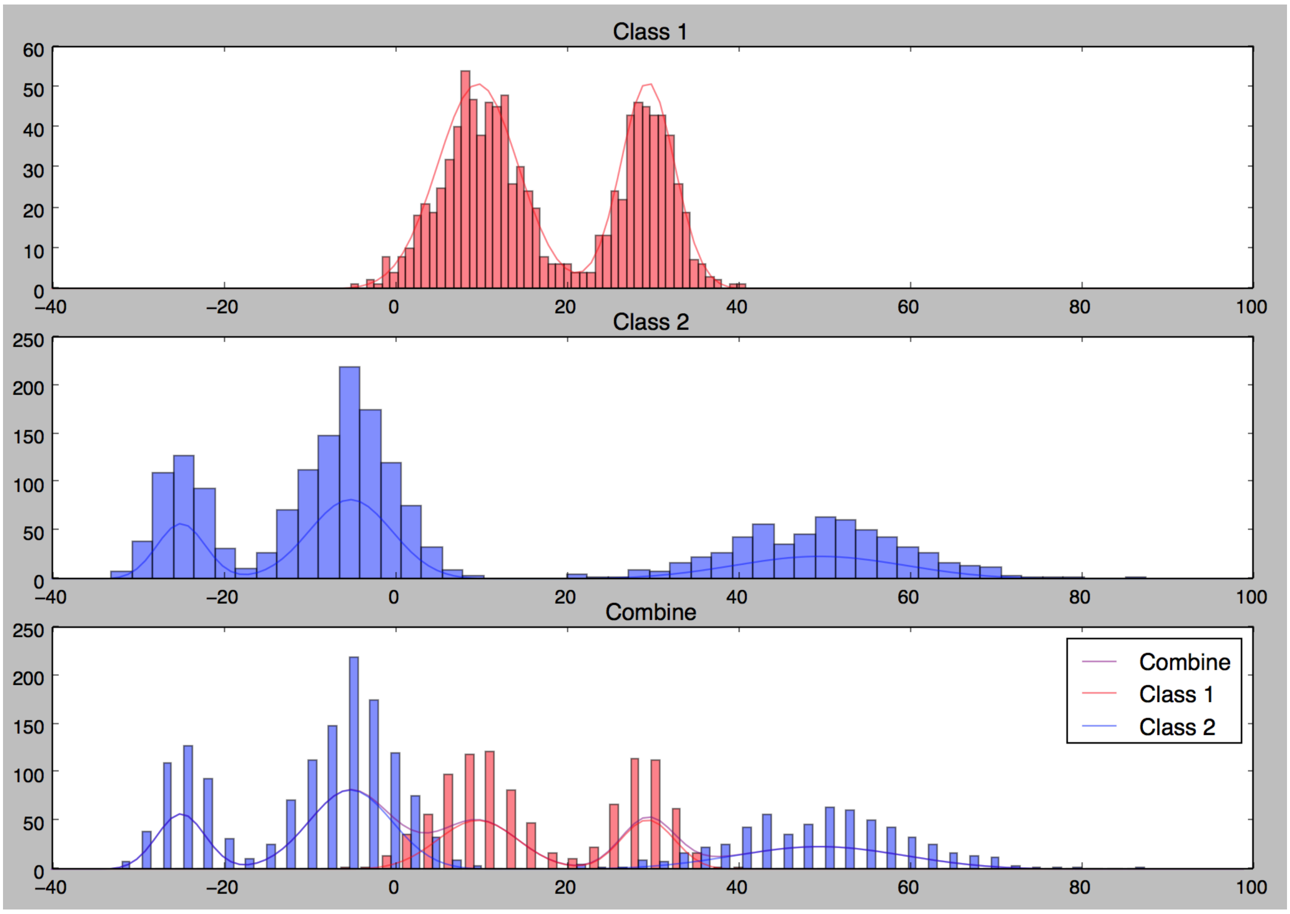
Zhiping Xiu

1):

Please see the implementation of gmm_est in gem_est.py

2):

The plot as required:



I printed the change of weight, mu, sigma² and log likelihood as followed:
The last column is log likelihood, as you can see, it stay steady at end of the iterations, so **I believe the result converge.**

➜ Zhiping_Xiu_hw6 python gmm_est.py 'gmm_train.csv'

