Week 1
CS 273a - Introduction to Machine Learning (Winter '15)[*]
Prof. Alex Ihler

Varad Meru
Student # 26648958
Due Date: 01/13/2015

Homework 1[†‡]

Problem 1: Matlab & Data Exploration

In this problem, I looked at some constructs provided by Matlab to perform basic statistics and get visualizations of an example data set. I used the downloaded and loaded the *Fisher iris* dataset provided using code given in Listing 1.

Listing 1: Load iris data set and create necessary variables

```
% Fetching the dataset and separating it into X and Y.
iris=load('data/iris.txt');     % load the text file
y = iris(:,end);              % target value is last column
X = iris(:,1:end-1);          % features are other columns
features = char('Sepal length','Sepal width','Petal length','Petal ...
    width','Species');
features_short = char('SL','SW','PL','PW','SP');
disp(whos);

% Changed the shuffleData function to take a seed from the user. If not
% provided, then it generates it randomly.
[X, y] = shuffleData(X,y,s);

% Used in some sub-problems
X = X(:,1:2);
[Xtr,Xte,Ytr,Yte] = splitData(X,y, .75);
```

I have listed the approaches I used for solving the sub-parts of the problem. This question had six sub-parts.

(a) The size of the data matrix X is computed using the `size()` function provided by Matlab.

Listing 2: Displaying the size of input matrix X. output written in the next line of the command

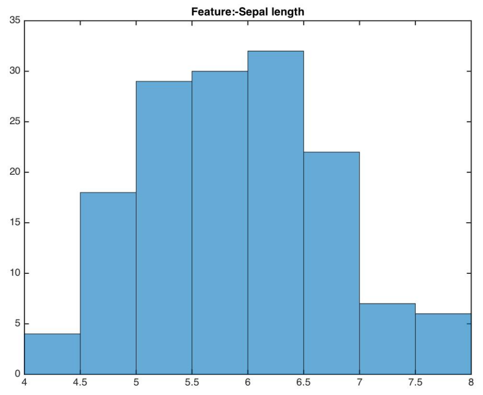
```
% Part (a) - Use size(X,2) to get the number of features, and size(X,1) ...
    to get the number of data points.
disp(size(X, 1));
   148
disp(size(X, 2));
     2
```

(b) For plotting the histograms for each feature, I used the `histogram()` function provided by Matlab as shown in Listing 3. You can see the histograms in Figure 1. The saves function is used to store the produced histograms as image files. Figure 2 shows a unified view of all the features of the dataset overlapped.

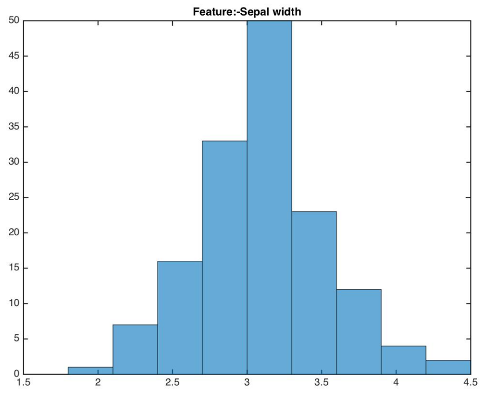
Listing 3: Matlab code for creating histograms.

```
%% Part (b)
% For each feature, plot a histogram ("hist") of the data values
for f=1:size(X, 2);
    h=figure;
    %subplot(1,4,f)
```

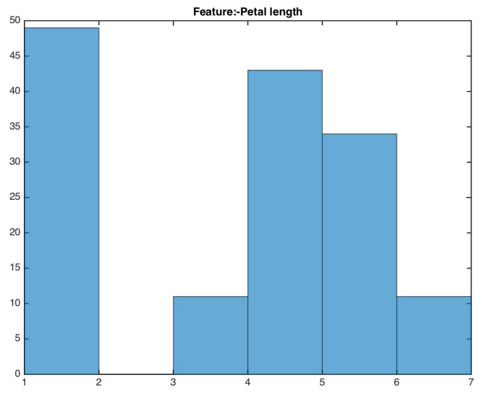
[*]Website: http://sli.ics.uci.edu/Classes/2015W-273a
[†]Questions available at http://sli.ics.uci.edu/Classes/2015W-273a?action=download&upname=HW1.pdf
[‡]All the figures and listing numbers are auto-refered.



