EE 511 Project #4: Advanced Clustering
Name: Yang Liu
Student ID number:5847572002
Due: Nov 11

Summary

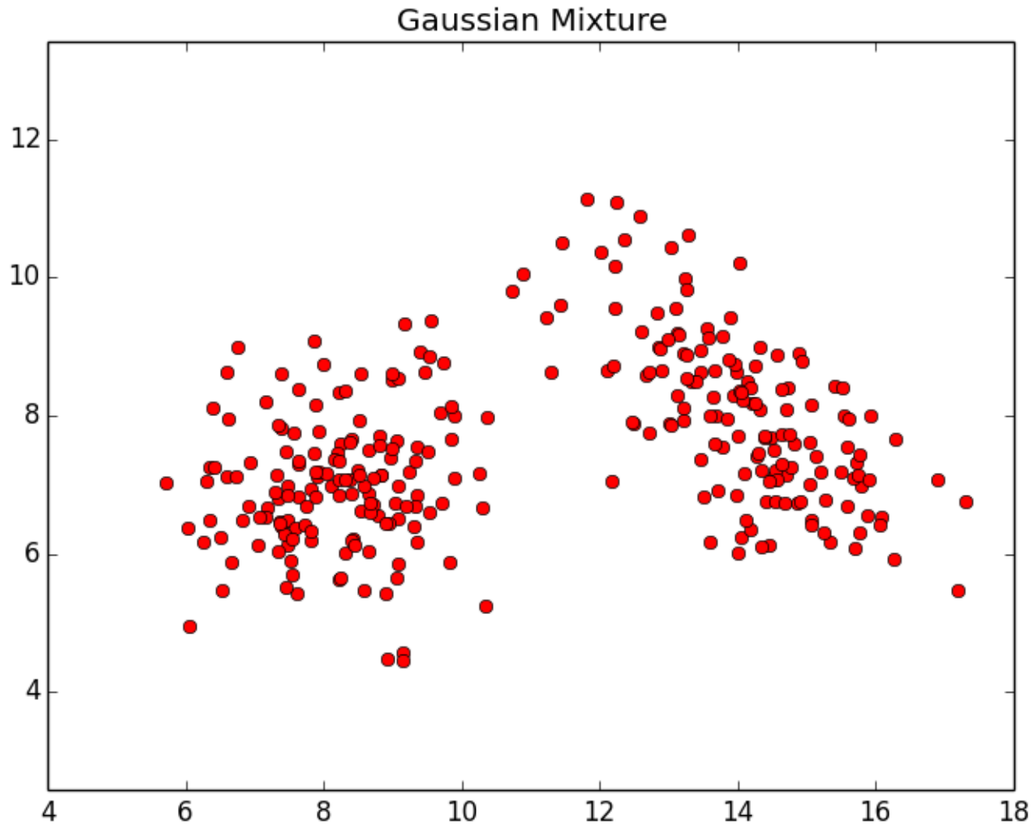
In this project, I realized the generation of GMM and conduct the EM-algorithm to separate the two sub-populations. The most useful tool for this project in python is the class called mixture.GaussianMixture in scikit.learn module, which is a machine learning module. In problem 2, I drew a contour plot of the GMM data for clustering. By accomplishing these two problems, I learned a lot about Gaussian Mixture Module and EM-algorithm, and I understand some useful methods and basic concepts in machine learning and estimating parameters.

Discussion of the results

Problem #1

Question 1
Write a 2-dimensional RNG for a Gaussian mixture model (GMM) pdf with 2 sub-populations. Use any function/sub-routine available in your language of choice.

The main point for this question is to combine 2 different Gaussian distribution together. Because there is a function in numpy called numpy.random.multicariate which can generate 2-dimensional Gaussian distribution samples according to the given mean and variance value, I used it to generate two different 2-dimensianal Gaussian samples and each has 150 samples. Then I used a function in numpy called numpy.vstack to vertically combine these 2 Gaussian samples sets together. Finally, I used a function called numpy.random.shuffle to reorder the GMM sets, which is a 300*2 data set, and each line represent the x and y value of an example. After generating the GMM data sets, I wrote a plot function called pltgmm(Q) to show the generated GMM data sets. The data I created is shown bellow. The parameters is `m1=[14, 8],cov1=[[1.5, -1], [-1, 1.5]],num1 =150, m2=[8, 7], cov2=[[1, 0.1], [0.1, 1]], num2 = 150` .So this is GMM data set which totally contains 300 samples.