[0.59340834 0.40659166] [9.68230554 29.51084123] [20.85982026 10.18902154]
-3459.19504772
[0.59609412 0.40390588] [9.74298059 29.55314588] [21.57960207   9.98529276]
-3458.76592202
[0.59704792 0.40295208] [9.76250259 29.57111185] [21.78993071   9.8624295]
-3458.70598542
[0.59741823 0.40258177] [9.77004432 29.57814046] [21.87062523   9.81404105]
-3458.69689282
[0.59756262 0.40243738] [9.7729979 29.5808621] [21.90240428   9.79546834]
-3458.69551759
[0.59761885 0.40238115] [9.77415103 29.58191725] [21.91485489   9.7883052]
-3458.69530976
[0.59764072 0.40235928] [9.77460012 29.58232688] [21.91971142   9.7855307]
-3458.69527835
[0.59764922 0.40235078] [9.77477481 29.58248602] [21.92160181   9.78445384]
-3458.69527361

[0.59765253 0.40234747] [9.77484273 29.58254787] [21.922337   9.78403552]
-3458.69527289
[0.59765381 0.40234619] [9.77486914 29.5825719] [21.92262282   9.78387296]
-3458.69527278
[0.59765431 0.40234569] [9.7748794   29.58258124] [21.92273392   9.78380978]
-3458.69527277
[0.59765451 0.40234549] [9.77488339 29.58258487] [21.9227771   9.78378523]
-3458.69527276
[0.59765458 0.40234542] [9.77488494 29.58258629] [21.92279389   9.78377568]
-3458.69527276
[0.59765461 0.40234539] [9.77488554 29.58258683] [21.92280041   9.78377197]
-3458.69527276
[0.59765462 0.40234538] [9.77488577 29.58258705] [21.92280295   9.78377053]
-3458.69527276
[0.59765463 0.40234537] [9.77488587 29.58258713] [21.92280394   9.78376997]
-3458.69527276
[0.59765463 0.40234537] [9.7748859   29.58258716] [21.92280432   9.78376975]
-3458.69527276
[0.59765463 0.40234537] [9.77488591 29.58258717] [21.92280447   9.78376967]
-3458.69527276
[0.59765463 0.40234537] [9.77488592 29.58258718] [21.92280453   9.78376963]
-3458.69527276
[0.59765463 0.40234537] [9.77488592 29.58258718] [21.92280455   9.78376962]
-3458.69527276

[0.20419658 0.49830344 0.29749998] [-24.82193248 -5.03862005 49.62553629] [7.83890831
22.96965594 99.97980953] -8246.15106118
[0.20375069 0.49874157 0.29750774] [-24.82049123 -5.05707661 49.62441463] [7.9543965
23.28002361 100.02549392] -8246.06992297
[0.20367981 0.49881264 0.29750755] [-24.82185015 -5.05932461 49.62444065]
[7.95216929   23.31239467 100.02449517] -8246.06907092
[0.20365964 0.49883283 0.29750752] [-24.82243492 -5.05988441 49.62444379]
[7.94913011   23.3193801   100.02437329] -8246.06899649
[0.20365294 0.49883954 0.29750752] [-24.8226425   -5.06006489 49.62444443] [7.94795933
23.32154797 100.02434875] -8246.06898779
[0.20365065 0.49884183 0.29750752] [-24.82271418 -5.06012623 49.62444462]
[7.94755014   23.32227988 100.02434131] -8246.06898676
[0.20364987 0.49884261 0.29750752] [-24.82273883 -5.06014727 49.62444469] [7.9474092
23.32253064 100.02433881] -8246.06898664
[0.2036496   0.49884288 0.29750752] [-24.8227473   -5.0601545   49.62444471] [7.94736076
23.32261675 100.02433795] -8246.06898663
[0.20364951 0.49884298 0.29750752] [-24.82275021 -5.06015698 49.62444472]
[7.94734412   23.32264633 100.02433766] -8246.06898663
[0.20364948 0.49884301 0.29750752] [-24.82275121 -5.06015784 49.62444472] [7.9473384
23.3226565   100.02433756] -8246.06898663
[0.20364946 0.49884302 0.29750752] [-24.82275155 -5.06015813 49.62444472]
[7.94733644   23.32265999 100.02433752] -8246.06898663
[0.20364946 0.49884302 0.29750752] [-24.82275167 -5.06015823 49.62444472]
[7.94733576   23.32266119 100.02433751] -8246.06898663
[0.20364946 0.49884302 0.29750752] [-24.82275171 -5.06015827 49.62444472]
[7.94733553   23.3226616   100.02433751] -8246.06898663
[0.20364946 0.49884302 0.29750752] [-24.82275172 -5.06015828 49.62444472]
[7.94733545   23.32266174 100.02433751] -8246.06898663
[0.20364946 0.49884302 0.29750752] [-24.82275173 -5.06015828 49.62444472]
[7.94733542   23.32266179 100.0243375] -8246.06898663

```
[ 0.20364946  0.49884302  0.29750752] [-24.82275173  -5.06015828  49.62444472]
[   7.94733541   23.32266181  100.0243375 ] -8246.06898663
[ 0.20364946  0.49884302  0.29750752] [-24.82275173  -5.06015828  49.62444472]
[   7.94733541   23.32266181  100.0243375 ] -8246.06898663
[ 0.20364946  0.49884302  0.29750752] [-24.82275173  -5.06015828  49.62444472]
[   7.94733541   23.32266181  100.0243375 ] -8246.06898663
[ 0.20364946  0.49884302  0.29750752] [-24.82275173  -5.06015828  49.62444472]
[   7.94733541   23.32266181  100.0243375 ] -8246.06898663
[ 0.20364946  0.49884302  0.29750752] [-24.82275173  -5.06015828  49.62444472]
[   7.94733541   23.32266181  100.0243375 ] -8246.06898663
```