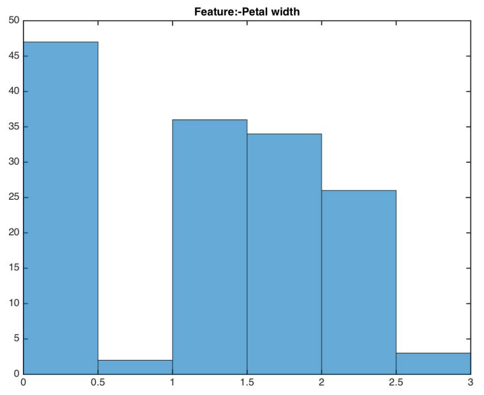
(a) Sepal Length



(b) Sepal Width



(c) Petal Length



(d) Petal Width

Figure 1: Histogram of features in the ‘Fisher iris’ data set

```
    h1=histogram(X(:,f)); % histogram is preferred over hist - matlab ...
        documentation
    title(strcat('Feature:- ',features(f,:)))
    %hold on;
    saveas(h,strcat('histogram',num2str(f)),'jpg');
end;
%hold off;
```

Listing 4: Overlapping histograms to show a unified view of ‘Fisher iris’ dataset.

```
% All in one
h=figure;
for f=1:size(X, 2);
    histogram(X(:,f));
    hold on;
end;
hold off;
saveas(h,'histogram5.jpg','jpg');
```

(c) The mean is computed using the `mean()` method provided by matlab (Listing 6)

Listing 5: Mean of features of ‘Fisher iris’ data set.

```
%% Part (c)
% Compute the mean of the data points for each feature (mean)
disp('Mean');
```

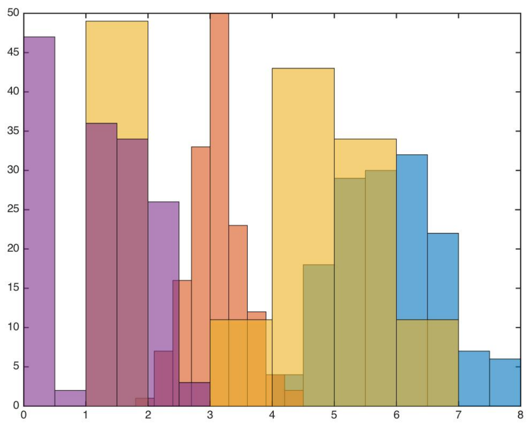


Figure 2: Histogram of features in the ‘Fisher iris’ data set.

```
meanX = mean(X);
for f=1:size(X, 2);
    disp(strcat({'Feature:- '}, features(f,:),{': '} , num2str(meanX(:,f))));
end;

%Output
Mean
    'Feature:- Sepal length: 5.9001'
    'Feature:- Sepal width: 3.0989'
    'Feature:- Petal length: 3.8196'
    'Feature:- Petal width: 1.2526'
```

(d) The mean is computed using the `mean()` method provided by matlab (Listing 6)

Listing 6: Mean of features of ‘Fisher iris’ data set.

```
%% Part (d)
% Compute the variance and standard deviation of the data points for each ...
    feature
disp('Standard Deviations')
stdX = std(X);
for f=1:size(X, 2);
    disp(strcat({'Feature:- '}, features(f,:),{': '} , num2str(stdX(:,f))));
end;

disp('Variances')
varX = var(X);
for f=1:size(X, 2);
    disp(strcat({'Feature:- '}, features(f,:),{': '} , num2str(varX(:,f))));
end;

%Output
Standard Deviations
    'Feature:- Sepal length: 0.83623'
    'Feature:- Sepal width: 0.43777'
    'Feature:- Petal length: 1.76'
    'Feature:- Petal width: 0.76135'

Variances
    'Feature:- Sepal length: 0.69928'
    'Feature:- Sepal width: 0.19165'
    'Feature:- Petal length: 3.0976'
    'Feature:- Petal width: 0.57965'
```

(e) I normalized the dataset using the `bsxfun()`, which applies element-by-element binary operations on the specified function. The functions I used were `@minus` and `@rdivide`. The code described in Listing 9 shows the use. One more approach to achieve the same is seen in Listing 8.

