Q8. Use the likelihood ratio test (LRT) for the binary classification using the dataset ("data_q4_q5.xlsx"). You must use a 5-fold CV-based selection of training and test instances to evaluate the LRT classifier. Evaluate the accuracy, sensitivity, and specificity values for the binary classifier.

```
import pandas as pd
import math
import numpy as np
import matplotlib.pyplot as plt
```

```
from google.colab import drive, files
# drive.mount("/content/gdrive")
uploaded = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving data_q4_q5.xlsx to data_q4_q5.xlsx
```

```
%cd /content/gdrive/My Drive/NNFL/Assignment1
```

```
/content/gdrive/My Drive/NNFL/Assignment1
```

```
# extracting the data
data = pd.read_excel("data_q4_q5.xlsx")
data = np.asarray(data)

print(data)
# data = np.random.permutation(data)
# print(data)
```

```
[[17.99 10.38 122.8 ... 0.4601 0.1189 'M']
 [20.57 17.77 132.9 ... 0.275 0.08902 'M']
 [19.69 21.25 130.0 ... 0.3613 0.08758 'M']
 ...
 [16.6 28.08 108.3 ... 0.2218 0.0782 'M']
 [20.6 29.33 140.1 ... 0.4087 0.124 'M']
 [7.76 24.54 47.92 ... 0.2871 0.07039 'B']]
```

```
# function for normalising data
def norm(data):
  # norm_data = data
  mean = np.mean(data, axis = 0)
  std = np.std(data, axis = 0)
  norm_data = (data-mean)/std
  return norm_data
```

```
# splitting into input and output
X = data[:, :-1]  #input
y = data[:,-1]    #output
for i in range(len(y)):
```

```
    if y[i] == 'M':
      y[i] = 1
    elif y[i] == 'B':
      y[i] = 2
  print(y)

  # normalizing X and y
  X = X.astype(float)
  X = norm(X)
  # y = norm(y)
  # print(X)
  # print(y)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 1
 2 1 2 1 1 2 2 2 1 1 2 1 1 1 2 2 2 1 2 2 1 1 2 2 2 1 1 2 2 2 2 1 2 2 1 2 2
 2 2 2 2 2 2 1 1 1 2 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 2 2 1 2 2 1 2 2 2 2 1 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 1 1 2 2 2 2 1 2 2 1 1 1 2 1
 2 1 2 2 2 1 2 2 1 1 2 1 1 1 1 2 1 1 1 2 1 2 1 2 2 1 2 1 1 1 1 2 2 1 1 2 2
 2 1 2 2 2 2 2 1 1 2 2 1 2 2 1 1 2 1 2 2 2 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 2 1 1 1 2 2
 2 2 1 2 1 2 1 2 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1
 1 2 1 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2
 2 1 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2
 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2
 2 2 2 2 2 2 1 2 1 2 2 1 2 2 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 2 1 2 1 1
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 1 1 1 1 1 1 2]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 1
 2 1 2 1 1 2 2 2 1 1 2 1 1 1 2 2 2 1 2 2 1 1 2 2 2 1 1 2 2 2 2 1 2 2 1 2 2
 2 2 2 2 2 2 1 1 1 2 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 2 2 1 2 2 1 2 2 2 2 1 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 1 1 2 2 2 2 1 2 2 1 1 1 2 1
 2 1 2 2 2 1 2 2 1 1 2 1 1 1 1 2 1 1 1 2 1 2 1 2 2 1 2 1 1 1 1 2 2 1 1 2 2
 2 1 2 2 2 2 2 1 1 2 2 1 2 2 1 1 2 1 2 2 2 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 2 1 1 1 2 2
 2 2 1 2 1 2 1 2 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1
 1 2 1 1 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2
 2 1 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 1 2 2 2 2 2 1 2 2
 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2
 2 2 2 2 2 2 1 2 1 2 2 1 2 2 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 2 1 2 1 1
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 1 1 1 1 1 1 2]
```

```
  # splitting the data into folds
  def cross_validation(k, X, y):
    print(len(X))
    fold_size = int(len(X)/5)
    # print(fold_size)
    X_testing = X[k*fold_size:(k+1)*fold_size]
    y_testing = y[k*fold_size:(k+1)*fold_size]
    X_training = np.delete(X, slice(k*fold_size, (k+1)*fold_size), axis = 0)
    y_training = np.delete(y, slice(k*fold_size, (k+1)*fold_size), axis = 0)
    print(len(X_testing), len(y_testing))
