Q9. Implement the Maximum a posteriori (MAP) decision rule for the multiclass classification tasks. You must use a 5-fold CV-based selection of training and test instances for the MAP classifier. You must use the dataset data_q6_q7.txt for this question. Evaluate individual accuracy and overall accuracy of MAP classifier.

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

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

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

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

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

```
# 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
```

```
# extracting the data
data = pd.read_excel("data_q6_q7.xlsx")
data = np.asarray(data)
data = np.random.permutation(data)
print(data)
```

```
[[19.15   16.45    0.889  ...  3.084   6.185   2.    ]
 [14.11   14.1    0.8911 ...  2.7     5.      1.    ]
 [13.89   14.02   0.888  ...  3.986   4.738   1.    ]
```

```
        ...
        [15.99   14.89    0.9064 ...  3.336   5.144   2.    ]
        [14.49   14.61    0.8538 ...  4.116   5.396   1.    ]
        [14.16   14.4     0.8584 ...  3.072   5.176   1.    ]]


# splitting into input and output
X = data[:, :-1]  #input
y = data[:,-1]    #output

# normalizing X and y
X = norm(X)


def likelihood(x, meanmat, covariance):
  n = len(x)
  coeff = 1 /((( 2 * np.pi )** (7/2) )*np.linalg.det(covariance)** 0.5 )
  l = coeff*np.exp(- 0.5 * np.dot(np.dot((x - meanmat),np.linalg.inv(covariance)),(x - mea
  return l


def MAP(x_testing, x, y):
  # finding prior prob
  p1 = len([i for (i, val) in enumerate(y) if val == 1 ])
  p2 = len([i for (i, val) in enumerate(y) if val == 2 ])
  p3 = len([i for (i, val) in enumerate(y) if val == 3 ])
  p1, p2, p3 = p1/len(y), p2/len(y), p3/len(y)

  # 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 ])
  x3 = np.array([x[i] for (i, val) in enumerate(y) if val == 3 ])

  # evidence
  e1, e2, e3 = len(x1)/(len(x)), len(x2)/(len(x)), len(x3)/(len(x))

  m1 = np.mean(x1, axis = 0)
  m2 = np.mean(x2, axis = 0)
  m3 = np.mean(x3, axis = 0)
  cov1 = np.cov(np.transpose(x1.astype(float)))
  cov2 = np.cov(x2.astype(float).T)
  cov3 = np.cov(x3.astype(float).T)

  # likelihood
  l1 = likelihood(x_testing, m1, cov1)
  l2 = likelihood(x_testing, m2, cov2)
  l3 = likelihood(x_testing, m3, cov3)

  # MAP
  map1, map2, map3 = (l1*p1)/e1, (l2 * p2)/e2, (l3 * p3)/e3

  # output
  if max(map1, map2, map3) == map1:
    return 1
  elif max(map1, map2, map3) == map2:
    return 2
  else:
```

```
  else:
    return 3


def confusion_mat(y_predicted, y_testing):
  conf_mat = np.zeros((3,3))
  for i in range(len(y_testing)):
    if y_testing[i] == 1:
      if y_predicted[i] == 1:
        conf_mat [0][0] += 1
      if y_predicted[i] == 2:
        conf_mat [0][1] += 1
      if y_predicted[i] == 3:
        conf_mat [0][2] += 1
    if y_testing[i] == 2:
      if y_predicted[i] == 1:
        conf_mat [1][0] += 1
      if y_predicted[i] == 2:
        conf_mat [1][1] += 1
      if y_predicted[i] == 3:
        conf_mat [1][2] += 1
    if y_testing[i] == 3:
      if y_predicted[i] == 1:
        conf_mat [2][0] += 1
      if y_predicted[i] == 2:
        conf_mat [2][1] += 1
      if y_predicted[i] == 3:
        conf_mat [2][2] += 1
  return conf_mat


ind_acc1 = []
ind_acc2 = []
ind_acc3 = []
overall_acc = []
for i in range(5):
  X_testing, y_testing, X_training, y_training = cross_validation(i, X, y)
  y_predicted = []
  for i in range(len(X_testing)):
    y_predicted.append(MAP(X_testing[i], X_training, y_training))
  # print(y_predicted, y_testing)
  conf_mat = confusion_mat(y_predicted, y_testing)
  # individual accuracy
  acc1 = conf_mat[ 0 ][ 0 ]/sum(conf_mat[ 0 ])
  ind_acc1.append(acc1)
  acc2 = conf_mat[ 1 ][ 1 ]/sum(conf_mat[ 1 ])
  ind_acc2.append(acc2)
  acc3 = conf_mat[ 2 ][ 2 ]/sum(conf_mat[ 2 ])
  ind_acc3.append(acc3)
  # overall accuracy
  acc = (conf_mat[ 0 ][ 0 ] + conf_mat[ 1 ][ 1 ] + conf_mat[ 2 ][ 2 ])/np.sum(conf_mat)
  overall_acc.append(acc)
avg_ind_acc1 = sum(ind_acc1)/len(ind_acc1)
avg_ind_acc2 = sum(ind_acc2)/len(ind_acc2)
avg_ind_acc3 = sum(ind_acc3)/len(ind_acc3)
avg_overall_acc = sum(overall_acc)/len(overall_acc)
```

```
print("Average individual accuracy of class 1:", avg_ind_acc1)
print("Average individual accuracy of class 2:", avg_ind_acc2)
print("Average individual accuracy of class 3:", avg_ind_acc3)
print("Average overall accuracy:", avg_overall_acc)
```

```
209
41 41
168 168
[2, 1, 1, 1, 1, 1, 3, 1, 3, 3, 2, 2, 3, 2, 2, 1, 2, 3, 1, 2, 3, 2, 2, 2, 3, 2, 1, 2,
 3. 2. 1. 1. 3. 1. 2. 1. 3. 3. 1. 1. 3. 3. 3. 1. 3.]
209
41 41
168 168
[1, 2, 1, 1, 2, 3, 3, 1, 1, 3, 2, 2, 1, 1, 1, 3, 1, 1, 3, 1, 1, 3, 1, 1, 1, 2, 3, 1,
 1. 2. 3. 1. 3. 2. 1. 2. 1. 2. 2. 1. 2. 1. 2. 3. 2.]
209
41 41
168 168
[3, 2, 3, 2, 2, 2, 1, 1, 3, 3, 2, 2, 2, 2, 1, 2, 3, 3, 1, 3, 1, 3, 1, 2, 3, 2, 3, 1,
 3. 2. 3. 1. 1. 3. 1. 3. 2. 2. 3. 2. 3. 1. 2. 3. 1.]
209
41 41
168 168
[2, 3, 3, 3, 3, 1, 3, 3, 1, 3, 2, 1, 2, 2, 2, 2, 1, 1, 3, 2, 3, 3, 1, 3, 1, 2, 3, 2,
 1. 2. 3. 2. 2. 1. 1. 2. 2. 2. 3. 1. 2. 3. 3. 1. 3.]
209
41 41
168 168
[2, 2, 3, 2, 1, 3, 2, 1, 1, 2, 1, 3, 3, 3, 3, 2, 1, 2, 1, 3, 2, 3, 1, 1, 3, 2, 1, 3,
 3. 2. 3. 3. 2. 2. 3. 2. 3. 2. 2. 1. 3. 1. 3. 3. 2.]
Individual accuracy of class 1: 0.9084848484848485
Individual accuracy of class 2: 0.9571428571428571
Individual accuracy of class 3: 0.9541666666666666
Overall accuracy: 0.9414634146341463
```