Course Number: EEL 5840

UF ID: 0699 6263

Name: Fang Zhu

**Homework 6:**

**Q1.**

For this problem you are given a high-dimensional Gaussian mixture data set

"WQDataSet HW7.zip". Implement Forward Feature Selection (FFS) and Backward Feature Selection (BFS) methods using the Fisher discriminant ratio to select

the features to discriminate the two types of wine.

(1) How many features should you retain based on the FFS and BFS approaches?

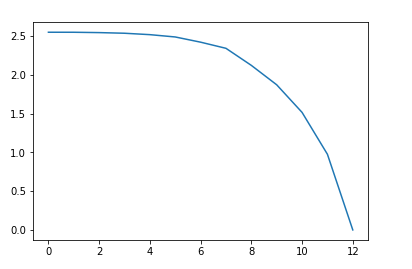
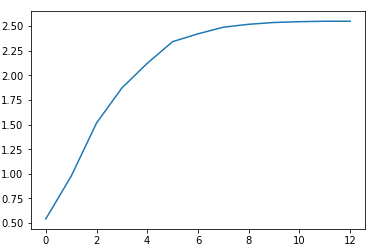
(2) Does classification performance using the Bayes Classifier have best performance

using the feature set you determined using FFS and BFS? Why or why not? How

did you determine this?

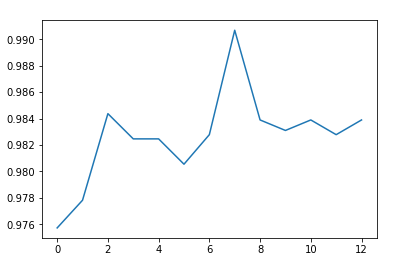
**Solution:**

1). The plots of fisher ratio below. (the first is FFS, the second is BFS)



The sequence of importance to the classification is: 6, 7, 3, 1, 10, 0, 5, 9, 4, 8, 2, 11, 12. According to the plots, the first eight features bring huge change in fisher ratio, while the last 5 features bring little change to the ratio, so I choose the 6, 7, 3, 1, 10, 0, 5, 9.

2). According to the first answer, choose the first eight features. After Using Bayes Classifier, I draw the conclusion by using the confused matrix and rate of right classification. I got the matrix [[4859 21], [39 1578]] and the rate 0.9907. I draw the plot of rate of successfully classified for different numbers of features (accuracy), as follow:



From the plot, when the number of x axis is seven (feature equals to eight), we can get the highest rate. Though eight is the best answer.

**Q2.**

For this problem you are given a high-dimensional Gaussian mixture data set "GMDataSet HW7.txt". Implement the EM algorithm to determine the mixture proportion,

mean and (diagonal) covariance associated with each of the mixture components.

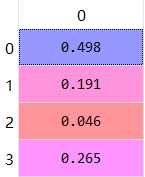
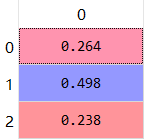
(1) How many mixture components are found in the data? How did you determine this?

(2) What did you estimate for the mixture proportions, means and covariances associated

with each mixture?

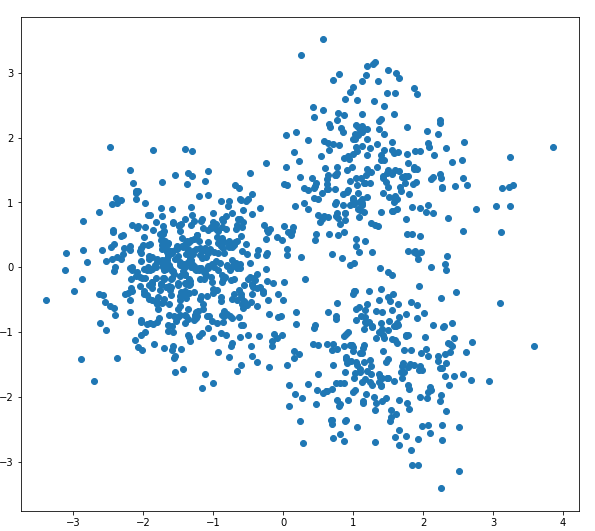
**Solution:**

1). To find how many components are there, I first observe the three parameters when the number of components are changing. Whatever the number of components increasing, the weight only contains three elements with high percentage. So, I guess there are three components in this Mixture Gaussian model. We can see the number of weight of each component when I choose 3 and 4 components. Apparently, one of the weight of the components are quite small.