Listing 7: Normalizing the data set

```
% bsxfun: applies the element-by-element binary operation specified by ...
    the function handle fun to arrays A and B, with singleton expansion ...
    enabled.
normX = bsxfun(rdivide, bsxfun(minus, X, mean(X)), std(X));
```

Listing 8: Normalizing the data set using `repmat()`

```
%One More way to do this (more understandable way) -
meanXMat = repmat(meanX, size(X,1),1); stdXMat = repmat(stdX, size(X,1),1);
norX1 = X - meanXMat; normX = norX1 ./ stdXMat;
```

(f) The plot for pairs of features (1,2), (1,3), and (1,4) is created using a for-loop and the `scatter()`. The colors are set by passing the class values to the scatter function. The code that performs the operation can be seen in Listing 9 and the resulting set of plots generated can be seen in Figure 3

Listing 9: Generating scatter plots for feature pairs - (1,2), (1,3), and (1,4).

```
h=figure;
i = 1;

for f=2:size(X, 2);
    subplot(1,3,i);
    h1 = scatter(X(:,1), X(:,f), 50, y, 'filled');
    i = i+1;
    title(strcat({'X: '},features_short(1,:), {',Y:'}, features_short(f,:)))
    hold on;
end;

hold off;
saveas(h,'plots.jpg','jpg');
```

Problem 2: kNN predictions

In this problem, I looked at knnClassify class and the features provided by it to classify data in the Iris dataset. The data is shuffle and split into training and test subsets. The question had 2 sub-questions:

(a) I modified the code snippet provided in the homework description and created the classifier. The code after modification is given in Listing 10. The classes for various values of k can be seen in Figure 4. The `Randstream()` function helps in giving a seed to the randomizer and helps in reproducing the results. The `shuffleData()` was modified for accommodating the same with an option of being empty, in which the results would be generated randomly.

Listing 10: Computes the classes using kNNClassify class and generates the scatter plot.

```
i = 1;
for k=[1, 5, 10, 50];
    h=figure;
    %subplot(2,2,i);
    i = i+1;
    knn = knnClassify( Xtr, Ytr, k );
    YteHat = predict( knn, Xte );
    plotClassify2D( knn, Xtr, Ytr );
    title(strcat({'K= '},num2str(k)))
```

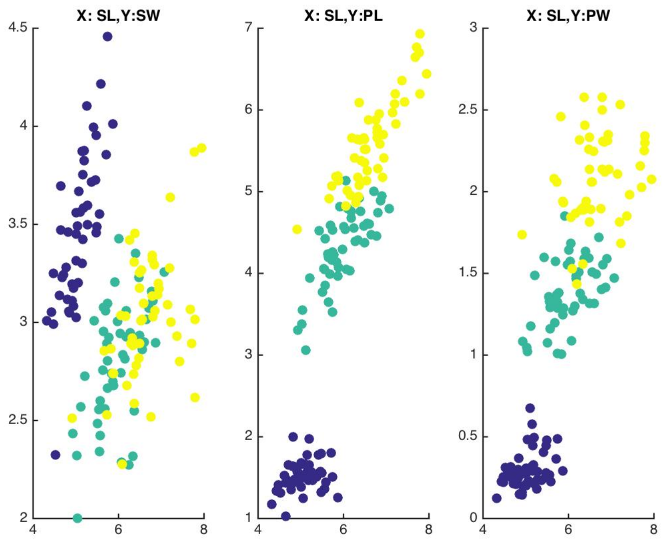


Figure 3: Plots generated with the feature pairs - (1,2), (1,3), and (1,4).

```
    saveas(h,strcat('2a-',num2str(k),'.jpg'),'jpg');
    hold on;
end;
hold off;
```

