EE511 Assignment 7

Zechen Ha

April 6, 2020

Question 1

Experiment

We can generate the multivariate normal random variables by transformation. Construct a random vector Y based on standard normal random vector Z.

$$Y = AZ + b$$

There is always:

$$\mu_Y = \mu_Z + b$$

$$\Sigma_Y = A\Sigma_Z A^T$$

where

$$\Sigma_Y = AA^T$$

$$\mu_Z = [0]$$

So we can write script to generate multivariate normal random vector Y with these formula.

Code

```python
import numpy as np

X = np.random.randn(3,1)
mu = np.array([[1],[2],[3]])
sigma = np.array([[3,-1,1],[-1,5,3],[1,3,4]])
value, vector = np.linalg.eig(sigma)
A = vector.dot(np.diag(np.sqrt(value)))
Y = A.dot(X)+mu
print(Y)
```

Listing 1: Q1 code

Question 2

Experiment

We can complete the assignment by just generating two Gaussian random variable and get the mixture with given algebra. Simulate several times and get the histogram.

Code

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

N = 2000
```

```python
X = []
for i in range(N):
    x = 0.4*np.random.normal(-1,1)+0.6*np.random.normal(1,1) #mu and std
    X.append(x)
X = np.array(X)
mu = 0.2
variance = 1
t = np.linspace(norm.ppf(0.05,mu),norm.ppf(0.95,mu),100)
plt.hist(X,alpha=0.7, edgecolor='black',density=True,bins=20)
plt.plot(t,norm.pdf(t,mu),'r')
plt.title("histogram and pdf of mixture",fontsize=15)
plt.show()
```

Listing 2: Q2 code

Result

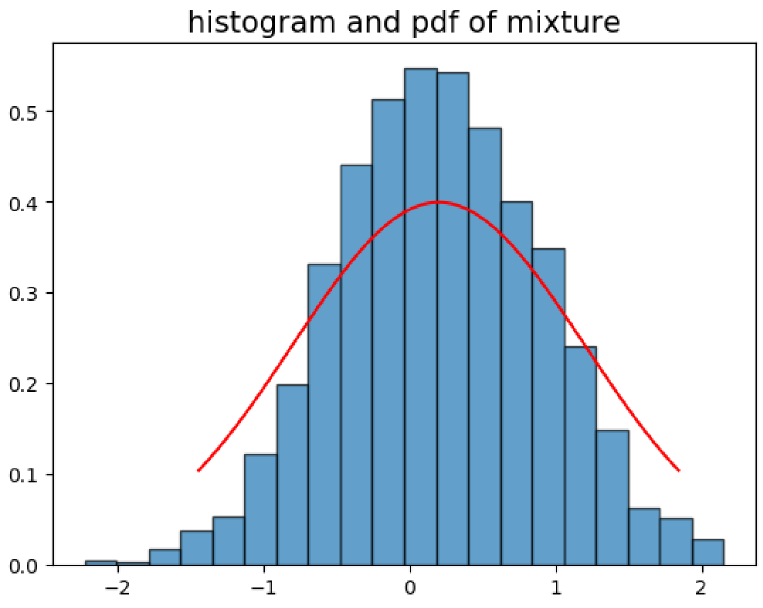

Figure 1: Histogram and theoretical pdf of the mixture

Question 3

Code

```python
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
```

```python
import time

mu1 = [0,0]
mu2 = [-1,3]
cov1 = [[0.5, 0], [0, 0.5]]
cov2 =  [[1, 0], [0, 1]]
p = 0.6
sample = 300
data = []
prelabel = []

for i in range(sample):
    if np.random.rand()<p:
        data.append(multivariate_normal.rvs(mu1,cov1))
        prelabel.append(0)
    else:
        data.append(multivariate_normal.rvs(mu2,cov2))
        prelabel.append(1)

data = np.array(data);prelabel = np.array(prelabel)
start_time = time.time()
gmm = GaussianMixture(n_components=2, covariance_type='full',random_state
    =1).fit(data)
end_time = time.time()
label = gmm.predict(data)
err_num = sum(prelabel!=label)
err_rate = err_num/sample
print(err_rate)
print(end_time-start_time)
print(gmm.means_,'\n',gmm.covariances_)
print(gmm.weights_)
```

Listing 3: Q3 code

Result

To make the quality of EM algorithm more clear, I define the quality as the accuracy rate. The speed of the algorithm is the time spent to calculate the expectation. The comparison is shown as follow.
When the co-variance is spherical, I use the example

$$\mu_1 = \begin{vmatrix} 0 \\ 0 \end{vmatrix} \Sigma_1 = \begin{vmatrix} 0.5 & 0 \\ 0 & 0.5 \end{vmatrix}$$

$$\mu_2 = \begin{vmatrix} -1 \\ 3 \end{vmatrix} \Sigma_2 = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

The accuracy rate is 0.97, and the time spent is 0.004s.
When the co-variance is ellipsoidal, I use the example

$$\mu_1 = \begin{vmatrix} 0 \\ 0 \end{vmatrix} \Sigma_1 = \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix}$$

$$\mu_2 = \begin{vmatrix} -1 \\ 3 \end{vmatrix} \Sigma_2 = \begin{vmatrix} 1 & 0 \\ 0 & 5 \end{vmatrix}$$

The accuracy rate is 0.87, and the time spent is 0.009s.
The above examples are all well-seperated, when the co-variance is closed and ellipsoidal:

$$\mu_1 = \begin{vmatrix} 0 \\ 0 \end{vmatrix} \Sigma_1 = \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix}$$

$$\mu_2 = \begin{vmatrix} 0 \\ -0.5 \end{vmatrix} \Sigma_2 = \begin{vmatrix} 1 & 0 \\ 0 & 5 \end{vmatrix}$$

The accuracy rate is 0.71, and the time spent is 0.007s.

Question 4

Code