The proportion for four features The proportion for three features

Afterwards, I draw the PCA of the whole data, the plot illustrating the three components more apparently.



Finally, I compute the fisher ratio of each data. Because if the fisher ratio is too small, it means the two features are very closed to each other. Low ratio may happen when the one component is wrongly divided. The ratio for components of 2, 3 and 4 is as follow:

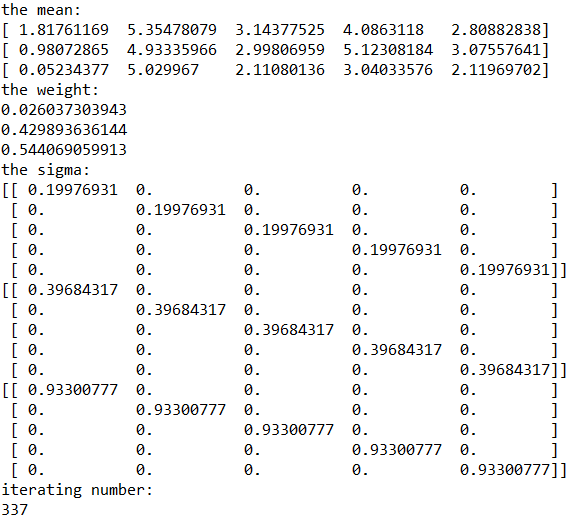
2 components: r = 1.1508783202;

3 components: r01 = 1.72781280869, r02 = 2.25967668, r12 = 4.12286719398;

4 components: r01 = 0.766194577725, r02 = 0.95894888773, r03 = 0.146415406779, r12 = 2.0298346673, r13 = 2.09791407134, r23 = 1.78463303478.

When we guess the number is four, the first three ratio are quite smaller than the last three ratios, which means that the first three components overlay (The first component could be dropped) with other components heavily. Above all, I finally draw a conclusion that this Mixture Gaussian is most likely to contain three components.

2). The weight and sigma for each component are as follow:



**Code:**

1. **Code for Forward Feature Selection:**

import numpy as np

import matplotlib.pyplot as plt

import math

import time

import random

def fisher\_ratio(x1,x2): #dimension x number

mean1 = x1.mean(1)

mean2 = x2.mean(1)

num1 = x1.shape[1]

num2 = x2.shape[1]

mean0 = (num1\*mean1+num2\*mean2)/(num1+num2)

Sb = (num1/(num1+num2))\*(mean1-mean0)@(mean1-mean0).T + (num2/(num1+num2))\*(mean2-mean0)@(mean2-mean0).T

Sw = np.cov(x1) + np.cov(x2)

r = np.trace(Sb@np.linalg.inv(Sw))

return r

#generate data

data1 = np.matrix(np.loadtxt("WhiteWine\_HW7.txt")).T

data2 = np.matrix(np.loadtxt("RedWine\_HW7.txt")).T

d1 = data1.shape[1]

d2 = data2.shape[1]

count = np.zeros(13)

index = np.zeros(13)

#find the best FFS

#find the best in 1st dimension

print("FFS: ")

count1 = 0

index1 = 0

for i in range(13):

x1 = data1[i,:]

x2 = data2[i,:]

mean1 = x1.mean(1)

mean2 = x2.mean(1)

num1 = x1.shape[1]

num2 = x2.shape[1]

mean0 = (num1\*mean1+num2\*mean2)/(num1+num2)

Sb = (num1/(num1+num2))\*(mean1-mean0)@(mean1-mean0).T + (num2/(num1+num2))\*(mean2-mean0)@(mean2-mean0).T

Sw = np.cov(x1) + np.cov(x2)

r = np.asscalar(Sb/Sw)

if count1 < r:

count1 = r

index1 = i

count[0] = count1

index[0] = index1

print(index1)

print(count1)

#find a best in 2nd dimensions

count2 = 0

index2 = 0

for i in range(13):

if i != index1:

x1 = np.vstack((data1[index1,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count2 < r:

count2 = r

index2 = i

count[1] = count2

index[1] = index2

print(index2)

print(count2)

#find a best in 3rd dimensions

count3 = 0

index3 = 0

for i in range(13):

if (i != index1 and i != index2):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count3 < r:

count3 = r

index3 = i

count[2] = count3

index[2] = index3

print(index3)

print(count3)