You can test the program using the same command:
The plot is named as: likelihood_classes.png

```
➜  Zhiping_Xiu_hw6 python gmm_est.py 'gmm_train.csv'
Class 1
mu = [  9.77488592  29.58258718]
sigma^2 = [ 21.92280455   9.78376962]
w = [ 0.59765463  0.40234537]

Class 2
mu = [-24.82275173  -5.06015828  49.62444472]
sigma^2 = [   7.94733541   23.32266181  100.0243375 ]
w = [ 0.20364946  0.49884302  0.29750752]
```

The initial parameters is chosen based on visualization of the chart, which is, this initial value is also used in following questions:

```
init_mu1 = np.array([10., 30.])
init_sigmasq1 = np.array([8., 6.])
init_wt1 = np.array([.6, .4])

init_mu2 = np.array([-25., -5., 50.])
init_sigmasq2 = np.array([3., 10., 20.])
init_wt2 = np.array([.2, .5, .3])
```

Zhiping Xiu

3):

The program as requested:
Took gmm_train.csv to get mus, sigma²s and weights, then test on gmm_test.csv, and print the output.

Please see more in my code

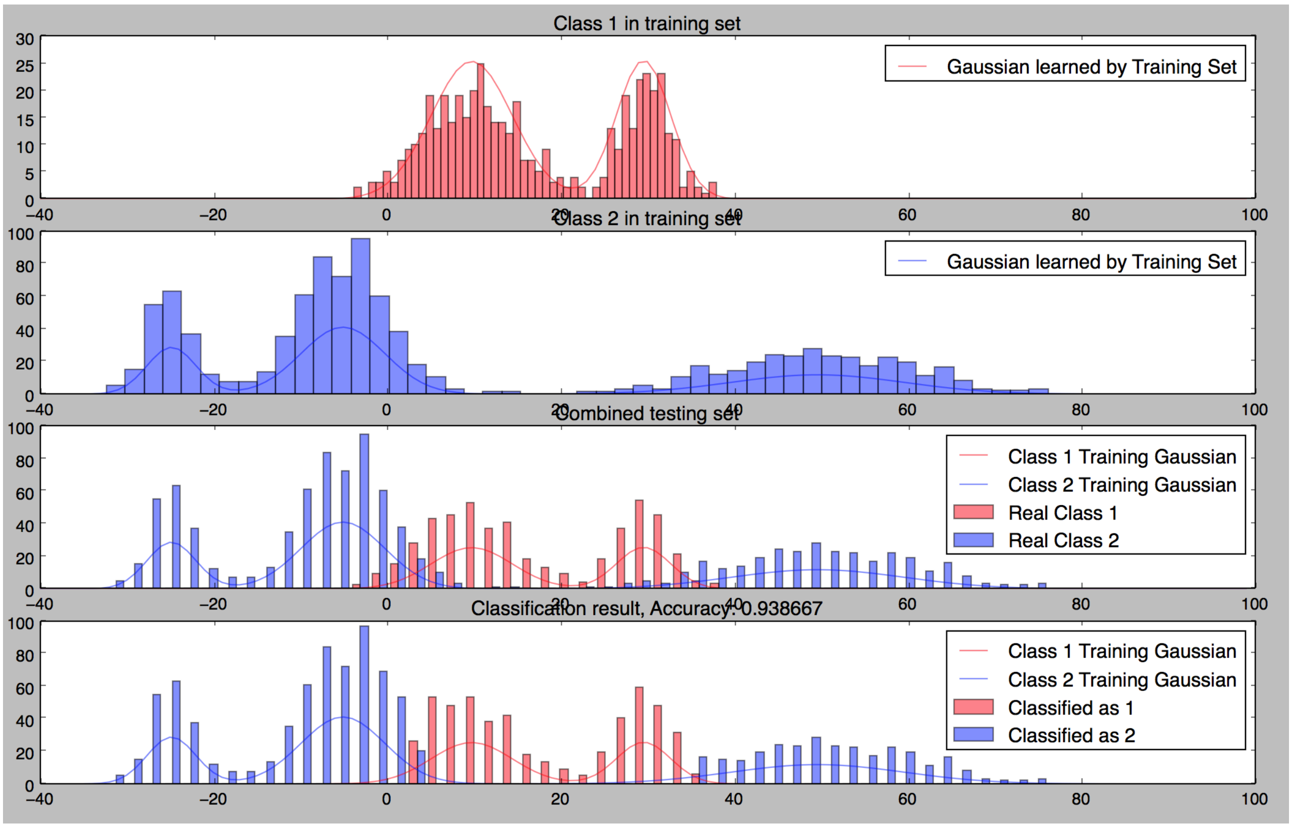
```
~ python gmm_classify.py 'gmm_test.csv'
Class 1
[ 13.178   12.785   5.8304   4.2179   6.1641  30.041   27.4     10.927
  31.627   13.667   34.575   4.1116  31.246   12.766   15.523   15.414
  26.287   30.779   14.618   9.6703  26.938   13.229   29.459   23.816
…

Class 2
[  1.36160000e+00  -1.82070000e+00   3.78400000e+01   2.44760000e+00
  -8.12120000e-01   3.71390000e+01   3.61260000e+01   1.20040000e+00
   4.80370000e-01   2.35520000e+00  -8.76680000e-01  -1.49610000e+00
   1.91650000e-01  -3.21510000e+00   1.56510000e+00   5.15000000e-01
…
```

The way Class 2 is formatted is a little weird, I guess it's because Class 2 is smaller so the np.array decided to print it this way?

4):

The plot as required:
