**PROBLEM**

This problem is to be implemented in R.

**Dataset**

The dataset is 2D data generated synthetically so that you can visualize the problem. Also, the test set covers a large region so that you can see which parts are labeled as class 1 and which parts are class 0

**To do**

1. Read the dataset GaussTrain.txt. First 2 columns are features and the last column is the label. You can visualize the data using the plot commands given in the following text.

2. Generate your MV Gaussian training model using the training set

3. Determine the predictions on the training data and see where the errors are.

4. Determine the predictions on the test data given in GaussTest.txt file and plot the decision boundary.

**R Commands: Accessing data, rows and columns**

**All indices in R start from 1**

datAll=read.table(“…”) #replace … with filename

datAll = data.matrix(datAll) #this will convert to a matrix data structure

# above is a built in function for reading text files. The entire dataset will be stored in datAll

f1 = datAll[:,1] #this stores first column in f1

r1 = datAll[1,] #this stores first row in r1

ncol(datAll) # this returns total cols in datAll

x= dat[,-1] #stores all columns of dat except column 1 in x

labels = datAll[,ncol(datAll)] #this will store last column of datAll in labels

trainX = datAll[,-ncol(datAll)] #this will store all features except label in dat

oneClass = labels==1 #this will give you all indices of class = 1

zeroClass = labels==0 #this will give you all indices of class = 0

oneDat = trainX[oneClass,] #this will give you the data matrix for class = 1

zeroDat = trainX[zeroClass,] #this will give you the data matrix for class = 0

oneMean = colMeans(oneDat) #this will give the mean of class = 1

oneCov = cov(oneDat) #this will give the covariance matrix of class = 1

invOneCov = solve(oneCov) #this will give the inverse of matrix

%\*% #this is the operator for multiplying matrices in R

**Training Phase and Generating the model**

Write a script in R to learn the Gaussian distribution’s parameters for the two classes.

1. You need to estimate the mean vector and covariance matrix for both classes (see the three cases below).
2. You need to compute the priors for the two classes

**Testing the model / Predicting the labels**

1. Write a function to compute the Gaussian density function at x when given a point x (or multiple points with x as a matrix), mean vector and covariance matrix. The prototype is:

gauss <- function(x, meanVe, covM){ #implement }

Here if x is a matrix then the function can return a column vector for probability density values and if it is row vector then the function should return only one value.

2. Compute the posterior P(C=0|**x**) and P(C=1|**x**) for each row and predict the label using MAP for both train and test set

3. One answer for posterior is given in the report and you can use it to check your answer.

**3 different cases to experiment with**

**Case 1**

Use identity matrix as covariance matrix for both classes and predict the labels for both training and test set.

**Parameters computed in training phase:** Mean vector for both classes

**Case 2**

Compute the covariance matrix for the entire data and use this covariance matrix to predict labels for both training and test set

**Parameters computed in training phase:** Mean vector for both classes + one covariance matrix computed from the entire training set

**Case 3**

Compute the full covariance matrix separately for both class 1 and class 0 and use them to predict the labels for both training and test set.

**Parameters computed in training phase:**  Mean vector for both classes + covariance matrix for both classes.

**Main script**

Once you have implemented the above functions write a main script that:

1. Reads training data
2. Finds the model parameters (relevant to the distribution to use)
3. Predicts labels of the training data by computing posterior
4. Reads the test data and predicts labels of test data by computing posterior

**Plotting data**

You can use the plot and points command to visualize the data for both classes on the same graph. For example:

plot(oneDat[,1],oneDat[,2],col=‘yellow’) #this will plot data points for class one in yellow color

points(zeroDat[,1],zeroDat[,2],col=‘green’) #this will plot data points for class zero in green color

#on the same graph

mistake = predict!=labels #here suppose your predictions are stored in predict

points(trainX[mistake,1],trainX[mistake,2],col=‘yellow’) #this will plot all the mistakes in red color

**Plotting the decision boundary**

The decision boundary has class 0 on one side and class 1 on the other side. It can be a line or some non-linear function of the features. The given test data points cover a large region of the feature space. Predict the label of each test point. Next, plot the points whose predicted label is one in red and plot the points whose predicted class labels is zero in blue as shown above. This will clearly indicate how the model separates the two classes and what the decision boundary looks like.