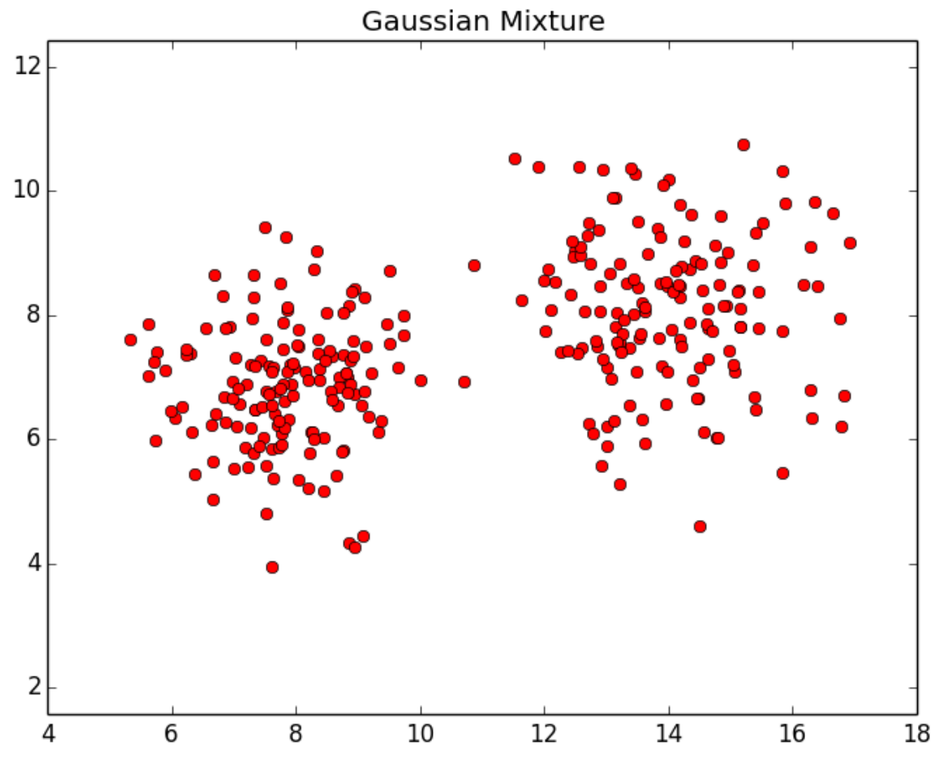


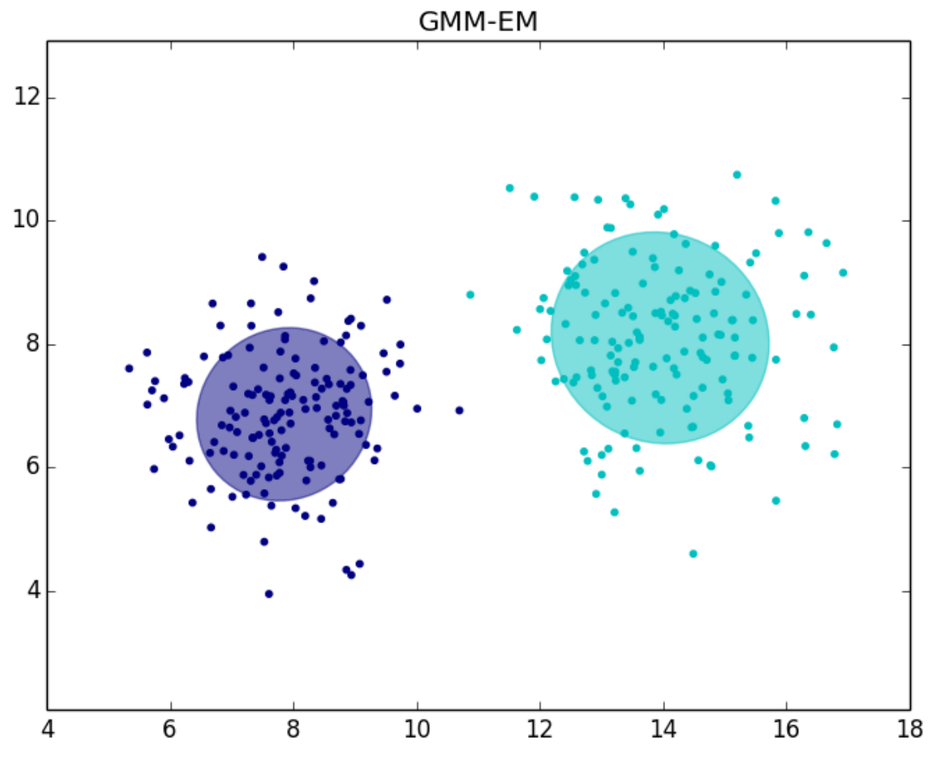
Question 2
Implement the expectation maximization (EM) algorithm for estimating the pdf parameters of 2-D GMMs from samples

In question 2, the most important thing is to build a EM-algorithm function. There is a powerful python module about machine learning called scikit.learn which contains a lot of useful machine learning algorithms including EM-algorithm. When I tried EM-algorithm in this module, the performance is quite nice and easy to use, so I used the module directly, but I also learned the EM-algorithm steps in this function exactly. The algorithm is referred in the source code.