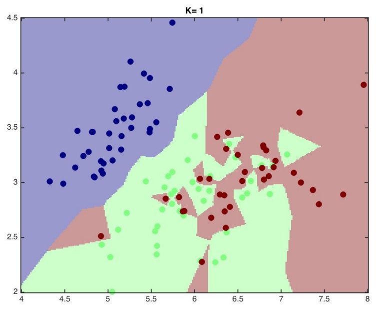
(b) The error is computed using a different function `errorTrain()`, which compares the expected and actual assigned classes and checks for the matches. `errorTrain()` is described in Listing 11. Listing 12 describes the whole approach to solve the problem given. I used the `semilogx` based graph to present the test-vs-train error plotting for various values of k and got $k = 50$ to be a good value, as also shown by Figure 5. Also to understand the graph in a better way, I changed the value of k to a range of 1 to 100. The results of this are portrayed in Figure 6. It is very similar to the one showed in the class and portrays the overfitting and underfitting zones well.

Listing 11: `errorTrain()` to compute the error in the classes assigned.

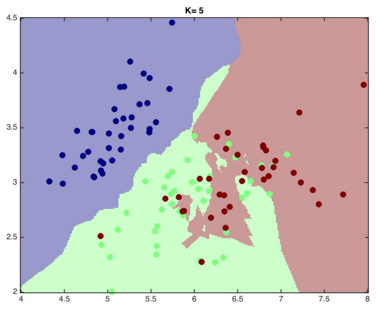
```
function [error] = errorTrain(Yte, YteHat)
% Y = from1ofK(Y1k [,values]) : convert 1-of-K valued Y into discrete ...
    representation
%  optional "values" specifies the possible values of Y (default 1..K)

error = 0;
for i=1:size(Yte,1);
    if(Yte(i) ≠ YteHat(i));
        error = error + 1;
    end;
end;

error = error/size(Yte,1);
```

Listing 12: Code to generate the `semilogx` classifier error graphs

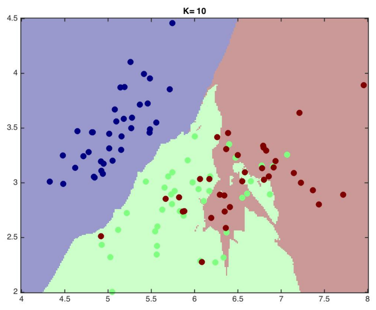
```
%% Part (b)
% Training with Xtr and Ytr
K=[1,2,5,10,50,100,200, 300];
errorsTr = zeros(size(K));
errorsTe = zeros(size(K));
for i=1:size(K,2);
```



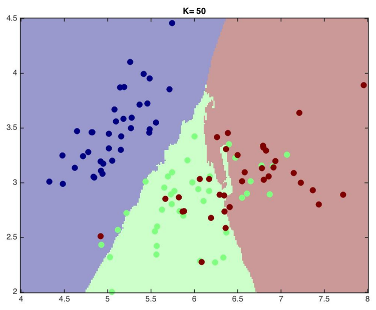
(a) kNN boundaries for K=1



(b) kNN boundaries for K=5



(c) kNN boundaries for K=10



(d) kNN boundaries for K=50

Figure 4: kNN Classifier output for various values of k.

```
    knn = knnClassify( Xtr, Ytr, K(:,i));
    YtrHat = predict( knn, Xtr );
    errorsTr(1,i) = errorTrain(Ytr, YtrHat);
    YteHat = predict( knn, Xte );
    errorsTe(1,i) = errorTrain(Yte, YteHat);
end;

h=figure;
semilogx(log(K), errorsTr);
hold on;
semilogx(log(K), errorsTe);
hold off;

disp(K(find(errorsTe == min(errorsTe))));
    50
```

Problem 3: Bayes Classifiers

(a) I computed the

Listing 13: Calculating the necessary probabilities for Naive Bayes

```
email_data = [0 0 1 1 0 -1;
     1 1 0 1 0 -1;
     0 1 1 1 1 -1;
     1 1 1 1 0 -1;
     0 1 0 0 0 -1;
     1 0 1 1 1 1;
     0 0 1 0 0 1;
     1 0 0 0 0 1;
     1 0 1 1 0 1;
```