I printed the accuracy on the chart, which is 0.938667



5 & 6):

The fact we already know is:

1: If we have one set of data X, and we know that every data points x in X is generated by one single Gaussian, we will have a closed-form algorithm to calculate that Gaussian.

2: In Gaussian Mixture Model, however, we currently don't have close-form algorithm, we have to use EM algorithm, and EM algorithm can only promise local optimal.

3: The definition of closed-form algorithm is that, we have a method to calculate our answer by just giving the method necessary data, and the method can get the final answer without iteration.

I will answer question 5 and 6 based on those 3 facts:

5):

Based on given info of the question, we can calculate the every Gaussian distributions in our GMM with a closed-form Algorithm, since every data point were coming from a single Gaussian, and we also know which Gaussian.

So, we can calculate mu, sigma for every Gaussians in our GMM.

For weight, we can calculate the weight of each Gaussians by the portion of the data points to the total number of the data points that each Gaussians are responsible for.

For example, if we know a Gaussian G in our GMM is responsible for 10 data points, and we know we have 100 data points in total, we can assign 0.1 as weight to Gaussian G, according to the definition of the closed-form method, the calculation of weights are also close-formed.

6):

Since we don't know the which data points are coming from which Gaussian, we don't have a closed-form method to solve this problem.

According to the fact 2, we have to use EM to do hill climbing.

UNLESS, when $K = 1$, we can use closed form solution again, just using fact 1 and set weight for the single gaussian = 1.