#find a best in 4th dimensions

count4 = 0

index4 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count4 < r:

count4 = r

index4 = i

count[3] = count4

index[3] = index4

print(index4)

print(count4)

#find a best in 5th dimensions

count5 = 0

index5 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count5 < r:

count5 = r

index5 = i

count[4] = count5

index[4] = index5

print(index5)

print(count5)

#find a best in 6th dimensions

count6 = 0

index6 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count6 < r:

count6 = r

index6 = i

count[5] = count6

index[5] = index6

print(index6)

print(count6)

#find a best in 7th dimensions

count7 = 0

index7 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count7 < r:

count7 = r

index7 = i

count[6] = count7

index[6] = index7

print(index7)

print(count7)

#find a best in 8th dimensions

count8 = 0

index8 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6 and i != index7):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[index7,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[index7,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count8 < r:

count8 = r

index8 = i

count[7] = count8

index[7] = index8

print(index8)

print(count8)

#find a best in 9th dimensions

count9 = 0

index9 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6 and i != index7 and i != index8 ):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[index7,:],data1[index8,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[index7,:],data2[index8,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count9 < r:

count9 = r

index9 = i

count[8] = count9

index[8] = index9

print(index9)

print(count9)

#find a best in 10th dimensions

count10 = 0

index10 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6 and i != index7 and i != index8 and i != index9 ):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[index7,:],data1[index8,:],data1[index9,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[index7,:],data2[index8,:],data2[index9,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count10 < r:

count10 = r

index10 = i

count[9] = count10

index[9] = index10

print(index10)

print(count10)

#find a best in 11th dimensions

count11 = 0

index11 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6 and i != index7 and i != index8 and i != index9 and i != index10 ):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[index7,:],data1[index8,:],data1[index9,:],data1[index10,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[index7,:],data2[index8,:],data2[index9,:],data2[index10,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count11 < r:

count11 = r

index11 = i

count[10] = count11

index[10] = index11

print(index11)

print(count11)

#find a best in 12th dimensions

count12 = 0

index12 = 0

for i in range(13):

if (i != index1 and i != index2 and i != index3 and i != index4 and i != index5 and i != index6 and i != index7 and i != index8 and i != index9 and i != index10 and i != index11):

x1 = np.vstack((data1[index1,:],data1[index2,:],data1[index3,:],data1[index4,:],data1[index5,:],data1[index6,:],data1[index7,:],data1[index8,:],data1[index9,:],data1[index10,:],data1[index11,:],data1[i,:]))

x2 = np.vstack((data2[index1,:],data2[index2,:],data2[index3,:],data2[index4,:],data2[index5,:],data2[index6,:],data2[index7,:],data2[index8,:],data2[index9,:],data2[index10,:],data2[index11,:],data2[i,:]))

r = fisher\_ratio(x1,x2)

if count12 < r:

count12 = r

index12 = i

count[11] = count12

index[11] = index12

print(index12)

print(count12)

#find the best in 13th dimension

print("BFS: ")

count13 = 0

index13 = 0

x1 = np.vstack((data1[0:(i),:],data1[(i+1):13,:]))

x2 = np.vstack((data2[0:(i),:],data2[(i+1):13,:]))

r = fisher\_ratio(x1,x2)

count13 = r

count[12] = count13

index[12] = index13

print(index13)

print(count13)

t = np.arange(13)

plt.plot(t,count)

1. **Code for Backward Feature Selection:**

import numpy as np

import matplotlib.pyplot as plt

import math

import time

import random

count = np.zeros(13)

index = np.zeros(13)

def fisher\_ratio(x1,x2): #dimension x number

mean1 = x1.mean(1)

mean2 = x2.mean(1)

num1 = x1.shape[1]

num2 = x2.shape[1]

mean0 = (num1\*mean1+num2\*mean2)/(num1+num2)

Sb = (num1/(num1+num2))\*(mean1-mean0)@(mean1-mean0).T + (num2/(num1+num2))\*(mean2-mean0)@(mean2-mean0).T

Sw = np.cov(x1) + np.cov(x2)

r = np.trace(Sb@np.linalg.inv(Sw))

return r

#generate data

data1 = np.matrix(np.loadtxt("WhiteWine\_HW7.txt")).T

data2 = np.matrix(np.loadtxt("RedWine\_HW7.txt")).T

d1 = data1.shape[1]

d2 = data2.shape[1]

count = np.zeros(13)

index = np.zeros(13)