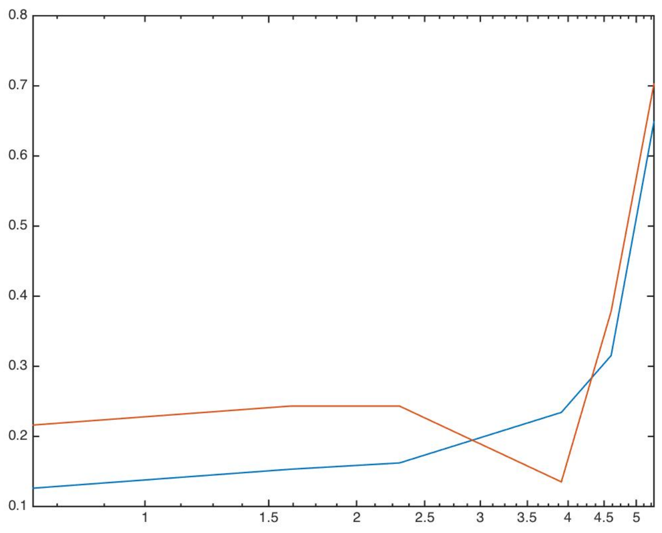


Figure 5: Resulting error rate functions using a semi-log plot with training errors in green and testing error in red for k = [1,2,5,10,50,100,200].

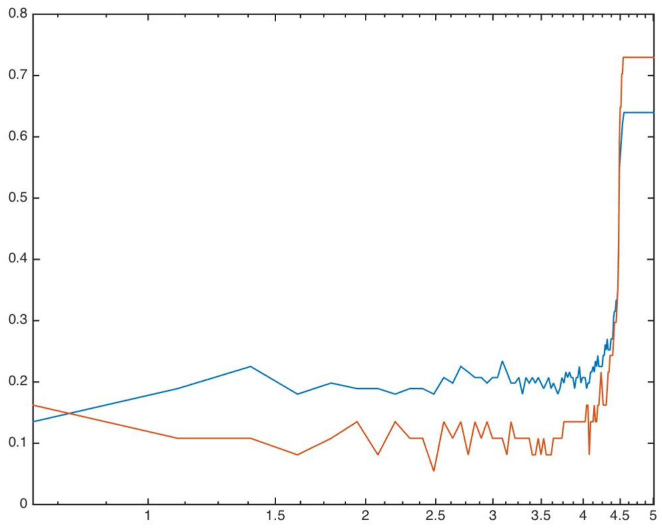


Figure 6: Resulting error rate functions using a semi-log plot with training errors in green and testing error in red for k = 1:100.

```
     1 1 1 1 1 -1];

y = email_data(:,end);
X = email_data(:,1:end-1);
classes = unique(y);

[uniqClasses, numUniqClasses] = count_unique(y);
class_prob = zeros(size(uniqClasses)); % We'll calculate it later.
% class_prob(i, :) = numUniqClasses (i,:) / sum(numUniqClasses);

class_count = zeros(size(uniqClasses,1), size(X,2));

% Counting the occurances of Xi for different values of the classes
% For each feature -
for xi = 1:size(X,2);
    [uniqX, numUniqX] = count_unique(X(:,xi));
    for xn = 1:size(X,1);
        if X(xn,xi) == 1;
            class_count(find(uniqClasses == y(xn)) , xi) = ...
                class_count(find(uniqClasses == y(xn)) , xi) + 1;
        end;
    end;
end;

% Gets the probabilities of the ones cases, where the data is 1.
probabilities_of_one_cases = zeros(size(class_count));
for col = 1:size(probabilities_of_one_cases, 2);
    for row = 1:size(probabilities_of_one_cases, 1);
        probabilities_of_one_cases(row,:) = class_count(row,:) ./ ...
            numUniqClasses (row);
    end;
end;

% Gets the probabilities of the zero cases, where the data is 0.
probabilities_of_zero_cases = 1 - probabilities_of_one_cases;

disp('probabilities_of_one_classes');
[probabilities_of_one_cases, uniqClasses]
disp('probabilities_of_zero_classes');
[probabilities_of_zero_cases, uniqClasses]

---------------------------------------
probabilities_of_one_classes
ans =
    0.5000    0.8333    0.6667    0.8333    0.3333   -1.0000
    0.7500         0    0.7500    0.5000    0.2500    1.0000

probabilities_of_zero_classes
ans =
    0.5000    0.1667    0.3333    0.1667    0.6667   -1.0000
    0.2500    1.0000    0.2500    0.5000    0.7500    1.0000
```