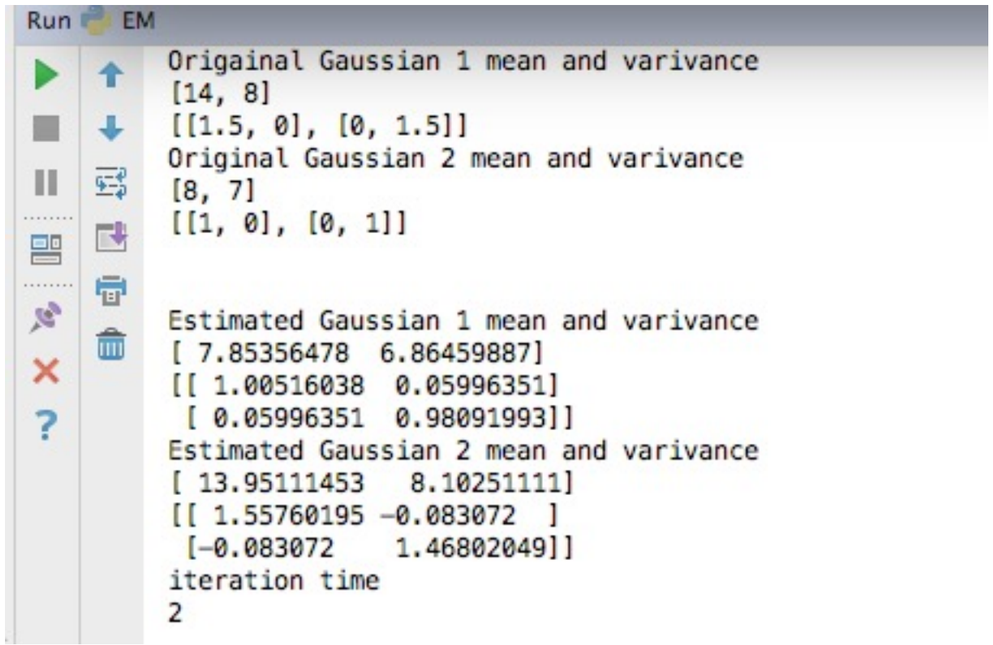
In EM-algorithm, at first we assumes random components and computes for each point a probability of being generated by each component of the model. Then, one tweaks the parameters to maximize the likelihood of the data given those assignments. Repeating this process is guaranteed to always converge to a local optimum. The function(method) called mixture.GaussianMixture.fit is responsible for the work. Inside the function, the method fit the model `n_init` times and set the parameters with which the model has the largest likelihood or lower bound. Within each trial, the method iterates between E-step and M-step for `max_iter` times until the change of likelihood or lower bound is less than `tol`, otherwise, a `ConvergenceWarning` is raised. Outside the function, I also wrote a

function to show the result of clustering and plot an ellipse to show the Gaussian component.





According to comparing the original data which haven't been clustered and the clustered data in different two color, we could clearly see the separation of the data is successful. And I can also get the estimated parameters including mean values and the variance from the fit function. by comparison of the estimated parameters and the original parameters, the result is also very close. And according to the returning iteration times which is only 2, I can conclude that if the two component is well separated, it takes less time and iteration times for clustering.
The printing result is shown below.



Question 3

Compare the quality and speed your GMM-EM estimation on 300 samples of different GMM distributions (e.g. spherical vs ellipsoidal covariance, close vs well-separated subpopulations).

