data = wdbc.data((wdbc.textdata(:,2))=="B",:);
pos80 = round(length(pos_data)*0.8);

neg80 = round(length(neg_data)*0.8);

pos_Train = (pos_data(1:pos80,:));
neg_Train = (neg_data(1:neg80,:));

pos_Test = (pos_data(pos80:length(pos_data),:));
neg_Test = (neg_data(neg80:length(neg_data),:));

train_data = [pos_Train; neg_Train];
test_data = [pos_Test; neg_Test];
```

The next step in creating the Bayes classifier is finding the features that will be used. To do this we iterated through all 30 features and created histograms comparing the benign distributions with the malignant distributions. We visually checked the histograms for which features were the most invariant and robust. We also retroactively calculated the errors between the histograms to ensure we were using the most accurate and separate data. This process can be seen in Listing 4. From the 30 features, we chose Feature 23 and Feature 28.

Next, a classifier was created based on the Bayes Decision Rule. To make the process simpler we created multiple functions that can be seen implemented in Listing 5. The first one was tasked with determining the different characteristic values of our data. We input our feature data into the function and were returned means, covariances, priors, as well as

Listing 4: Breast Cancer Histograms

```
%create histograms to compare positive and negative data to find good
%discriminant features
feature_overlap = size(neg_data,2);
for feature=1:30
    figure;
    hold on;
    histogram(pos_data(:,feature));
    histogram(neg_data(:,feature));
    xlabel("Value");
    ylabel("Frequency");
    %figtitle = sprintf("Feature %d");
    title("Feature ",feature);
    hold off;
    feature_overlap(feature) = hist_overlap(pos_data(:,feature), neg_data(:,feature));

end
```

a whole data set and classification table. The next function created was to develop our dichotomizer. We input the data set and the characteristic values of our data. The function calculates the terms needed for a Bayes classifier and creates a prediction list depending on the classifier.

Listing 5: Full Bayes

```
%make test set feature vectors
pos_feat_test = [pos_Test(:,feature_1) pos_Test(:,feature_2)];
neg_feat_test = [neg_Test(:,feature_1) neg_Test(:,feature_2)];

%get dataset and augmented matrix from test features
[X_test, Y_test, ~] = feat_details(pos_feat_test,neg_feat_test);

%classify test set
[predict,~] = dichotomizer(X_test, Y_test, mu_pos, cov_pos, prior_pos, mu_neg, cov_neg,
↪  prior_neg);
```

Next, we tested the accuracy of our dichotomizer, using both confusion maps and ROC curves. As seen in Listing 6 we were able to get the confusion map by using the built-in `confusionmat()` function from Matlab. To get the ROC curve, we swept the priors of the classes and therefore changing our dichotomizer. And for every prior pair we got a new confusion map and calculated the values that would be plotted on the ROC.

Another task we had to complete was the Naive Bayes classifier. The Naive Bayes assumes independence between features making it easier to compute. The Naive Bayes code can be found in Listing 7.

Listing 6: Confusion Map and ROC Code

```
%showing results through confusion matrix (TP, TN, FP, FN)
C = confusionmat(Y_test, predict);
confusionchart(C,["Benign","Malignant"]);

%ROC curve
step = 0.05;
prior_list = 0:step:1;
tp_list = length(prior_list);
fp_list = length(prior_list);

for i = 1:length(prior_list)
%gather predictions for different priors
[predict,~] = dichotomizer(X_test, Y_test, mu_pos, cov_pos, prior_list(i), mu_neg,
↪  cov_neg, 1-prior_list(i));

%collect TP and FP info for each set of priors
C = confusionmat(Y_test, predict);
% tp_list(i) = C(2,2)/sum(C(:));
% fp_list(i) = C(1,2)/sum(C(:));
tp_list(i) = C(2,2)/(C(2, 2)+ C(2, 1));
fp_list(i) = C(1,2)/(C(1, 2)+ C(1, 1));
end

plot(fp_list,tp_list);
xlabel("False Alarms (FP)");
ylabel("Correct Detections (TP)");
```

Listing 7: Naive Bayes Code

```
%Naïve Bayes classifier
cov_pos_naive = [cov(pos_feat(:,1));cov(pos_feat(:,2))];
cov_neg_naive = [cov(neg_feat(:,1));cov(neg_feat(:,2))];