(b) For computing the class using the Naive Bayes classifier, we used the basic code used in the previous question (Listing 13) and computing it by calculating the chance of the class being one of the given classes (+1 and -1 in our example). The code to compute the associated class is given in Listing 14.

- For input = [0, 0, 0, 0, 0]: The computed class of the email is 1 with a probability of 0.8351. The system would recommended this email to be read.
- For input = [1, 1, 0, 1, 0]: The computed class of the email is -1 with a probability of 1. The system would recommended this email to not be read.

Listing 14: For some input

```
%
% P(y|X) = p(X|y) p(y) / P(X)
% P(y = -1 |X) = p(X|y = -1) p(y = -1) / P(X)

% P(y = -1 |X) =
% p(X|y = -1) p(y = -1)
% ------------------------------------
%(p(X|y=-1) p(y=-1) + p(X|y=1) p(y=1))

% P(y = 1 |X) =
% p(X|y = 1) p(y = 1)
% ------------------------------------
%(p(X|y=-1) p(y=-1) + p(X|y=1) p(y=1))

input = [0, 0, 0, 0, 0];
% input = [1, 1, 0, 1, 0];

% finding the classes for -1
productClass1 = 1;
for i = 1:size(input,2);
    if input(:,i) == 1;
       productClass1 = productClass1 *  probabilities_of_one_cases(1,i);
    else
       productClass1 = productClass1 *  probabilities_of_zero_cases(1,i);
    end;
end;

% finding the classes for 1
productClass2 = 1;
for i = 1:size(input,2);
    if input(:,i) == 1;
       productClass2 = productClass2 *  probabilities_of_one_cases(2,i);
    else
       productClass2 = productClass2 *  probabilities_of_zero_cases(2,i);
    end;
end;

probClass1 = productClass1 * class_prob(1,:) / (productClass1 * ...
    class_prob(1,:) + productClass2 * class_prob(2,:));
probClass2 = productClass2 * class_prob(2,:) / (productClass1 * ...
    class_prob(1,:) + productClass2 * class_prob(2,:));
```

(c) The posterior probability of y = +1 given X = [1, 1, 0, 1, 0] is 0. This corroborates with the fact that X was also a data point which the data was trained on, and its class with -1.

(d) Using a Bayes classifier for such kind of data where there is high cost associated with false positives and false negatives can have high impact. Also, for very high number of features, not assuming independent features can be computationally very expensive as it would be a combinatorial explosion.

Problem 4: Gaussian Bayes Classifiers

I used some of the features provided in the `gaussBayesClassify` class and was able to create

(a) As before, I split data vertically to get the first 2 features of Iris dataset. After getting this data, I computed the number of unique classes and each ones count. This was done using the `count_unique()` function. This is previously used in Problem 3. Once I got the unique classes, I fetched all the indices from the training set which matched the class value and found the mean and covariance of the temporary matrices using the `mean()` and `cov()` functions of Matlab.