    print(len(X_training), len(y_training))
    return X_testing, y_testing, X_training, y_training
```

```
X_testing, y_testing, X_training, y_training = cross_validation(0, X, y)
# print(X_testing, y_testing, "\n\n", X_training, y_training)
```

```
569
113 113
456 456
```

```
sens = []
spec = []
acc = []
for i in range(5):
  X_testing, y_testing, X_training, y_training = cross_validation(i, X, y)
  y_pred = []
  for i in range(len(X_testing)):
    y_pred.append(lrtrule(X_testing[i], X_training, y_training))
  # a = TP, b = FN, c = FP, d = TN
  print(y_pred, "\n\n", y_testing)
  a, b, c, d = confusion_mat(y_pred, y_testing)
  acc.append((a+d)/(a+b+c+d))
  sens.append((a)/(a+b))
  spec.append((d)/(d+c))
  # print(sens, spec, acc)
avg_sens = sum(sens)/len(sens)
avg_spec = sum(spec)/len(spec)
avg_acc = sum(acc)/len(acc)
print(avg_sens, avg_spec, avg_acc)
```

```
569
113 113
456 456
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1,

 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 1
 2 1 2 1 1 2 2 2 1 1 2 1 1 1 2 2 2 1 2 2 1 1 2 2 2 1 1 2 2 2 2 1 2 2 1 2 2
 2 2]
569
113 113
456 456
[2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2,

 [2 2 2 2 1 1 1 2 1 1 2 2 2 1 1 2 1 2 1 1 2 1 1 2 2 1 2 2 1 2 2 2 2 1 2 2 2
 2 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 1 1 2 2 2 2 1 2 2 1 1 1 2 1 2 1
 2 2 2 1 2 2 1 1 2 1 1 1 1 2 1 1 1 2 1 2 1 2 2 1 2 1 1 1 1 2 2 1 1 2 2 2 1
 2 2]
569
113 113
456 456
[2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1,

 [2 2 2 1 1 2 2 1 2 2 1 1 2 1 2 2 2 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2
 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 1 1 1 2 2 2 2 1 2
 1 2]
569
113 113
```

```
456 456
[1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1,

 [1 2 2 2 1 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 1 2 1 1 2 2
 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2
 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 1 2 2 2 2 2 1 2 2 1 2 1 2 2 1
 2 1]
569
113 113
456 456
[2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,

 [2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2
 1 2 2 1 2 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 2 1 2 1 1 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
 1 1]
0.9484129036229877 0.9684502424110859 0.9575221238938052
```

```
def lrtrule(x_testing, x, y):
  # print(x_testing.shape)
  n = len(x_testing)
  # print(n)
  # finding prior prob
  # print(len(y))
  p1 = len([i for (i, val) in enumerate(y) if val == 1 ])
  p2 = len([i for (i, val) in enumerate(y) if val == 2 ])
  p1, p2 = p1/(len(y)), p2/(len(y))
  # print(p1 + p2)
  # splitting the input data into it's different classes
  x1 = np.array([x[i] for (i, val) in enumerate(y) if val == 1 ])
  x2 = np.array([x[i] for (i, val) in enumerate(y) if val == 2 ])
  # print(x1)
  m1 = np.mean(x1, axis = 0)
  m2 = np.mean(x2, axis = 0)
  # print(m1)
  cov1 = np.cov(np.transpose(x1.astype(float)))
  cov2 = np.cov(x2.astype(float).T)
  # print(cov1, cov2)
  # calculating likelihood of P(X,yi)
  coeff1 = 1 /(( 2 * np.pi )** (n/2) )*np.linalg.det(cov1)** 0.5 )
  coeff2 = 1 /(( 2 * np.pi )** (n/2) )*np.linalg.det(cov2)** 0.5 )
  # print(coeff1, coeff2)
  l1 = coeff1*np.exp(- 0.5 * np.dot(np.dot((x_testing - m1),np.linalg.inv(cov1)),(x_testin
  l2 = coeff2*np.exp(- 0.5 * np.dot(np.dot((x_testing - m2),np.linalg.inv(cov2)),(x_testin
  # print(l1)
  # print(l2)
  # lrt rule
  if (l1/p2) > (l2/p1):
    return 1
  else :
    return 2


def confusion_mat(y_predicted, y_testing):
  a, b, c, d = 0 , 0 , 0 , 0
```

```
    for i in range(len(y_testing)):
      if y_testing[i] == 1 :
        if y_predicted[i] == 1 :
          a = a + 1
        if y_predicted[i] == 2 :
          b = b + 1
      if y_testing[i] == 2 :
        if y_predicted[i] == 1 :
          c = c + 1
        if y_predicted[i] == 2 :
          d = d + 1
    return a, b, c, d
```