[predict,~] = naive_bayes(X_test, Y_test, mu_pos, cov_pos_naive, prior_pos, mu_neg,
↪  cov_neg_naive, prior_neg);
```

Results

From the 30 different features we could use from the breast cancer data set, we chose to work with the features 23, 28, 21, and 8 because they had the most separability between distributions. In other words, they had the least amount of area overlap between the two classes. The histograms for these features can be seen in figures 7 to 10.

With the features selected, we can move on to developing and training a classifier. Once both classifiers were trained, we used the test data that we had set aside from earlier. To test the accuracy of the classifiers, we created ROC curves. To do this, we swept the priors of the classes and created the confusion matrix at every instance. This can be seen in Figures

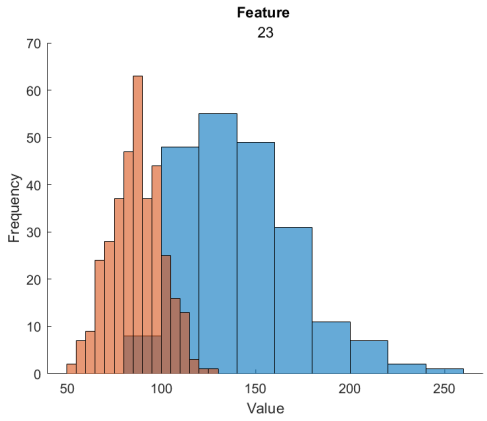

FIGURE 7: Breast Cancer Feature 23

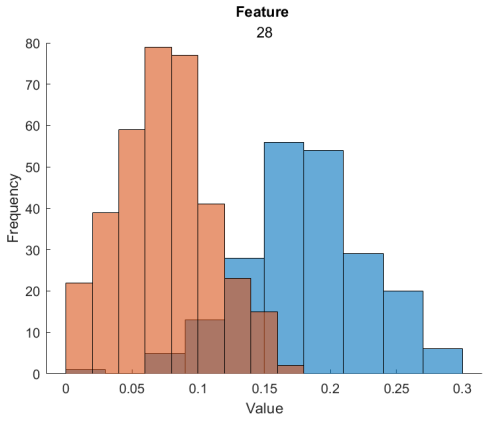

FIGURE 8: Breast Cancer Feature 28

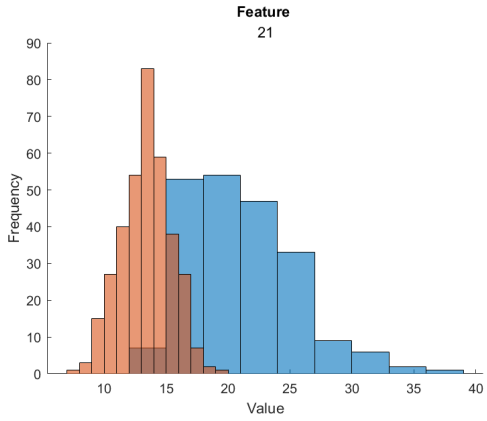

FIGURE 9: Breast Cancer Feature 21

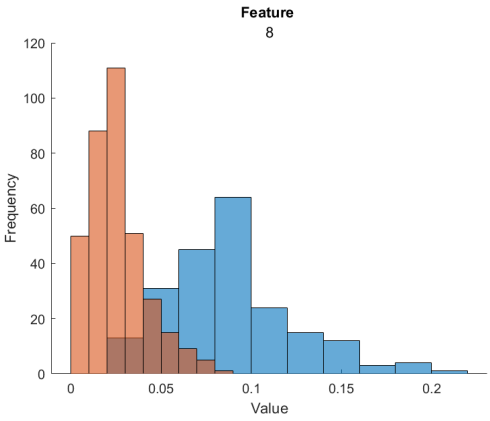

FIGURE 10: Breast Cancer Feature 8

11 to 14.

The results can be found in Table 2.

TABLE 2: Breast Cancer Accuracy Results

Features	Training (%)	Test (%)	Naive Test (%)
28v8	91	90	95
28v23	94	95	88
28v23v8	93	95	91

We can see from the table and the figures that using features 28 and 23 give us the best results. This makes sense because these are the two features with the least overlap area. When comparing the Bayes and the Naive classifier, we see a drop in accuracy. When observing the difference in performance of Bayes and Naive classifiers for features 28 and 8 this is not the case. The Naive classifier in this case sees an increase in accuracy. It is also important to point out that in the table, when increasing the features used to classify data, we do not see an increase upon our best performance. Our Best classifier turns out to be using just soley the two features 28 and 23.

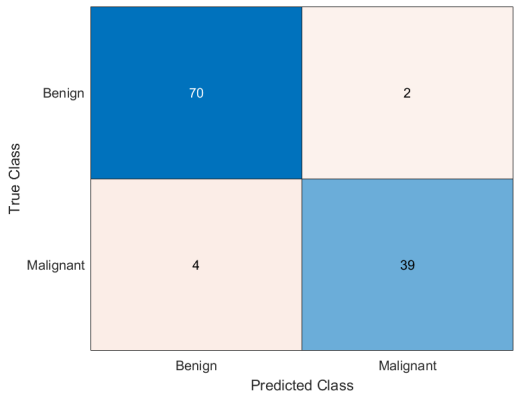

FIGURE 11: Bayes Confusion Matrix: Feature 28 vs 23