Listing 15: Visualizeing the classifier and the boundaries

```
%% Part(a)
% Splitting by class
[tempUniq, numTemp] = count_unique(Ytr);
mean_matrix = zeros(size(tempUniq,1), size(Xtr,2));
for i = 1:size(tempUniq,1);
    temp = Xtr(find(Ytr==tempUniq(i,:)),:); % All the indexes of
    disp(strcat('Class:',num2str(i)))
    disp('--------------------------------')
    mean_matrix (i,:) = mean(temp); % Mean Stored for future use
    disp('Mean of the class')
    disp(mean_matrix (i,:));
    disp('Covariance matrix')
    disp(cov(temp)); % Creating and displaying the Covariance matrix of ...
        the class data.
end;
```

Listing 16: Output of the code from Listing 18

```
Class:1
--------------------------------
Mean of the class
    5.0721    3.4765
Covariance matrix
    0.1194    0.0880
    0.0880    0.1159

Class:2
--------------------------------
Mean of the class
    5.9581    2.8093
Covariance matrix
    0.2967    0.0939
    0.0939    0.1138

Class:3
--------------------------------
Mean of the class
    6.5659    3.0264
Covariance matrix
    0.3730    0.1039
    0.1039    0.1039
```

(b) The scatter plot generated by Listing 18 is given in Figure 7.

Listing 17: Visualizeing the classifier and the boundaries

```
h = figure;
scatter(Xtr(:,1), Xtr(:,2), 50, Ytr, 'filled');
saveas(h,'scatter-bayes.jpg','jpg');
```

(c) The scatter plot along with the contours based on the covariance matrix and the mean of data is generated by Listing 16 is given in Figure 8.

Listing 18: Visualizeing the classifier and the boundaries

```
% Evaluate each point of feature space and predict the class
[tempUniq, numTemp] = count_unique(Ytr);
mean_matrix = zeros(size(tempUniq,1), size(Xtr,2));
color = {'blue','green','yellow'};
figure;
```

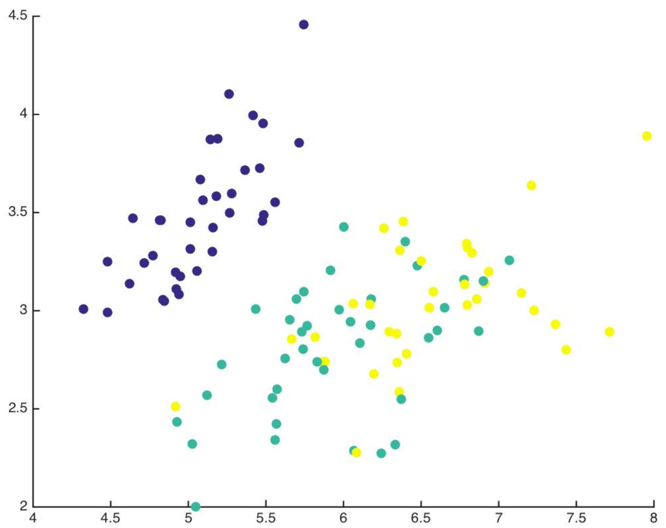


Figure 7: Scatter plot of Iris data set used Gaussian Bayes Learner.

```
scatter(Xtr(:,1),Xtr(:,2), 50, Ytr, 'filled');
hold on;
for i = 1:size(tempUniq,1);
    temp = Xtr(find(Ytr==tempUniq(i,:)),:); % All the indexes of
    mean_matrix (i,:) = mean(temp); % Mean Stored for future use
    plotGauss2D(mean(temp),cov(temp), color{i});
    hold on;
end;
hold off;
```

(d) The code given in Listing 19 generates the plot given in Figure 9.

Listing 19: Visualizeing the classifier and the boundaries

```
%% Part(d)
h = figure;
bc = gaussBayesClassify( Xtr, Ytr );
plotClassify2D(bc, Xtr, Ytr);
saveas(h,'bayes-2.jpg','jpg');
```

(e) The code given in Listing 20 computes the training error rate and the testing error rate. The code uses `errorTrainBayes()` to compute the number of misclassifications and the rate. It is a variant of Listing 11.

