Project 7

Qidi Han

April 8, 2020

Question 1

Implement a random number generator for a random vector X=[X1,X2,X3]' having multivariate Gaussian distribution with u=[1;2;3] sigma=[3 -1 1; -1 5 3; 1 3 4]

Theory

The first step of this question is that calculate the lower triangular matrix from sigma. I will use np.linalg.cholesky() function to do this step. There are 1000 and 10000 task in my example, each task will generate a Gaussian distribution. I will calculate the mean of X1, x2 and x3.I will discussion the result in the explanation part

Algorithm 1: Code for random vector X having multivariate Gaussian distribution

```
/* Formula for compute the X                                    */
1 X^t = A * (Z^T) + mu^T
2
```

```
import pprint
import numpy as np
import scipy.linalg

number_time=10000;
mu=np.array([1 ,2,3])
sigma = np.array([[3 ,-1,1], [-1,5,3], [1,3,4]])
A = np.linalg.cholesky(sigma)
# print (A)
# print (sigma[0]*np.transpose([1, 2, 3]))
# print (L)
X=[];
x1=[];
x2=[];
x3=[];


for i in range(0,number_time):
    x=sum(np.transpose(A*np.random.normal(0, 1, 3)))+np.transpose(mu)
    X.append(x)
    x1.append(x[0])
    x2.append(x[1])
    x3.append(x[2])

print(np.mean(x1))
print(np.mean(x2))
print(np.mean(x3))
print(np.cov(np.transpose(X)))
```

```
0.9952103277828647
1.9918457269109144
2.9803767517074227
[[ 3.05302795 -1.02671324  1.01250879]
 [-1.02671324  4.98411805  2.97701688]
 [ 1.01250879  2.97701688  3.9900059 ]]
```

Figure 1: Code and result for question 1

Explanation

When I try 1000 and 10000 times, The x1, x2 and x3 is around 1 2 and 3. Therefore, our result and program is correct, because our mean is from 1,2 and 3.

Question 2

Implement a random number generator for a random variable with the following mixture distribution: f(x)=0.4N(-1,1)+0.6*N(1,1). Generate a histogram and overlay the theoretical p.d.f. of the random variable.

Theory

In this question, I will generate two random Gaussian number, I will set the probability of the first Gaussian number to 0.4. The probability of choosing the second Gaussian number as 0.6. I will plot the result, and I will plot the theoretical result overlay the experiment result. The mixture distribution is f(X)=0.4N(-1,1)+0.6N(1,1). For this mixture distribution, The probability of chose N(-1,1) will be 0.4. The probability of chose N(1,1) will be 0.6.

Algorithm 2: Second Question.

```
/* Choose the Gaussian distribution                    */
for i in range(N):
if np.random.rand() < p:
s[i] = np.random.normal(mu1, sigma1)
else:
s[i] = np.random.normal(mu2, sigma2)
```

```
[9]: import numpy as np
import matplotlib.pyplot as plt
N = 1000
mu1, sigma1 = -1, 1 # mean and standard deviation of the two models
mu2, sigma2 = 1, 1
p = 0.4
s = np.zeros(N)


for i in range(N):
    if np.random.rand() < p:
        s[i] = np.random.normal(mu1, sigma1)
    else:
        s[i] = np.random.normal(mu2, sigma2)

count, bins, ignored = plt.hist(s, 30, density=True)
plt.plot(bins, p*(1/(sigma1 * np.sqrt(2 * np.pi)) *
                  np.exp( - (bins - mu1)**2 / (2 * sigma1**2) )) + (1-p)*(1/(sigma2 * np.sqrt(2 * np.pi)) *
                 np.exp( - (bins - mu2)**2 / (2 * sigma2**2)) ),linewidth=2, color='r')
plt.show()
```

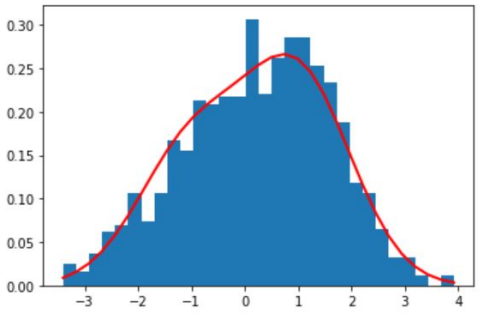


Figure 2: Code and result for question 2

Explanation

The theoretical result is perfectly overlay the experiment result. As I increase the number of the tasks, the result will become much better.