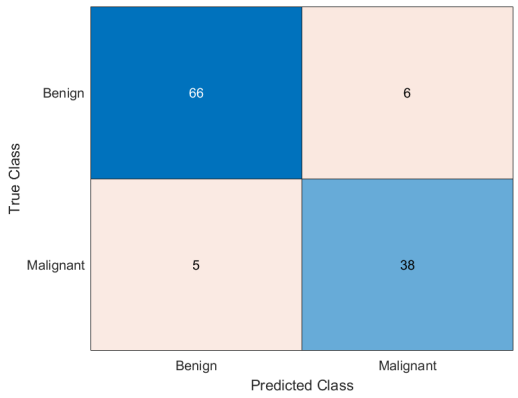

FIGURE 12: Bayes Confusion Matrix: Feature 28 vs 8

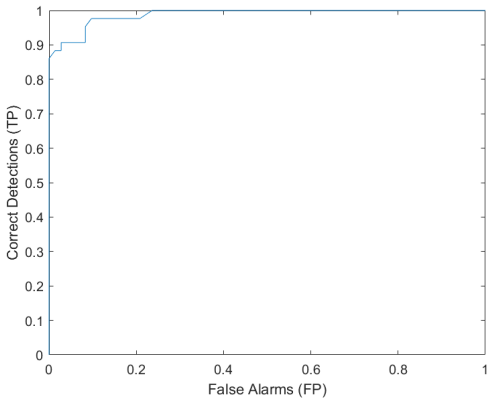

FIGURE 13: Bayes ROC: Feature 28 vs 23

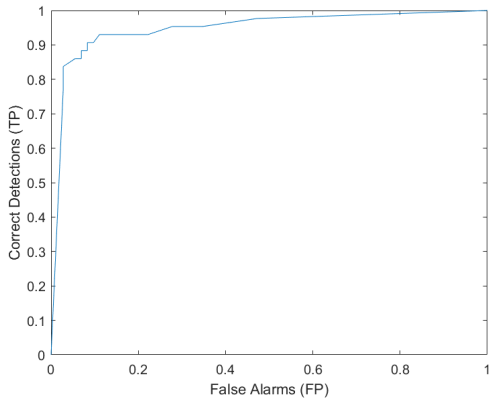

FIGURE 14: Bayes ROC: Feature 28 vs 8

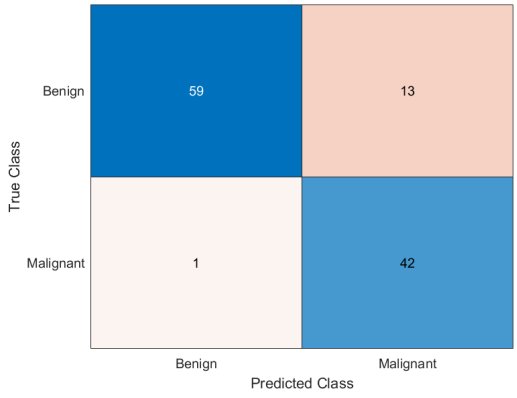

FIGURE 15: Naive Confusion Matrix: Feature 28 vs 23

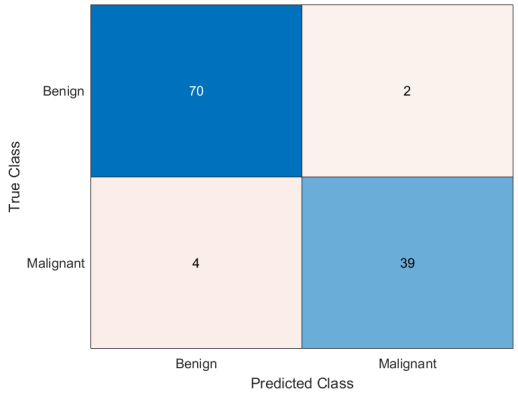

FIGURE 16: Naive Confusion Matrix: Feature 28 vs 8

Conclusion

Nathan Jaggers

In this project we explored the different ways of training and testing Bayes and Naive Bayes classifiers for different sized data sets. For the smaller data set we empolyed the Leave-One-

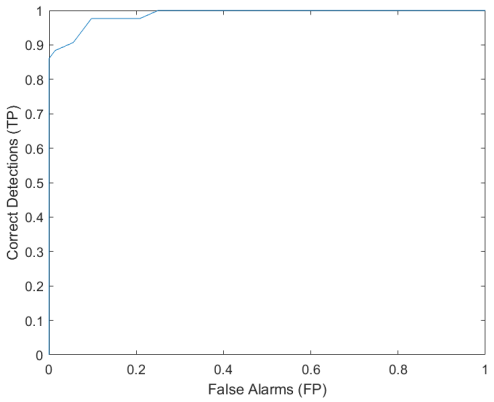

FIGURE 17: Naive ROC: Feature 28 vs 23

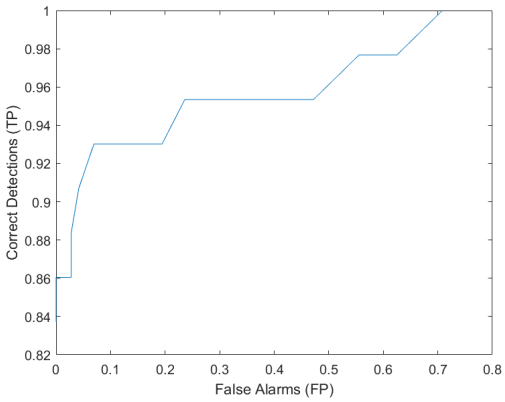

FIGURE 18: Naive ROC: Feature 28 vs 8

Out procedure so we could still have a significant testing and training pool of data. For the larger data set we used a more traditional 80% and 20% split of the data for training and testing respectively. This project was a great way to explore these concepts and the effectiveness of Naive Bayes in different situations.

Nicholas Brunet

In this lab, we learned several extensions to the simple Bayes classifier explored in the previous lab. For the Iris dataset, we worked with a 3-class classifier with the leave-one-out cross-validation method. We explored the effects of varying the number of parameters on the breast cancer dataset. We also implemented the Naive Bayes classifier on both datasets. My primary work was implementing the Bayes classifier with leave-one-out for the Iris dataset.

Jordan Perlas

During this lab, I learned a lot more about the different variations of the Bayes Classifier and how they work. The two data sets offer a great insight into all the different ways pattern recognition can be applied in the practical world. The breast cancer data especially showed me the impact of misclassification in a real-world setting. I enjoyed using the confusion maps and seeing how they can be used to create the ROC curve.

Complete Code Listings

Part A