Listing 20: Calculating the error rate.

```
%% Part(e)
% Training error
bc = gaussBayesClassify( Xtr, Ytr );
YtrHat = predict( bc, Xtr );
[errorsCount, errorRate] = errorTrainBayes(Ytr, YtrHat);
>> [errorsCount, errorRate]
ans =
   23.0000    0.2072

% Testing error
bcTe = gaussBayesClassify( Xte, Yte );
YteHat = predict( bcTe, Xte );
[errorsCount, errorRate] = errorTrainBayes(Yte, YteHat);
```

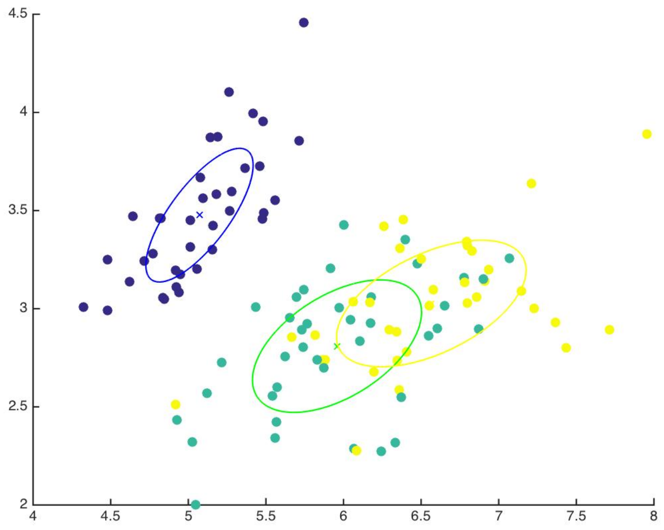


Figure 8: Scatter plot with counters of Iris data set.

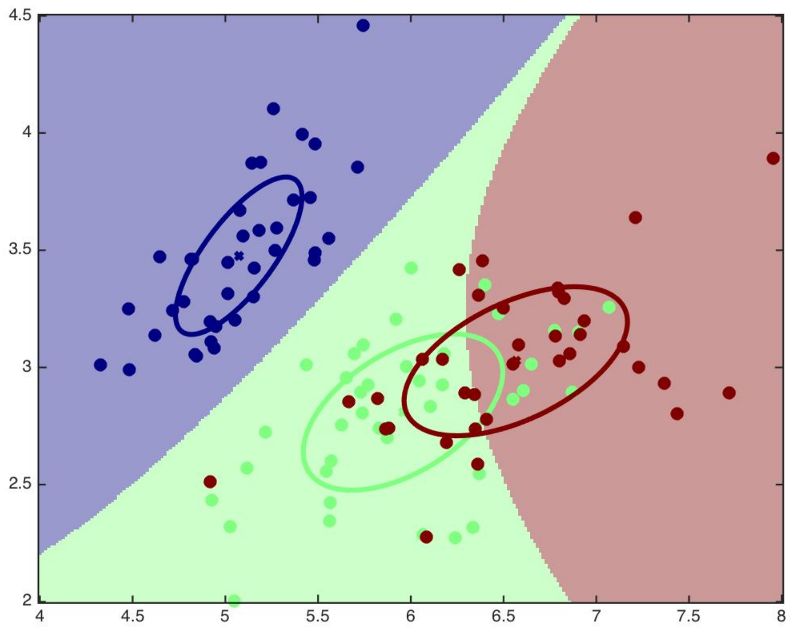


Figure 9: Bayes classifier in action: Generating the decision boundaries using Bayes classifier.

```
>> [errorsCount, errorRate]
ans =
     9.0000    0.2432
```

(f) When training the data with all the features of the data set, I saw tremendous improvement in the working of the classifier. It predicted all the test data results with 100% accuracy. The code of the same is given in the Listing 21. I've put the output within the code itself.

Listing 21: Bayesian classifier for all the 4 features of the Iris data set.

```
%% Part(f)
bc = gaussBayesClassify( Xtr, Ytr );
YtrHat = predict( bc, Xtr );
[errorsCountTr, errorRateTr] = errorTrainBayes(Ytr, YtrHat);
[errorsCountTr, errorRateTr]
% Output
ans =
     2.0000    0.0180

bcTe = gaussBayesClassify( Xte, Yte );
YteHat = predict( bcTe, Xte );
[errorsCountTe, errorRateTe] = errorTrainBayes(Yte, YteHat);
[errorsCountTe, errorRateTe]
% Output
ans =
      0     0
```