Question 3

Implement a 2-dimensional random number generator for a Gaussian mixture model (GMM) pdf with 2 subpopulations. Use the expectation maximization (EM) algorithm to estimate the pdf parameters of the 2-D GMM from samples. Compare the quality and speed of your GMM-EM estimates using 300 samples from different GMM distributions (e.g. spherical vs ellipsoidal covariance, close vs well-separated subpopulations, etc.).

Theory

In this question, I will use the code that proved by the instructions. There are mu, sigma and probability that the GMM needed. Then I will use GMM-EM function to run the program.

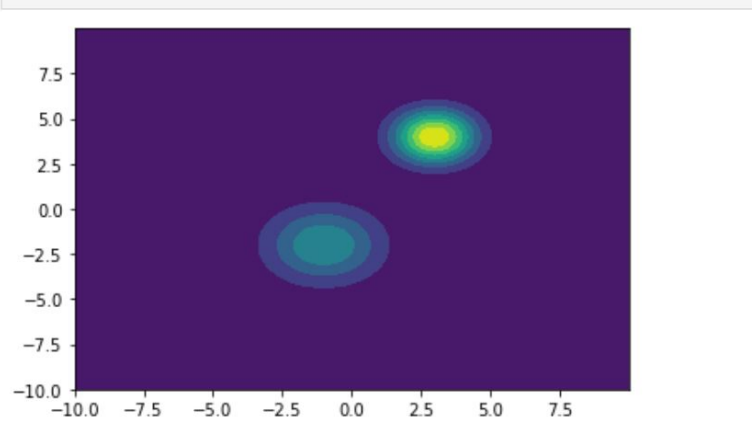
There are two different covariance that I will compute the time.The output will include the log-likelihood, and the computing time.

Algorithm 3: .

```
   /* EM step from the duck University 663-class                    */
 1 E-step ws = np.zeros((k, n)) for j in range(len(mus)):
 2 for i in range(n):
 3 ws[j, i] = pis[j] * mvn(mus[j], sigmas[j]).pdf(xs[i])
 4 ws /= ws.sum(0)
 5 pis = np.zeros(k)
 6 for j in range(len(mus)):
 7 for i in range(n):
 8 pis[j] += ws[j, i]
 9 pis /= n
10 mus = np.zeros((k, p))
   /* step of GMM                                                   */
11 x, y = np.mgrid[-3:3:.01, -3:5:.01] pos = np.empty(x.shape + (2,))
12 pos[:, :, 0] = x; pos[:, :, 1] = y
13 mu1 = [0,0]
14 mu2 = [-1,3]
15 cov1 = [[2.0, 0], [0, 0.5]]
16 cov2 = [[1, 0], [0, 1]]
17 rv1 = multivariate_normal(mu1, cov1)
18 rv2 = multivariate_normal(mu2, cov2)
19 plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))
20
```

```
from scipy.stats import multivariate_normal as mvn
import scipy.stats as st
import time
np.random.seed(123)
start_mar = time.time()

# create data set
n = 300
mus = np.array([[1,-2], [-3,-4]])
sigmas = np.array([[[2, 0], [0, 0.5]], [[1,0],[0,1]]])
poba = np.array([0.25, 0.75])

x, y = np.mgrid[-10:10:.01, -10:10:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y

rv1 = multivariate_normal(mus[0], sigmas[0])
rv2 = multivariate_normal(mus[1], sigmas[1])
plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))
plt.show()

xs = np.concatenate([np.random.multivariate_normal(mu, sigma, int(pi*n))
                     for pi, mu, sigma in zip(poba, mus, sigmas)])

# initial guesses for parameters
pis = np.random.random(2)
pis /= pis.sum()
mus = np.random.random((2,2))
sigmas = np.array([np.eye(2)] * 2)
ll1, pis1, mus1, sigmas1 = em_gmm(xs, pis, mus, sigmas)
end_mar = time.time()
print(end_mar - start_mar)
print(ll1)
```