```
%% Project 5
%
% EE 516 - Pattern Recognition
% Spring 2023
%
% Group 4: Nathan Jaggers, Nicholas Brunet, Jordan Rubio Perlas
%
% Description: See coresponding document <Can add description later>
%% Part 1
close all;
clear;
clc;

%%
% load data set
load("fisheriris.mat")

%%
% Features:
% 1. sepal length
% 2. sepal width
% 3. petal length
% 4. petal width

setosa_X = meas(1:50,:); % w1
versicolor_X = meas(51:100,:); % w2
virginica_X = meas(101:150,:); % w3

setosa_Y = ones(50, 1);
versicolor_Y = ones(50, 1) * 2;
virginica_Y = ones(50, 1) * 3;
Y = [setosa_Y; versicolor_Y; virginica_Y];

% BAD
hold on
histogram(setosa_X(:,1), 'FaceColor', 'r', 'BinWidth', .2);
histogram(versicolor_X(:,1), 'FaceColor', 'g', 'BinWidth', .2);
histogram(virginica_X(:,1), 'FaceColor', 'b', 'BinWidth', .2);
xlabel("Value");
ylabel("Frequency");
title("Sepal Length");
legend("Setosa", "Versicolor", "Virginica");
hold off

% BAD
figure;
hold on
histogram(setosa_X(:,2), 'FaceColor', 'r', 'BinWidth', .2);
```

```
histogram(versicolor_X(:,2), 'FaceColor', 'g', 'BinWidth', .2);
histogram(virginica_X(:,2), 'FaceColor', 'b', 'BinWidth', .2);
xlabel("Value");
ylabel("Frequency");
title("Sepal Width");
legend("Setosa", "Versicolor", "Virginica");
hold off