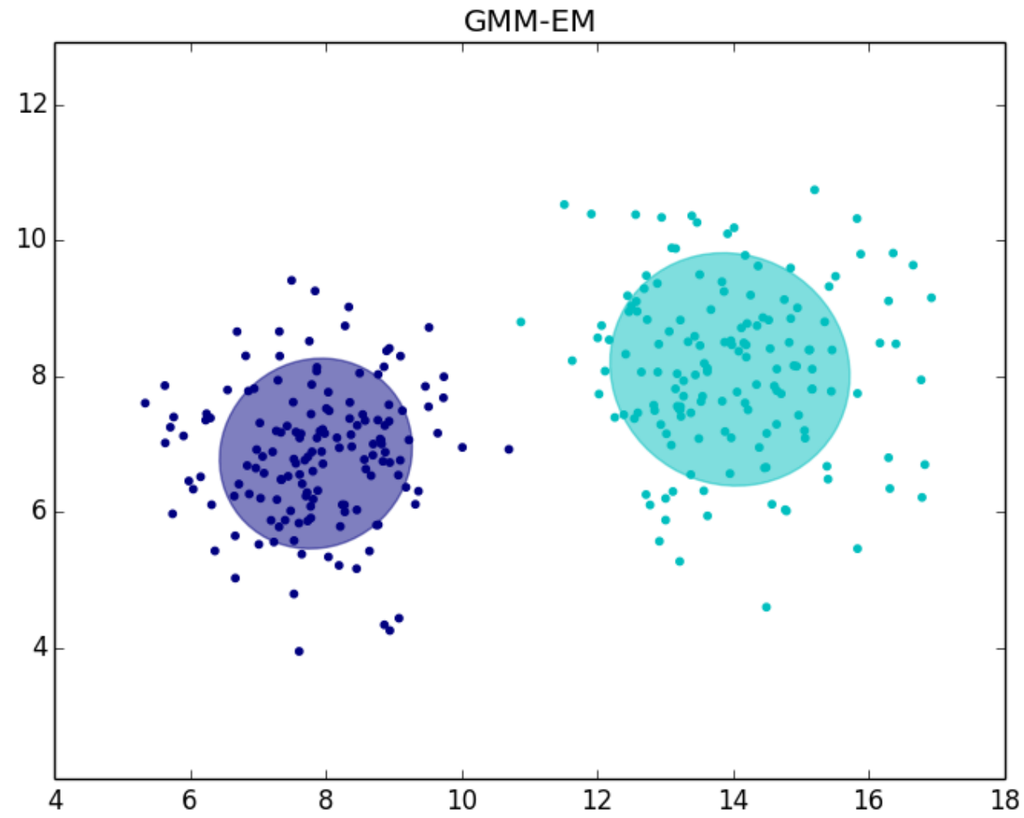
In question 3, the only difference is that we need to change the original mean value and variance value to generate spherical and ellipsoidal GMM, close vs well-separated GMM.

For example, if we want to generate spherical GMM, we need covariance which is close to zero, in another word, comparatively smaller than the variance. And if we want to generate well separated GMM, we need the mean values of the two components to be closer.

I also used a time.clock function to record the real time takes in EM-algorithm as well as the iteration time.

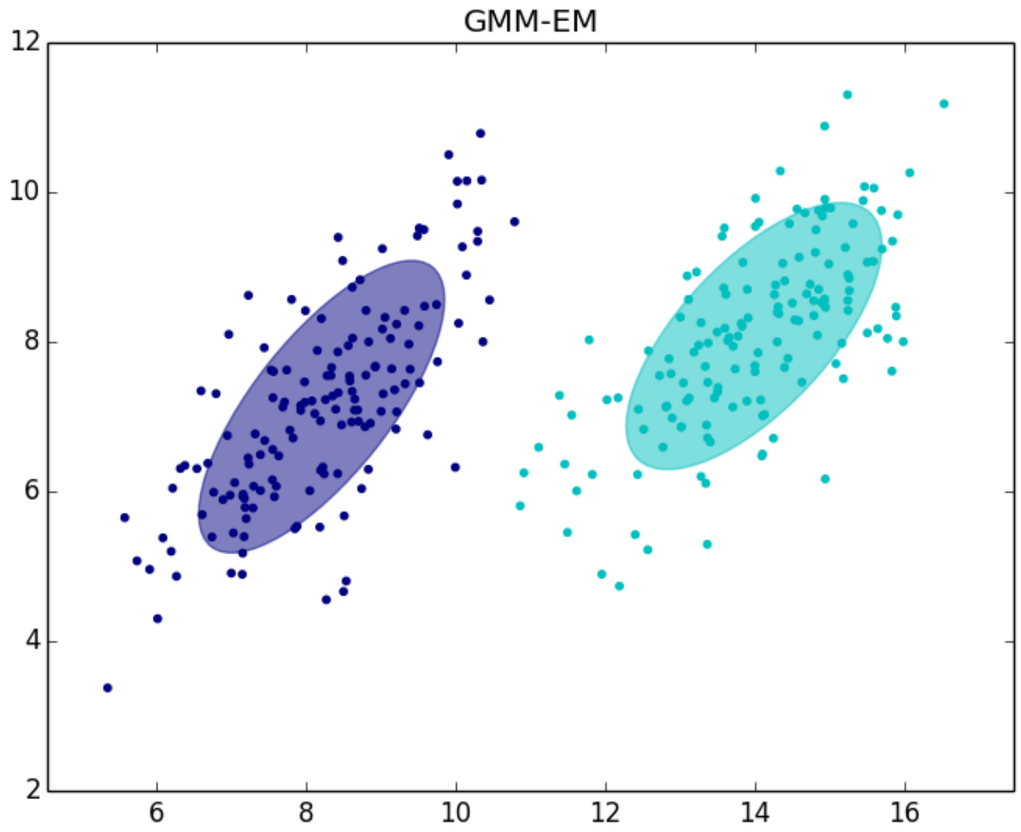
The result is shown bellow

```
Origainal Gaussian 1 mean and varivance
[14, 8]
[[1.5, 0], [0, 1.5]]
Original Gaussian 2 mean and varivance
[8, 7]
[[1, 0], [0, 1]]


Estimated Gaussian 1 mean and varivance
[ 7.85356478  6.86459887]
[[ 1.00516038  0.05996351]
 [ 0.05996351  0.98091993]]
Estimated Gaussian 2 mean and varivance
[ 13.95111453   8.10251111]
[[ 1.55760195 -0.083072  ]
 [-0.083072    1.46802049]]
iteration time
2
exact time
0.188628
```