#find the best BFS

#find the best in 13th dimension

print("BFS: ")

count13 = 0

index13 = 0

x1 = data1

x2 = data2

r = fisher\_ratio(x1,x2)

count13 = r

count[0] = count13

index[0] = index13

xx1 = data1

xx2 = data2

k = 13

#find the best in 12th dimension

for j in range(13):

count1 = 0

index1 = 0

if j == 11:

break

for i in range(13-j):

x1 = np.delete(xx1,i,0)

x2 = np.delete(xx2,i,0)

r = fisher\_ratio(x1,x2)

if count1 < r:

count1 = r

index1 = i

xx1 = np.delete(xx1,index1,0)

xx2 = np.delete(xx2,index1,0)

count[j+1] = count1

#index[j] = index1

t = np.arange(13)

plt.plot(t,count)

1. **Code for Bayes Classification:**

import numpy as np

import matplotlib.pyplot as plt

import math

import time

#initial parameters and load data

data1 = np.matrix(np.loadtxt("WhiteWine\_HW7.txt")).T

data2 = np.matrix(np.loadtxt("RedWine\_HW7.txt")).T

rate = np.zeros(13)

d1 = data1.shape[1]

d2 = data2.shape[1]

d = d1 + d2

data = np.hstack((data1,data2))

vali = np.hstack((np.zeros(d1),np.ones(d2)))

al = np.vstack((data[6,:],data[7,:],data[3,:],data[1,:],data[10,:],data[0,:],data[5,:],data[9,:],data[4,:],data[8,:],data[2,:],data[11,:],data[12,:]))

for j in range(13):

if j == 0:

alldata = al

else:

alldata = np.delete(al,13-j,0)

ald1 = alldata[:,0:d1]

ald2 = alldata[:,d1:d]

p\_prior1 = d1/d

p\_prior2 = d2/d

mean1 = ald1.mean(1)

mean2 = ald2.mean(1)

sig1 = 0

sig2 = 0

for i in range(d1):

sig1 = sig1 + (1/d1)\*(ald1[:,i]-mean1)@(ald1[:,i]-mean1).T

for i in range(d2):

sig2 = sig2 + (1/d2)\*(ald2[:,i]-mean2)@(ald2[:,i]-mean2).T

count1 = 0

count2 = 0

for i in range(d):

g1 = -0.5\*((np.asmatrix(alldata[:,i]) - np.asmatrix(mean1)).T@np.linalg.inv(sig1)@(np.asmatrix(alldata[:,i]) - np.asmatrix(mean1))) - 3.5\*math.log1p(2\*math.pi) - 0.5\*math.log1p(np.linalg.det(sig1)) + math.log1p(p\_prior1)

g0 = -0.5\*((np.asmatrix(alldata[:,i]) - np.asmatrix(mean2)).T@np.linalg.inv(sig2)@(np.asmatrix(alldata[:,i]) - np.asmatrix(mean2))) - 3.5\*math.log1p(2\*math.pi) - 0.5\*math.log1p(np.linalg.det(sig2)) + math.log1p(p\_prior2)

if g1 >= g0:

if vali[i] == 0:

count1 = count1 + 1

else:

if g0 > g1:

if vali[i] == 1:

count2 = count2 + 1

error1 = d1 - count1

error2 = d2 - count2

Matrix1 = np.ones([2,2])

Matrix1[0,0] = count1

Matrix1[1,0] = error1

Matrix1[0,1] = error2

Matrix1[1,1] = count2

print(Matrix1)

print(1-(error1+error2)/(count1+count2))

rate[12-j] = 1-(error1+error2)/(count1+count2)

t = np.arange(13)

plt.plot(t,rate)

1. **Code for PCA:**

import numpy as np

import matplotlib.pyplot as plt

import math

data = np.loadtxt("GMDataSet\_HW7.txt") #1000 x 5

length = data.shape[0]

order = data.shape[1]

fig = plt.figure(figsize = (10,20))

# plot data by PCA for data

x1 = data[:,0]

x2 = data[:,1]

x3 = data[:,2]

x4 = data[:,3]

x5 = data[:,4]

E1 = sum(x1)/x1.size

E2 = sum(x2)/x2.size

E3 = sum(x3)/x3.size

E4 = sum(x4)/x4.size

E5 = sum(x5)/x5.size

Mx1 = np.array([(x1[m]-E1) for m in range(length)])

Mx2 = np.array([(x2[m]-E2) for m in range(length)])