% GOOD
figure;
hold on
histogram(setosa_X(:,3), 'FaceColor', 'r', 'BinWidth', .2);
histogram(versicolor_X(:,3), 'FaceColor', 'g', 'BinWidth', .2);
histogram(virginica_X(:,3), 'FaceColor', 'b', 'BinWidth', .2);
xlabel("Value");
ylabel("Frequency");
title("Petal Length");
legend("Setosa", "Versicolor", "Virginica");
hold off

% GOOD
figure;
hold on
histogram(setosa_X(:,4), 'FaceColor', 'r', 'BinWidth', .2);
histogram(versicolor_X(:,4), 'FaceColor', 'g', 'BinWidth', .2);
histogram(virginica_X(:,4), 'FaceColor', 'b', 'BinWidth', .2);
xlabel("Value");
ylabel("Frequency");
title("Petal Width");
legend("Setosa", "Versicolor", "Virginica");
hold off

% Prior probabilities
w1_P = size(setosa_X,1)/size(meas,1); % 1/3 % each have 50 samples
w2_P = size(versicolor_X,1)/size(meas,1); % 1/3 % total samples is 150
w3_P = size(virginica_X,1)/size(meas,1); % 1/3 % 1/3 = 50/150

%% Data cleaning

% Remove sepal features
setosa_X = setosa_X(:,3:4);
versicolor_X = versicolor_X(:,3:4);
virginica_X = virginica_X(:,3:4);


%% Analysis

% PROCESS
% for s in samples
%   leave s out
%   train with dataset
%   test on s => record score
% average scores

```

```
103
104
105 C = loo_bayes(Y, setosa_X, w1_P, versicolor_X, w2_P, virginica_X, w3_P);
106 %confusionchart(C);
107 confusionchart(C,["Setosa", "Versicolor", "Virginica",]);
108
109 % xvalues = {'Setosa', 'Versicolor', 'Virginica'};
110 % yvalues = {'Setosa', 'Versicolor', 'Virginica'};
111 % h = heatmap(xvalues, yvalues, conf_mat);
112 % h.XLabel = 'Actual';
113 % h.YLabel = 'Predicted';
114
115 %%
116 C = naive_loo_bayes(Y, setosa_X, w1_P, versicolor_X, w2_P, virginica_X, w3_P);
117 %confusionchart(C);
118 confusionchart(C,["Setosa", "Versicolor", "Virginica",]);
119
120
121 %%
122 function conf_mat = loo_bayes(Y, setosa_X, w1_P, versicolor_X, w2_P, virginica_X, w3_P)
123
124     correct = 0;
125     conf_mat = zeros(3, 3);  % confusion matrix
126
127     for i = 1:150
128
129         % create train/test split
130         setosa_X_train = setosa_X;
131         versicolor_X_train = versicolor_X;
132         virginica_X_train = virginica_X;
133         if i <= 50
134             x = setosa_X(i,:);
135             setosa_X_train(i,:) = [];
136         elseif i <= 100
137             x = versicolor_X(i-50,:);
138             versicolor_X_train(i-50,:) = [];
139         else
140             x = virginica_X(i-100,:);
141             virginica_X_train(i-100,:) = [];
142         end
143         y = Y(i);
144
145         x = x'; % make x a column vector
146
147         w1_mean = mean(setosa_X_train)';
148         w2_mean = mean(versicolor_X_train)';
149         w3_mean = mean(virginica_X_train)';
150
151         w1_cov = cov(setosa_X_train);
152         w2_cov = cov(versicolor_X_train);
153         w3_cov = cov(virginica_X_train);
154
155         % Test
156
```

```
        g1 = g(x, w1_mean, w1_cov, w1_P);
        g2 = g(x, w2_mean, w2_cov, w2_P);
        g3 = g(x, w3_mean, w3_cov, w3_P);

        if g1 >= g2 && g1 >= g3
            result = 1;
        elseif g2 >= g1 && g2 >= g3
            result = 2;
        else
            result = 3;
        end

        conf_mat(result, y) = conf_mat(result, y) + 1;
        if(result == y)
            correct = correct + 1;
        end



    end

    fprintf("Accuracy: %.3f\n", correct / 150);

end

function conf_mat = naive_loo_bayes(Y, setosa_X, w1_P, versicolor_X, w2_P, virginica_X,
↪  w3_P)

    correct = 0;
    conf_mat = zeros(3, 3);  % confusion matrix

    for i = 1:150

        % create train/test split
        setosa_X_train = setosa_X;
        versicolor_X_train = versicolor_X;
        virginica_X_train = virginica_X;
        if i <= 50
            x = setosa_X(i,:);
            setosa_X_train(i,:) = [];
        elseif i <= 100
            x = versicolor_X(i-50,:);
            versicolor_X_train(i-50,:) = [];
        else
            x = virginica_X(i-100,:);
            virginica_X_train(i-100,:) = [];
        end
        y = Y(i);

        x = x'; % make x a column vector

        %Training
        w1_mean = mean(setosa_X_train)';
        w2_mean = mean(versicolor_X_train)';
```