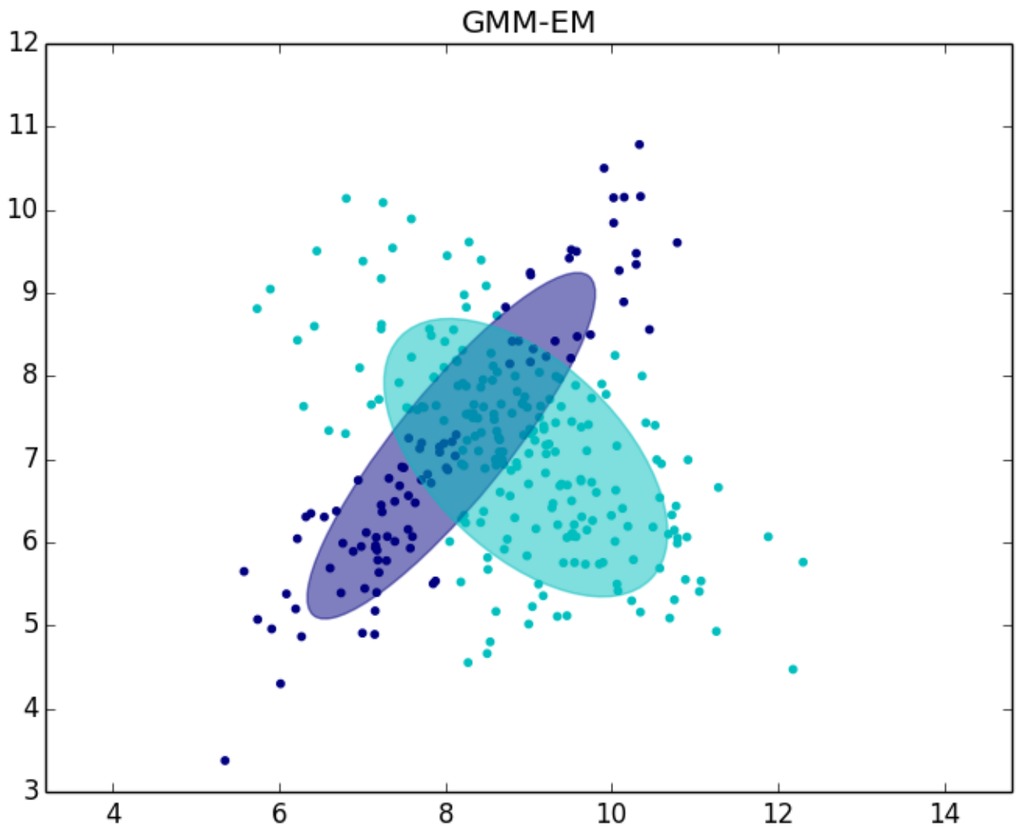
Example 1: spherical

```
Origainal Gaussian 1 mean and varivance
[14, 8]
[[1.5, 1], [1, 1.5]]
Original Gaussian 2 mean and varivance
[8, 7]
[[1.2, 1], [1.2, 2]]


Estimated Gaussian 1 mean and varivance
[ 8.20922899  7.13313833]
[[ 1.34830231  1.17580044]
 [ 1.17580044  1.90693611]]
Estimated Gaussian 2 mean and varivance
[ 13.98846541   8.07716074]
[[ 1.45871432  1.03232278]
 [ 1.03232278  1.58878756]]
iteration time
4
exact time
0.378356
```



