Part 1

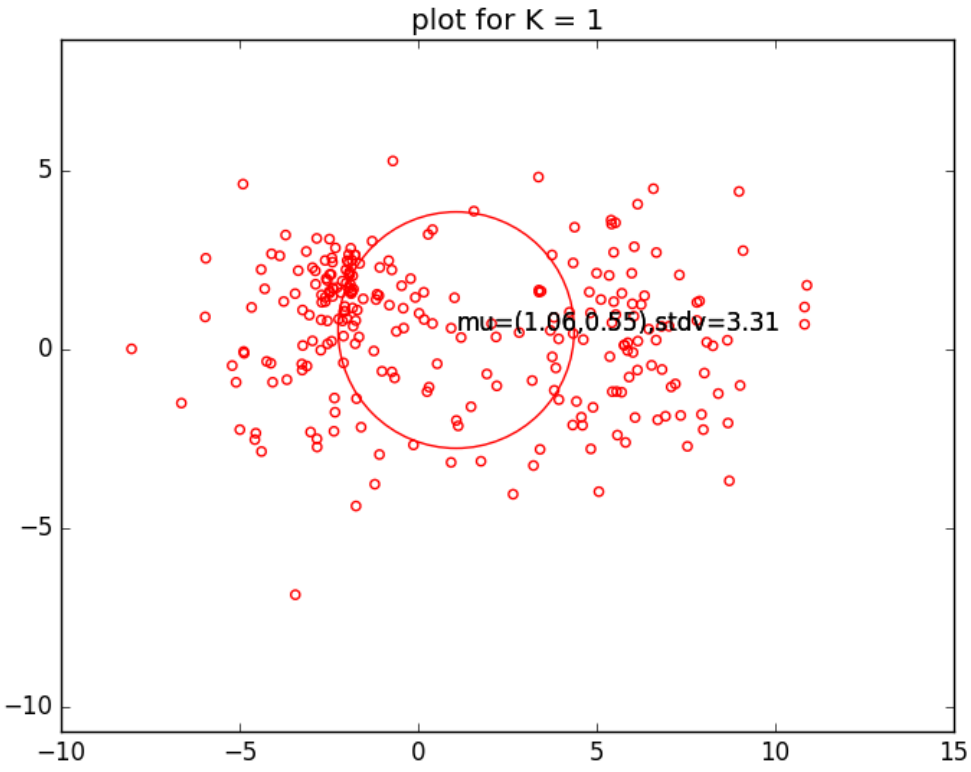
a.

Code:

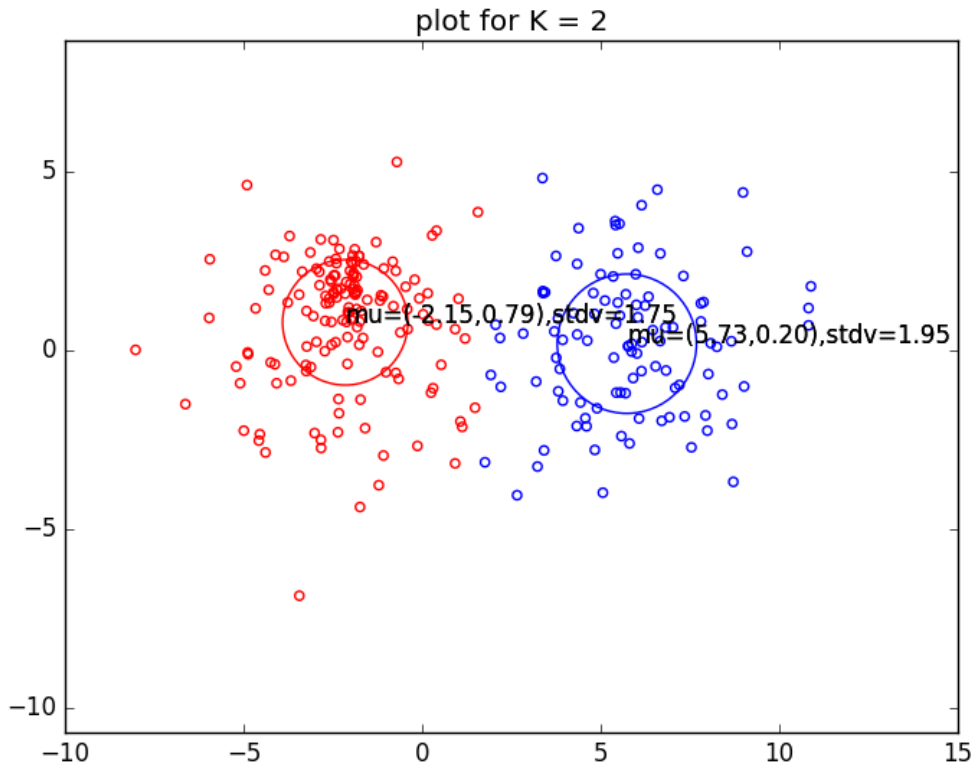
```
X = p3.readData('toy_data.txt')

for K in [1,2,3,4]:
      print "Calculating for K=", K
      (Mu, P, Var) = p3.init(X,K)
      (Mu,P,Var,post) = p3.kMeans(X, K, Mu, P, Var)
      p3.plot2D(X,K,Mu,P,Var,post,"plot for K = %d" % K)
```