Mx3 = np.array([(x3[m]-E3) for m in range(length)])

Mx4 = np.array([(x4[m]-E4) for m in range(length)])

Mx5 = np.array([(x5[m]-E5) for m in range(length)])

M1 = np.matrix([Mx1,Mx2,Mx3,Mx4,Mx5])

Cov1 = M1@M1.T/1000

eigen\_vals\_1, eigen\_vecs\_1 = np.linalg.eig(Cov1)

eigen\_pairs\_1 = [(np.abs(eigen\_vals\_1[i]), np.array(eigen\_vecs\_1[:,i].T)[0]) for i in range(len(eigen\_vals\_1))]

eigen\_pairs\_1.sort(key = lambda x : x[0],reverse=True)

#print(eigen\_pairs\_1)

w1 = np.hstack((eigen\_pairs\_1[0][1][:, np.newaxis], eigen\_pairs\_1[1][1][:, np.newaxis]))

M1\_pca = M1.T@w1

p1 = fig.add\_subplot(\*[2,1,1])

p1.scatter(np.array(M1\_pca[:,0].T)[0],np.array(M1\_pca[:,1].T)[0])

1. **Code for Fisher ratio between different selections:**

import numpy as np

import matplotlib.pyplot as plt

import math

import copy

import time

from scipy.stats import multivariate\_normal

def fisher\_ratio(mean1,mean2,S1,S2,P1,P2): #dimension x number

mean0 = (P1\*mean1+P2\*mean2)/(P1+P2)

Sb = (P1/(P1 + P2))\*np.matrix((mean1-mean0)).T@np.matrix((mean1-mean0)) + (P1/(P1 + P2))\*np.matrix((mean2-mean0)).T@np.matrix((mean2-mean0))

Sw = S1 + S2

r = np.trace(Sb@np.linalg.inv(Sw))

return r

data = np.loadtxt("GMDataSet\_HW7.txt") #1000 x 5

for m in range(3):

k = m + 2

length = data.shape[0]

order = data.shape[1]

U = np.zeros([k,5])

U\_old = np.zeros([k,5])

P = np.zeros(k)

P\_1 = np.zeros(k)

P\_old = np.zeros(k)

Sig = np.zeros([5,5])

S = np.zeros([5,5\*k])

Sig = np.cov(data.T)

si = np.zeros(k)

S\_old = np.zeros([5,5\*k])

Cu = np.zeros([length,k])

Cd = np.zeros(length)

C = np.zeros([length,k])

Sig = np.cov(data.T)

for i in range(k):

P[i] = 1/k

U[i,:] = data[round(np.random.randint(0,1000)),:]

S[:,(i\*5):((i+1)\*5)] = 1\*np.eye(U[i,:].size)

diff = 1 # the threshold

o = 0

while diff > 0.0001:

if o > 1000:

break

U\_old = copy.copy(U)

S\_old = copy.copy(S)

P\_old = copy.copy(P)

# E-step

for i in range(length):

for j in range(k):

mvn = multivariate\_normal(U[j,:],S[:,j\*5:(j+1)\*5])

Cu[i,j] = P[j]\*mvn.pdf(data[i,:])

#Cu[i,j] = P[j]\*(((2\*math.pi)\*\*(-5/2))\*(np.linalg.det(S[:,j\*5:(j+1)\*5])\*\*(-0.5))\*math.exp(-0.5\*np.matrix((data[i,:]-U[j,:]))@np.linalg.inv(S[:,j\*5:(j+1)\*5])@np.matrix((data[i,:]-U[j,:])).T))

Cd[i] = sum(Cu[i,:])

for i in range(length):

for j in range(k):

C[i,j] = Cu[i,j]/Cd[i]

# M-step

for i in range(k):

P[i] = sum(C[:,i])/length

for i in range(k):

S1 = 0

S2 = 0

for j in range(length):

S1 = S1 + C[j,i]\*(np.matrix((data[j,:]-U[i,:]))@np.matrix((data[j,:]-U[i,:])).T)

S2 = S2 + 5\*C[j,i]

S[:,i\*5:(i+1)\*5] = np.asscalar(S1/S2)\*np.identity(5)

for i in range(k):

U[i,:] = sum(np.matrix(C[:,i])\*data)/sum(C[:,i])

o = o + 1

#converge

diff = sum(sum(abs(U-U\_old))) + sum(sum(abs(S-S\_old))) + sum(abs(P-P\_old))