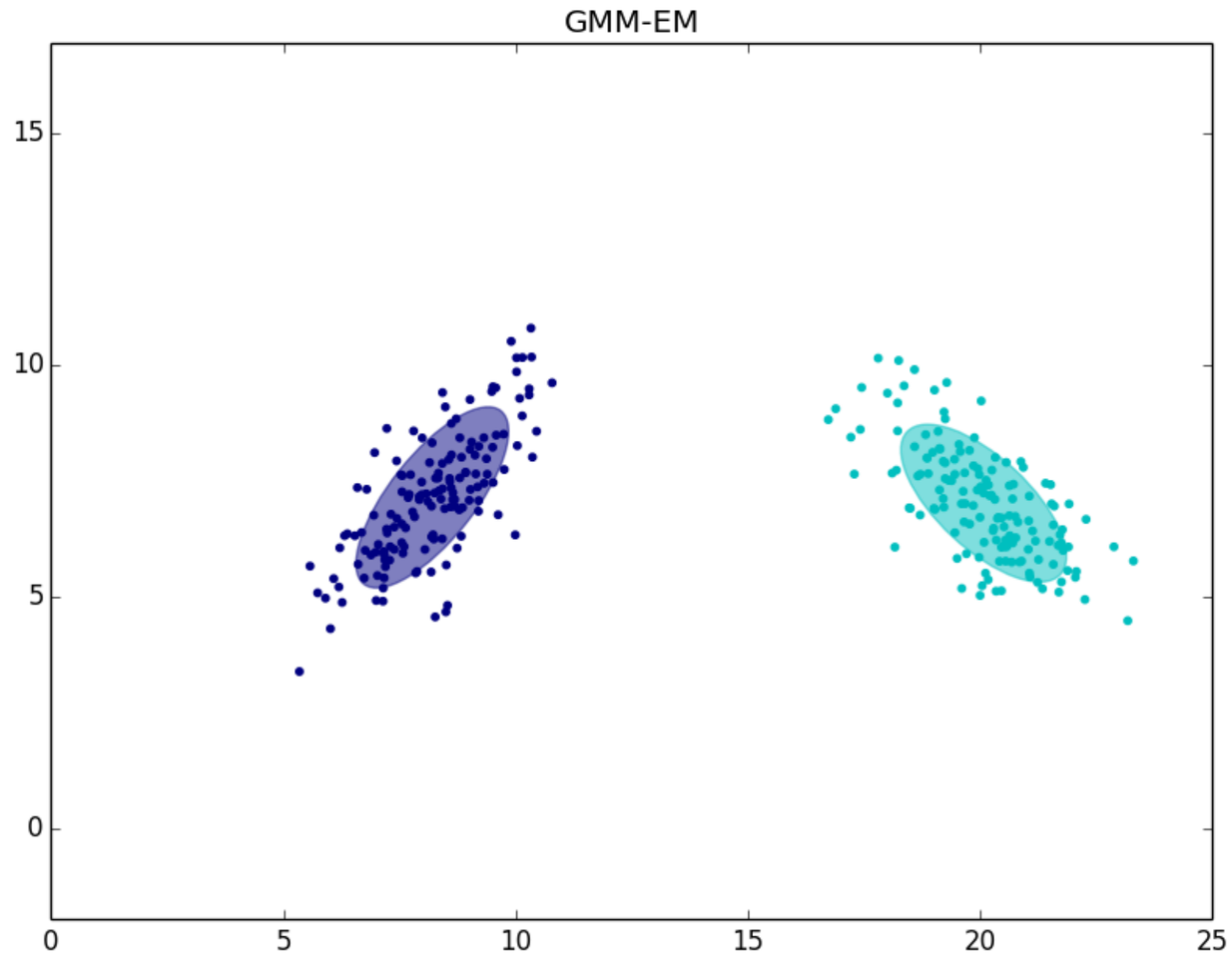
Example 2: ellipsoidal

```
Origainal Gaussian 1 mean and varivance
[9, 7]
[[1.5, -1], [-1, 1.5]]
Original Gaussian 2 mean and varivance
[8, 7]
[[1.2, 1], [1.2, 2]]


Estimated Gaussian 1 mean and varivance
[ 8.06959387  7.16182754]
[[ 1.50071551  1.58272227]
 [ 1.58272227  2.1667434 ]]
Estimated Gaussian 2 mean and varivance
[ 8.96711973  7.01868878]
[[ 1.45389774 -0.76919164]
 [-0.76919164  1.39837071]]
iteration time
57
exact time
1.785903
```