```
        w3_mean = mean(virginica_X_train)';

        w1_cov = zeros(size(setosa_X_train,2),1);
        w2_cov = zeros(size(versicolor_X_train,2),1);
        w3_cov = zeros(size(virginica_X_train,2),1);

        for cols = 1:size(setosa_X_train,2)
            w1_cov(cols,1) = cov(setosa_X_train(:,cols));
            w2_cov(cols,1) = cov(versicolor_X_train(:,cols));
            w3_cov(cols,1) = cov(virginica_X_train(:,cols));
        end

        % Test

        %multiplication pdfs of of first class
        pdf_per_class =
    ↪  (1./(sqrt(2.*pi().*w1_cov))).*exp(-0.5.*((x-w1_mean).^2)./w1_cov);
        pdf_pi = prod(pdf_per_class);
        %discriminant of first class
        g1 = log(pdf_pi)*w1_P;

        %multiplication pdfs of of second class
        pdf_per_class =
    ↪  (1./(sqrt(2.*pi().*w2_cov))).*exp(-0.5.*((x-w2_mean).^2)./w2_cov);
        pdf_pi = prod(pdf_per_class);
        %discriminant of second class
        g2 = log(pdf_pi)*w2_P;

        %multiplication pdfs of of third class
        pdf_per_class =
    ↪  (1./(sqrt(2.*pi().*w3_cov))).*exp(-0.5.*((x-w3_mean).^2)./w3_cov);
        pdf_pi = prod(pdf_per_class);
        %discriminant of second class
        g3 = log(pdf_pi)*w3_P;

        if g1 >= g2 && g1 >= g3
            result = 1;
        elseif g2 >= g1 && g2 >= g3
            result = 2;
        else
            result = 3;
        end

        conf_mat(result, y) = conf_mat(result, y) + 1;
        if(result == y)
            correct = correct + 1;
        end

    end

    fprintf("Accuracy: %.3f\n", correct / 150);

end
```

```
% Calculate parameters once in separate function if this takes too long
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

```

Part B

```
    %% Project 5
%
% EE 516 - Pattern Recognition
% Spring 2023
%
% Group 4: Nathan Jaggers, Nicholas Brunet, Jordan Rubio Perlas
%
% Description: See coresponding document <Can add description later>
%% Part 2
close all;
clear;
clc;

%%
% load data set
wdbc = importdata("wdbc.data");

%separate benign and malignant
pos_data = wdbc.data((wdbc.textdata(:,2))=="M",:);
neg_data = wdbc.data((wdbc.textdata(:,2))=="B",:);

%%
%create histograms to compare positive and negative data to find good
%discriminant features
feature_overlap = size(neg_data,2);
for feature=1:30
    figure;
    hold on;
    histogram(pos_data(:,feature));
    histogram(neg_data(:,feature));
    xlabel("Value");
    ylabel("Frequency");
    %figtitle = sprintf("Feature %d");
    title("Feature ",feature);
    hold off;
    feature_overlap(feature) = hist_overlap(pos_data(:,feature), neg_data(:,feature));

end

```

```
%%
%use overlap to  find best features
best_features = zeros(5,2);
min_Fill = max(feature_overlap);
tempMat = feature_overlap;
for i=1:size(best_features,1);
[best_features(i,1), best_features(i,2)] = min(tempMat);
tempMat(best_features(i,2)) = min_Fill;
fprintf("Minimums at %d: %f\n",best_features(i,2),best_features(i,1));
end

%%
% create data set for training and test

pos80 = round(length(pos_data)*0.8);
neg80 = round(length(neg_data)*0.8);

pos_Train = (pos_data(1:pos80,:));
neg_Train = (neg_data(1:neg80,:));

pos_Test = (pos_data(pos80:length(pos_data),:));
neg_Test = (neg_data(neg80:length(neg_data),:));

train_data = [pos_Train; neg_Train];
test_data = [pos_Test; neg_Test];

%getting prior probabilities from ratios in training data
% pos_prior = pos80/(pos80+neg80);
% neg_prior = neg80/(pos80+neg80);