```python
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans

data = np.loadtxt(open("faithful.dat.txt",'rb'),skiprows=26)
data = data[:,1:3]
kmeans = KMeans(n_clusters=2, random_state=1).fit(data)
label = kmeans.predict(data)
centers = kmeans.cluster_centers_
fig1 = plt.figure()
ax1 = fig1.add_subplot(111)
ax1.scatter(data[:,0],data[:,1],c=label,s=40)
ax1.scatter(centers[:,0],centers[:,1],c='red',s=80)
ax1.set_title("Kmeans Clusters",fontsize=15)
ax1.set_xlabel("eruptions")
ax1.set_ylabel("waiting")
plt.show()

gmm = GaussianMixture(n_components=2,covariance_type='full',random_state
    =1).fit(data)
labelgm = gmm.predict(data)
mu = gmm.means_
covariance = gmm.covariances_
t = gmm.weights_
x,y = np.mgrid[1:5.5:0.01,40:100:0.1]
pos = np.empty(x.shape+(2,))
pos[:,:,0] = x; pos[:,:,1] = y
fig2 = plt.figure()
```

```
ax2 = fig2.add_subplot(111)
ax2.contourf(x,y,t[0]*multivariate_normal.pdf(pos,mu[0,:],covariance
    [0,:,:])+t[1]*multivariate_normal.pdf(pos,mu[1,:],covariance[1,:,:]))
ax2.scatter(data[:,0],data[:,1],c=labelgm,s=20,cmap='winter')
ax2.set_title("Gaussian mixture",fontsize=15)
ax2.set_xlabel("eruptions")
ax2.set_ylabel("waiting")
plt.show()
```

Listing 4: Q4 Code

Result

The k-means scatter graph is shown as follow:

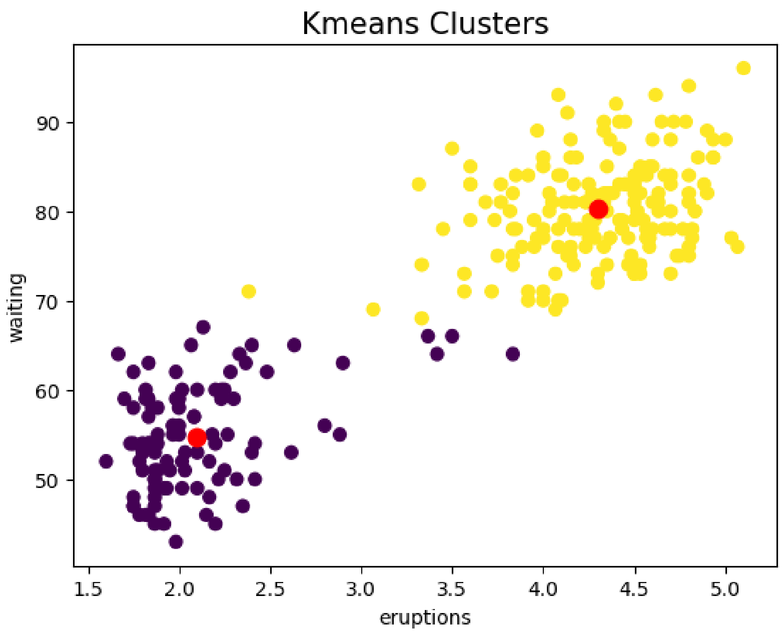

Figure 2: k-means clusters

The EM algorithm estimates the parameters μ and Σ, and get the probability density function based on the final parameters.

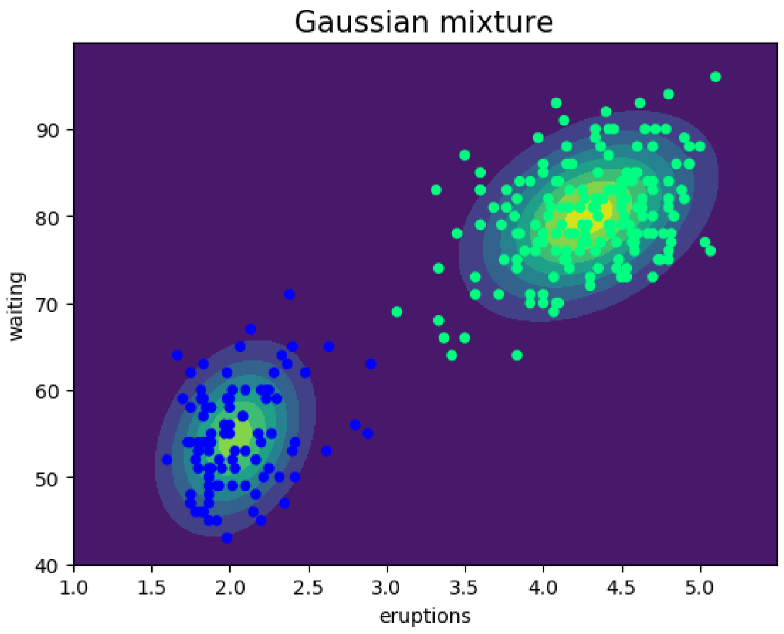

Figure 3: GMM-EM

Different from the k-means algorithm, the GMM-EM is a soft classifier. We classify the data points on the graph based on the estimated probability of belonging to which cluster. When the probability of belonging to the cluster A is larger than B, we classify that it belongs to A. So the probability of the point belonging to B also exists.