Example 3: close

```
Origainal Gaussian 1 mean and varivance
[20, 7]
[[1.5, -1], [-1, 1.5]]
Original Gaussian 2 mean and varivance
[8, 7]
[[1.2, 1], [1.2, 2]]


Estimated Gaussian 1 mean and varivance
[ 8.21034368  7.13230374]
[[ 1.34823407  1.1733971 ]
 [ 1.1733971   1.90609873]]
Estimated Gaussian 2 mean and varivance
[ 20.07876117   7.00796085]
[[ 1.58706486 -1.02640794]
 [-1.02640794  1.44464668]]
iteration time
1
exact time
0.426495
```



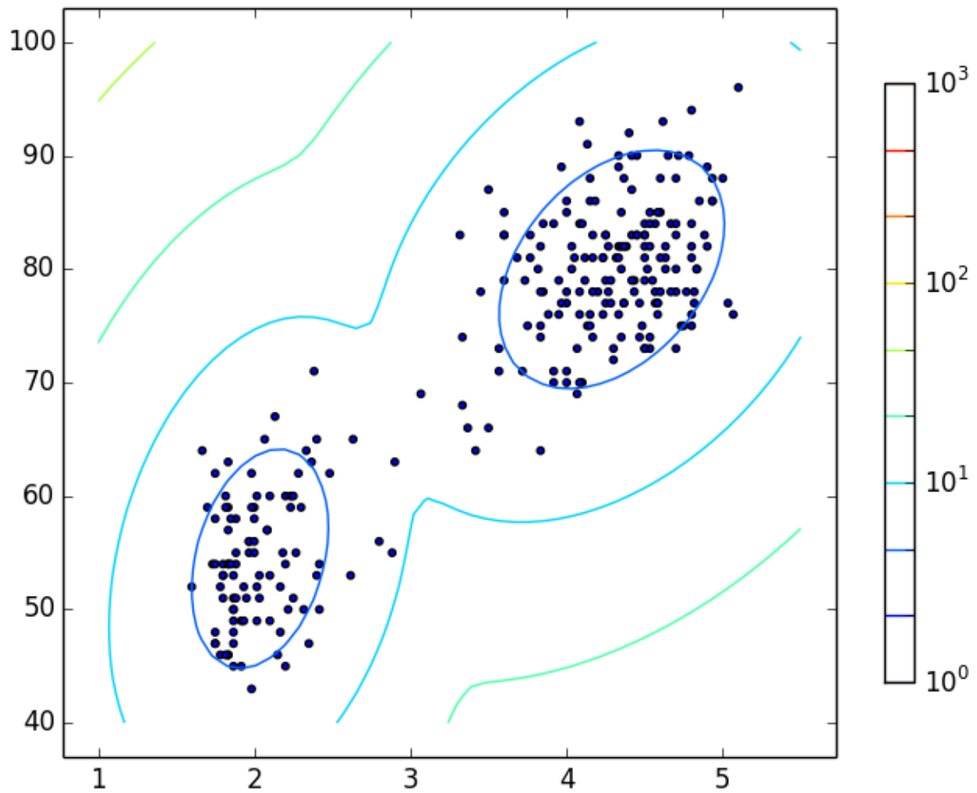
Example 4: well-separated

By comparing example 1 and 2, we can conclude that ellipsoidal covariance takes more time than the spherical, and the quality (accuracy of estimated parameters) is also worse than the spherical.
By comparing example 3 and 4, we can conclude that close takes more time than the well-separated, and the quality (accuracy of estimated parameters) is also worse than the well-separated.

Problem #2

Apply your GMM-EM algorithm to fit the “old faithful” data set to a GMM pdf. Plot a contour plot of your final GMM pdf. Overlay the contour plot with a scatterplot of the data set. How would you use the GMM pdf estimates to cluster the data?

Firstly we need to read the data from the old-faithful.txt by using function np.genfromtxt().In the GaussianMixture Class used in problem #1 question 2, there is a method(function) called score_sample(X), which can compute the weighted log probabilities for each GMM sample. Let Z equals to the negative returning value of score.samplex(), we could draw a contour plot regarding to X axle, Y axle and Z(Negative log probability).Here Z represent the negative predicted score of a point, and if any point whose value of Z is smaller than z, this means that the real log probability of this point belong to a cluster is bigger than z. The result is shown as bellow.



Based on this contour plot, we can cluster the data according to the point and the color of the contour line. The point in the dark blue line belong to the adjoining Gaussian distribution in a very big chance, the point in the light blue belong to the adjoining Gaussian distribution possibly, while the point outside the light blue line is almost not possible to belong to the Gaussian distribution.