%%
%ones that we like: 28, 23, 21ish, 8, 3

%trying to use features 8 and 28
%make bayesian characteristics mu and sigma
feature_1 = 23;
feature_2 = 28;
feature_3 = 8;
% pos_feat = [pos_Train(:,feature_1) pos_Train(:,feature_2)];
% neg_feat = [neg_Train(:,feature_1) neg_Train(:,feature_2)];

pos_feat = [pos_Train(:,feature_1) pos_Train(:,feature_2) pos_Train(:,feature_3)];
neg_feat = [neg_Train(:,feature_1) neg_Train(:,feature_2) neg_Train(:,feature_3)];

%%
%training and set up for dichotomizer
[X, Y, mu_pos, cov_pos, prior_pos, mu_neg, cov_neg, prior_neg] =
↪  feat_details(pos_feat,neg_feat);

%using dichotomizer
[predict,~] = dichotomizer(X, Y, mu_pos, cov_pos, prior_pos, mu_neg, cov_neg, prior_neg);

%showing results through confusion matrix
C = confusionmat(Y, predict);
```

```
confusionchart(C,["Benign","Malignant"]);

%%
%using previous training on test set

%make test set feature vectors
% pos_feat_test = [pos_Test(:,feature_1) pos_Test(:,feature_2)];
% neg_feat_test = [neg_Test(:,feature_1) neg_Test(:,feature_2)];

pos_feat_test = [pos_Test(:,feature_1) pos_Test(:,feature_2) pos_Test(:,feature_3)];
neg_feat_test = [neg_Test(:,feature_1) neg_Test(:,feature_2) neg_Test(:,feature_3)];

%get dataset and augmented matrix from test features
[X_test, Y_test, ~] = feat_details(pos_feat_test,neg_feat_test);

%classify test set
[predict,~] = dichotomizer(X_test, Y_test, mu_pos, cov_pos, prior_pos, mu_neg, cov_neg,
↪  prior_neg);

%showing results through confusion matrix (TP, TN, FP, FN)
C = confusionmat(Y_test, predict);
confusionchart(C,["Benign","Malignant"]);

%%
%ROC curve
step = 0.05;
prior_list = 0:step:1;
tp_list = length(prior_list);
fp_list = length(prior_list);

for i = 1:length(prior_list)
%gather predictions for different priors
[predict,~] = dichotomizer(X_test, Y_test, mu_pos, cov_pos, prior_list(i), mu_neg,
↪  cov_neg, 1-prior_list(i));

%collect TP and FP info for each set of priors
C = confusionmat(Y_test, predict);
tp_list(i) = C(2,2)/(C(2, 2)+ C(2, 1));
fp_list(i) = C(1,2)/(C(1, 2)+ C(1, 1));
end

plot(fp_list,tp_list);
xlabel("False Alarms (FP)");
ylabel("Correct Detections (TP)");

%%
%Naïve Bayes classifier
% cov_pos_naive = [cov(pos_feat(:,1));cov(pos_feat(:,2))];
% cov_neg_naive = [cov(neg_feat(:,1));cov(neg_feat(:,2))];

cov_pos_naive = [cov(pos_feat(:,1));cov(pos_feat(:,2));cov(pos_feat(:,3))];
cov_neg_naive = [cov(neg_feat(:,1));cov(neg_feat(:,2));cov(neg_feat(:,3))];

```

```
[predict,~] = naive_bayes(X_test, Y_test, mu_pos, cov_pos_naive, prior_pos, mu_neg, cov_neg_naive, prior_neg);

%showing results through confusion matrix (TP, TN, FP, FN)
C = confusionmat(Y_test, predict);
confusionchart(C,["Benign","Malignant"]);

%%
%ROC curve
step = 0.05;
prior_list = 0:step:1;
tp_list = length(prior_list);
fp_list = length(prior_list);

for i = 1:length(prior_list)
%gather predictions for different priors
[predict,~] = naive_bayes(X_test, Y_test, mu_pos, cov_pos_naive, prior_list(i), mu_neg, cov_neg_naive, 1-prior_list(i));

%collect TP and FP info for each set of priors
C = confusionmat(Y_test, predict);
tp_list(i) = C(2,2)/(C(2, 2)+ C(2, 1));
fp_list(i) = C(1,2)/(C(1, 2)+ C(1, 1));
end

plot(fp_list,tp_list);
xlabel("False Alarms (FP)");
ylabel("Correct Detections (TP)");