Figure 3: Code for GMM-EM process




```
2.7388253211975098
-1067.2743738983345
```

Figure 4: result of spherical covariance

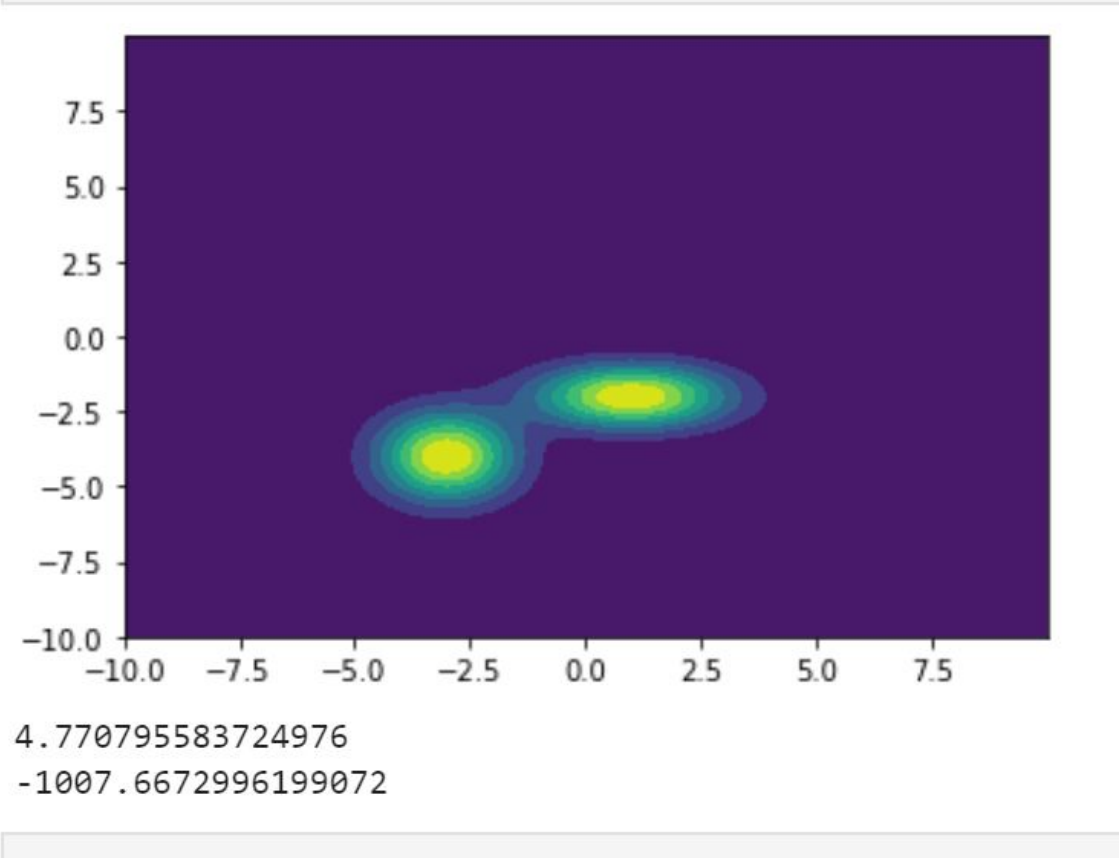


Figure 5: result of ellipsoidal covariance

Explanation

The speed of compute a spherical covariance is around 2.7 s. And the speed of computing ellipsoidal covariance is around 4.7707 s. Therefore, when we give a ellipsoidal covariance, it will have slow process. The log-likelihood is similar to each other for both ellipsoidal and spherical. There are all around 1010.