Source Code

Problem 1:

Question 1-GMM_RNG.py

```
import itertools
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import mixture

def gmmrng(m1,cov1,num1,m2,cov2,num2):
    np.random.seed(0) # fixed number call the same random number each time

   # m1, cov1,num1 = [14, 8], [[2, -1], [-1, 2]],150  ## first gaussian
    data1 = np.random.multivariate_normal(m1, cov1, num1)

   # m2, cov2,num2 = [4, 7], [[1, 0.1], [0.1, 1]],150  ## second gaussian
    data2 = np.random.multivariate_normal(m2, cov2, num2)

    X = np.vstack((data1, data2)) #vertivally combination gaussian 1 and 2
    np.random.shuffle(X) #reorder
    return X

def pltgmm(Q):
    plt.figure()
    for i in range(0,len(Q),1):
        plt.plot(Q[i, 0], Q[i, 1],'ro')
    plt.axis('equal')
    plt.title('Gaussian Mixture')
    plt.show()


Q=gmmrng(m1=[14, 8],cov1=[[1.5, -1], [-1, 1.5]],num1 =150,
         m2=[8, 7], cov2=[[1, 0.1], [0.1, 1]], num2 = 150
        )
pltgmm(Q)
```

Question 2-EM.py

```
from GMM_RNG import *

color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold','darkorange'])

def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(1, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], 10, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.title(title)
    plt.axis('equal')

m1,cov1,num1=[14, 8],[[1.5,0],[0, 1.5]],150
m2,cov2,num2=[8, 7] ,[[1, 0], [0, 1]], 150
X=gmmrng(m1,cov1,num1,m2,cov2,num2)
pltgmm(X)
# Fit a Gaussian mixture with EM using 2 components
gmm = mixture.GaussianMixture(n_components=2, covariance_type='full',tol=1e-5).fit(X)
plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_, 0,'GMM-EM')
plt.show()

print ('Origainal Gaussian 1 mean and varivance')
print m1
print cov1
print ('Original Gaussian 2 mean and varivance')
```

```
print m2
print cov2
print ('\n')
print ('Estimated Gaussian 1 mean and varivance')
print gmm.means_[0]
print gmm.covariances_[0]
print ('Estimated Gaussian 2 mean and varivance')
print gmm.means_[1]
print gmm.covariances_[1]
print ('iteration time')
print gmm.n_iter_
```

Question 3-EM_Compare.py

```
from GMM_RNG import *
import time

color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold','darkorange'])

def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(1, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], 10, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi  # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.title(title)
```

```
    plt.axis('equal')

m1,cov1,num1=[9, 7],[[1.5,-1],[-1, 1.5]],150
m2,cov2,num2=[8, 7] ,[[1.2, 1], [1.2, 2]], 150
X=gmmrng(m1,cov1,num1,m2,cov2,num2)
#pltgmm(X)
t0 = time.clock()
# Fit a Gaussian mixture with EM using 2 components
gmm = mixture.GaussianMixture(n_components=2, covariance_type='full',tol=1e-5).fit(X)
plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_, 0,'GMM-EM')
plt.show()
t = time.clock()-t0

print ('Origainal Gaussian 1 mean and varivance')
print m1
print cov1
print ('Original Gaussian 2 mean and varivance')
print m2
print cov2
print ('\n')
print ('Estimated Gaussian 1 mean and varivance')
print gmm.means_[0]
print gmm.covariances_[0]
print ('Estimated Gaussian 2 mean and varivance')
print gmm.means_[1]
print gmm.covariances_[1]
print ('iteration time')
print gmm.n_iter_
print ('exact time')
print t
```

Problem 2:

```
from itertools import islice
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl
import itertools
from sklearn import mixture
from matplotlib.colors import LogNorm
```

```
with open('old-faithful.txt') as lines:
    array = np.genfromtxt(islice(lines, 26, 298))

data = np.delete(array,0,1)

color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold',
                              'darkorange'])

def plot_results(X, Y_, means, covariances, index, title):

    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], s=28, color=color)
    plt.title(title)

# fit a Gaussian Mixture Model with two components
clf = mixture.GaussianMixture(n_components=2, covariance_type='full')
clf.fit(data)

# display predicted scores by the model as a contour plot
x = np.linspace(1., 5.5)
y = np.linspace(40., 100.)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
Z = -clf.score_samples(XX)
Z = Z.reshape(X.shape)

CS = plt.contour(X, Y, Z, norm=LogNorm(vmin=1.0, vmax=1000.0),
                 levels=np.logspace(0, 3, 10))
CB = plt.colorbar(CS, shrink=0.8, extend='both')
plt.scatter(data[:, 0], data[:, 1], 10)

plt.show()
```

```
plt.figure(2)
 #Fit a Gaussian mixture with EM using 2 components
gmm = mixture.GaussianMixture(n_components=2, covariance_type='full',tol=1e-5).fit(data)
plot_results(data, gmm.predict(data), gmm.means_, gmm.covariances_, 0,
             'Gaussian Mixture')
plt.show()
```