#print(h)

if k == 4:

print(k)

print(fisher\_ratio(U[0,:],U[1,:],S[:,0:5],S[:,5:10],P[0],P[1]))

print(fisher\_ratio(U[0,:],U[2,:],S[:,0:5],S[:,10:15],P[0],P[2]))

print(fisher\_ratio(U[0,:],U[3,:],S[:,0:5],S[:,15:20],P[0],P[3]))

print(fisher\_ratio(U[1,:],U[2,:],S[:,5:10],S[:,10:15],P[1],P[2]))

print(fisher\_ratio(U[1,:],U[3,:],S[:,5:10],S[:,15:20],P[1],P[3]))

print(fisher\_ratio(U[2,:],U[3,:],S[:,10:15],S[:,15:20],P[2],P[3]))

else:

if k == 3:

print(k)

print(fisher\_ratio(U[0,:],U[1,:],S[:,0:5],S[:,5:10],P[0],P[1]))

print(fisher\_ratio(U[0,:],U[2,:],S[:,0:5],S[:,10:15],P[0],P[2]))

print(fisher\_ratio(U[1,:],U[2,:],S[:,5:10],S[:,10:15],P[1],P[2]))

else:

if k == 2:

print(k)

print(fisher\_ratio(U[0,:],U[1,:],S[:,0:5],S[:,5:10],P[0],P[1]))

1. **Code for the parameters:**

import numpy as np

import matplotlib.pyplot as plt

import math

import copy

import time

import random

from scipy.stats import multivariate\_normal

data = np.loadtxt("GMDataSet\_HW7.txt") #1000 x 5

k = 3

length = data.shape[0]

order = data.shape[1]

U = np.zeros([k,5])

U\_old = np.zeros([k,5])

P = np.zeros(k)

P\_1 = np.zeros(k)

P\_old = np.zeros(k)

Sig = np.zeros([5,5])

S = np.zeros([5,5\*k])

Sig = np.cov(data.T)

si = np.zeros(k)

S\_old = np.zeros([5,5\*k])

Cu = np.zeros([length,k])

Cd = np.zeros(length)

C = np.zeros([length,k])

Sig = np.cov(data.T)

for i in range(k):

P[i] = 1/k

U[i,:] = data[round(np.random.randint(0,1000)),:]

S[:,(i\*5):((i+1)\*5)] = 1\*np.eye(U[i,:].size)

diff = 1 # the threshold

o = 0

while diff > 0.0001:

if o > 1000:

break

U\_old = copy.copy(U)

S\_old = copy.copy(S)

P\_old = copy.copy(P)

# E-step

for i in range(length):

for j in range(k):

mvn = multivariate\_normal(U[j,:],S[:,j\*5:(j+1)\*5])

Cu[i,j] = P[j]\*mvn.pdf(data[i,:])

#Cu[i,j] = P[j]\*(((2\*math.pi)\*\*(-5/2))\*(np.linalg.det(S[:,j\*5:(j+1)\*5])\*\*(-0.5))\*math.exp(-0.5\*np.matrix((data[i,:]-U[j,:]))@np.linalg.inv(S[:,j\*5:(j+1)\*5])@np.matrix((data[i,:]-U[j,:])).T))

Cd[i] = sum(Cu[i,:])

for i in range(length):

for j in range(k):

C[i,j] = Cu[i,j]/Cd[i]

# M-step

for i in range(k):

P[i] = sum(C[:,i])/length

for i in range(k):

S1 = 0

S2 = 0

for j in range(length):

S1 = S1 + C[j,i]\*(np.matrix((data[j,:]-U[i,:]))@np.matrix((data[j,:]-U[i,:])).T)

S2 = S2 + 5\*C[j,i]

S[:,i\*5:(i+1)\*5] = np.asscalar(S1/S2)\*np.identity(5)

for i in range(k):

U[i,:] = sum(np.matrix(C[:,i])\*data)/sum(C[:,i])

o = o + 1

#converge

diff = sum(sum(abs(U-U\_old))) + sum(sum(abs(S-S\_old))) + sum(abs(P-P\_old))

#print(h)

print("the mean:")

print(U[0,:])

print(U[1,:])

print(U[2,:])

print("the weight:")

print(P[0])

print(P[1])

print(P[2])

print("the sigma:")

print(S[:,0:5])

print(S[:,5:10])

print(S[:,10:15])

print("iterating number:")

print(o)