Question 4

A geyser is a hot spring characterized by an intermittent discharge of water and steam. Old Faithful is a famous cone geyser in Yellowstone National Park, Wyoming. It has a predictable geothermal discharge and since 2000 it has erupted every 44 to 125 minutes. Refer to the addendum data file that contains waiting times and the durations for 272 eruptions. a. Generate a 2-D scatter plot of the data. Run a 2d-means clustering routine on the data for k=2 Show the two clusters on a scatterplot. b. Use a GMM-EM algorithm to fit the dataset

to a GMM pdf. Draw a contour plot of your final GMM pdf. Overlay the contour plot with a scatterplot of the data set. How can you use the GMM pdf estimates to cluster the data?

Theory

There are two part of this question, the first part is run the k-means with 2 class. The first step is read the data from the file. Then I will use np.vstack() function to combine the eruptions and waiting times. Next, I will run kmeans() function to classifier the data points. After I have the two class data point, I will plot the data points and the center of the data points. Finally, I will use those data point to do the second question. The first step is calculate the mean and co-variance for each class. Then I will use those means and co-variance to use countourf() function to plot the data point with GMM pdf.

Algorithm 4: .

```
/* Code for kmean function)                                          */
kmeans = KMeans(n_clusters = 2, random_state = 0).fit(X)
/* compute the means and co-variance                                 */
or i in range(len(x1)): f x1[i]!=float(0): new_x.append(x1[i])new_y.append(y1[i])
v=np.vstack((new_x, new_y))cov1 = np.cov(v)

```

```
]: import numpy as np
from scipy import stats
from sklearn.utils import resample
import math
from random import random
import matplotlib.pyplot as plt
import random
from sklearn.cluster import KMeans

file = open("data1.txt","r")
file2 = open("data2.txt","r")
a=95
#Repeat for each song in the text file
eruptions =[];
waiting=[];
for line in file:
    eruptions.append(float(line))
for line2 in file2:
    waiting.append(float(line2))
file.close()
file2.close()

X = np.vstack((eruptions, waiting)).T #Cascade data points
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

plt.figure(figsize=(9, 9))
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_.astype(float))
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], marker='^', c='cyan')

x1=X[:,0]*kmeans.labels_.astype(float)
y1=X[:,1]*kmeans.labels_.astype(float)
new_x=[];
new_y=[];
for i in range(len(x1)):
    if x1[i]!=float(0):
        new_x.append(x1[i])
        new_y.append(y1[i])

va=np.vstack((new_x,new_y))
cov1=np.cov(va)
new_x=[];
new_y=[];
for i in range(len(x1)):
    if x1[i]==float(0):
        new_x.append(eruptions[i])
        new_y.append(waiting[i])
va=np.vstack((new_x,new_y))
cov2=np.cov(va)

plt.title('Data Points with Labels by K-means Clustering')
plt.show()
mu1 = [kmeans.cluster_centers_[:,0][0] ,kmeans.cluster_centers_[:,1][0]      ]
mu2 = [kmeans.cluster_centers_[:,0][1],kmeans.cluster_centers_[:,1][1]]


rv1 = multivariate_normal(mu1, cov1)
rv2 = multivariate_normal(mu2, cov2)
plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))
plt.show()
```