%%
%set up for dichotomizer
function [dataset, classification, w1_mean, w1_cov, w1_Prior, w2_mean, w2_cov, w2_Prior] = feat_details(w1_features, w2_features)
    %calculate mean and cov to train dichotomizer
    w1_mean = mean(w1_features)';
    w2_mean = mean(w2_features)';

    w1_cov = cov(w1_features);
    w2_cov = cov(w2_features);

    %priors section
    w1_samples = size(w1_features,1);
    w2_samples = size(w2_features,1);
    w1_Prior = w1_samples/(w1_samples+w2_samples);
    w2_Prior = w2_samples/(w1_samples+w2_samples);

    %create input matrix and augmented matrix
    dataset = [w1_features; w2_features]; %input
    classification = [ones(size(w1_features, 1), 1); zeros(size(w2_features, 1), 1)]; %augmented (Y)
end

%general dichotomizer
```

```
function [prediction, accuracy] = dichotomizer(dataset, classification, w1_mean, w1_cov,
    w1_Prior, w2_mean, w2_cov, w2_Prior)
    %dataset - input data to be classified
    %classification - true classification of samples
    %w1_mean  - mean for class 1
    %w1_cov   - covariance for class 2
    %w1_Prior - prior for class 1
    %w2_mean  - mean for class 2
    %w2_cov   - covariance for class 2
    %w2_Prior - prior for class 2

    %create class prediction matrix
    prediction = [zeros(size(classification, 1), 1)];

    %initialize correct counter
    correct = 0;

    for i = 1:size(dataset, 1)
        x = dataset(i,:)';
        y = classification(i);
        g1_result = g(x, w1_mean, w1_cov, w1_Prior);
        g2_result = g(x, w2_mean, w2_cov, w2_Prior);
        prediction(i) = g1_result - g2_result > 0;
        correct = correct + (prediction(i) == y);
    end
    accuracy = correct / size(dataset, 1);
    fprintf("Accuracy: %.2f\n", accuracy);
end

% Calculate parameters once in separate function if this takes too long
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

%Naïve Bayes classifier
function [prediction, accuracy] = naive_bayes(dataset, classification, w1_mean, w1_cov,
    w1_Prior, w2_mean, w2_cov, w2_Prior)
    %dataset - input data to be classified
    %classification - true classification of samples
    %w1_mean  - mean for class 1
    %w1_cov   - naive covariance for class 2 (one variance for each feature; column
    vector)
    %w1_Prior - prior for class 1
    %w2_mean  - mean for class 2
    %w2_cov   - covariance for class 2 (one variance for each feature; column vector)
    %w2_Prior - prior for class 2

    %create class prediction matrix
    prediction = [zeros(size(classification, 1), 1)];

```

```
244     %initialize correct counter
245     correct = 0;
246 
247     for i = 1:size(dataset, 1)
248         x = dataset(i,:)';
249         y = classification(i);
250         %multiplication pdfs of of first class
251         pdf_per_class =
    ↪  (1./(sqrt(2.*pi().*w1_cov))).*exp(-0.5.*((x-w1_mean).^2)./w1_cov);
252         pdf_pi = prod(pdf_per_class);
253         %discriminant of first class
254         g1_result = log(pdf_pi)*w1_Prior;
255 
256         %multiplication pdfs of of second class
257         pdf_per_class =
    ↪  (1./(sqrt(2.*pi().*w2_cov))).*exp(-0.5.*((x-w2_mean).^2)./w2_cov);
258         pdf_pi = prod(pdf_per_class);
259         %discriminant of second class
260         g2_result = log(pdf_pi)*w2_Prior;
261 
262         prediction(i) = g1_result - g2_result > 0;
263         correct = correct + (prediction(i) == y);
264     end
265     accuracy = correct / size(dataset, 1);
266     fprintf("Accuracy: %.2f\n", accuracy);
267 end
268 
269 %% Compare Histograms
270 % edited from Dylan Baxter's and Nicholas Brunet's code
271 function [overlap] = hist_overlap(feature1, feature2)
272     numBins = 50;
273     % Extract histogram Information
274     max_val = max([feature1;feature2], [], "all");
275     min_val = min([feature1;feature2], [], "all");
276     bins = linspace(min_val, max_val, numBins+1);
277     hist1 = histcounts(feature1, bins);
278     hist2 = histcounts(feature2, bins);
279 
280     % Iterate over histograms and calculate overlap
281     overlap = 0;
282     for m = 1:length(hist1)
283         overlap = overlap + min(hist1(m), hist2(m));
284     end
285     overlap = overlap / sum(hist1, "all");
286 end
287 
```