Figure 6: Code for kmeans and GMM pdf

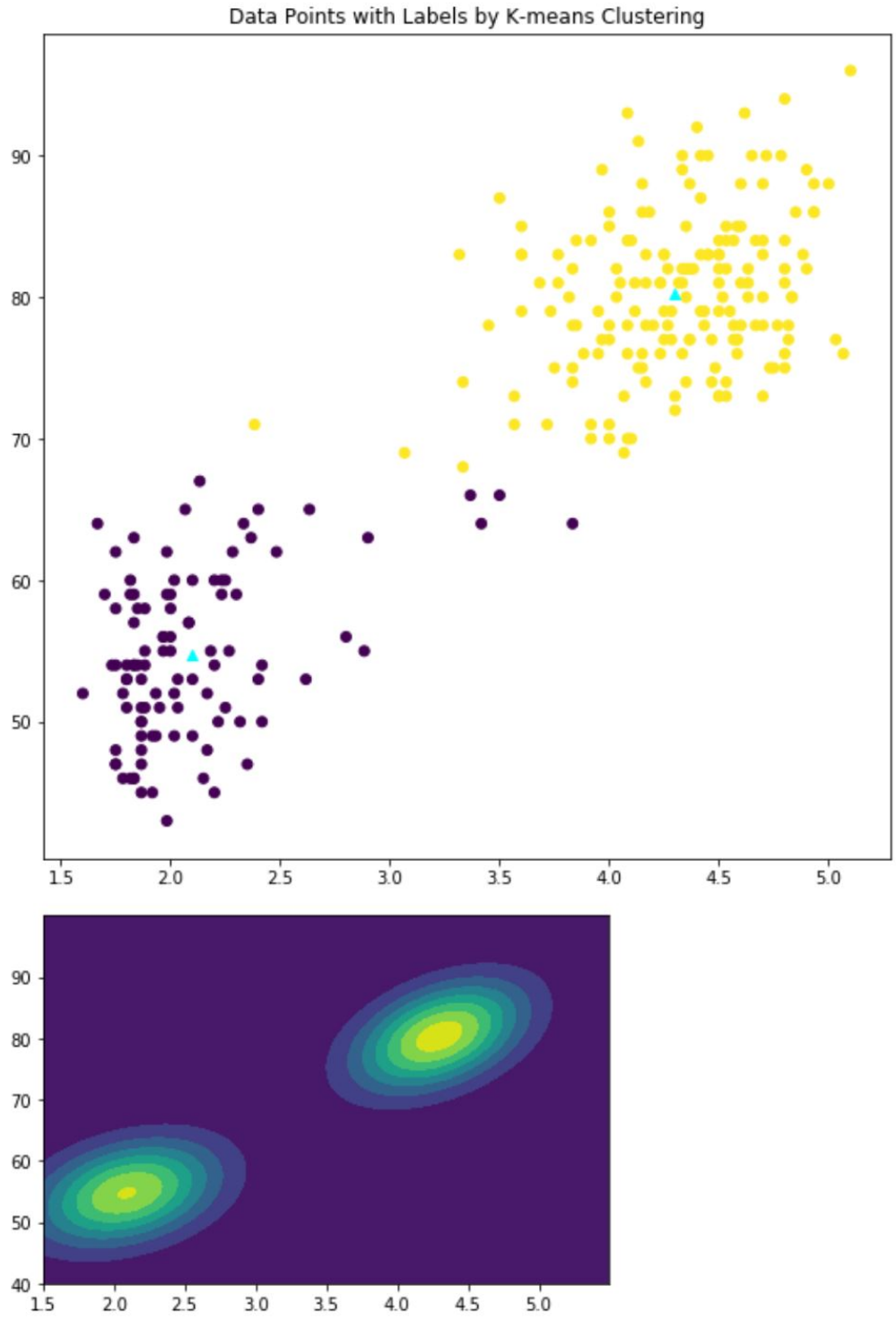


Figure 7: result for kmeans and GMM pdf

Explanation

The figure for kmeans is very close to GMM pdf figure. Therefore, I result is correct for our program. The GMM pdf is very clearly explain how the data spread. And the center of the data is also present in the figure. It is a very good technical for us to learn. The GMM use the spherical and ellipsoidal circle represent the covariance, the mu will represent the center of the data.