Plots:

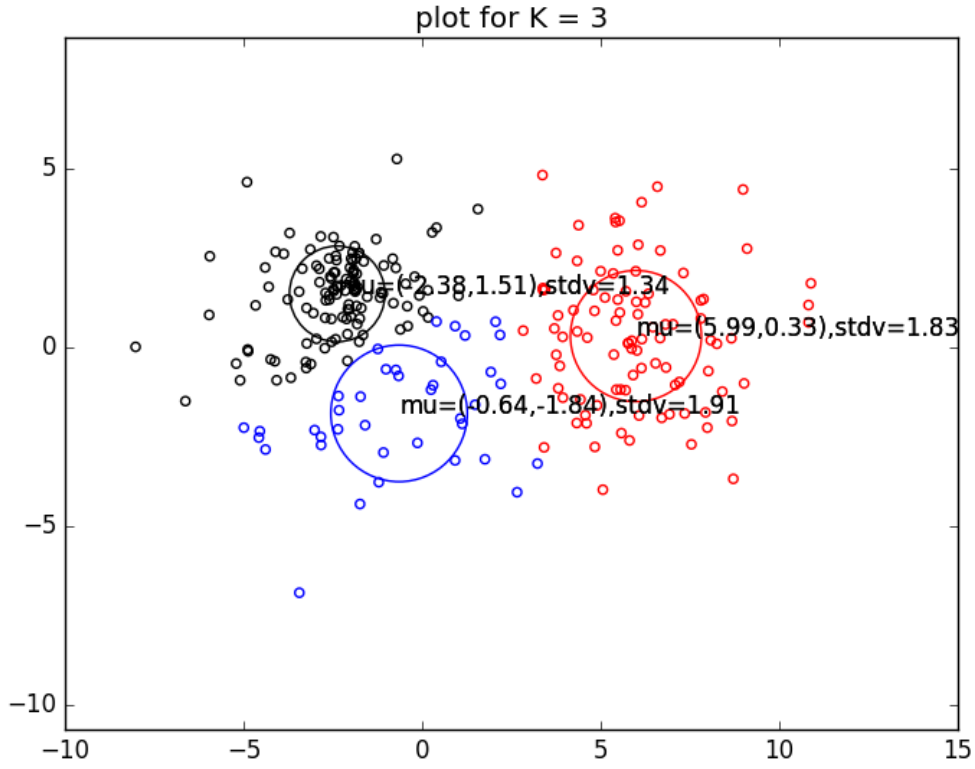
Best solution for K=1:
(LL = -5462.29)



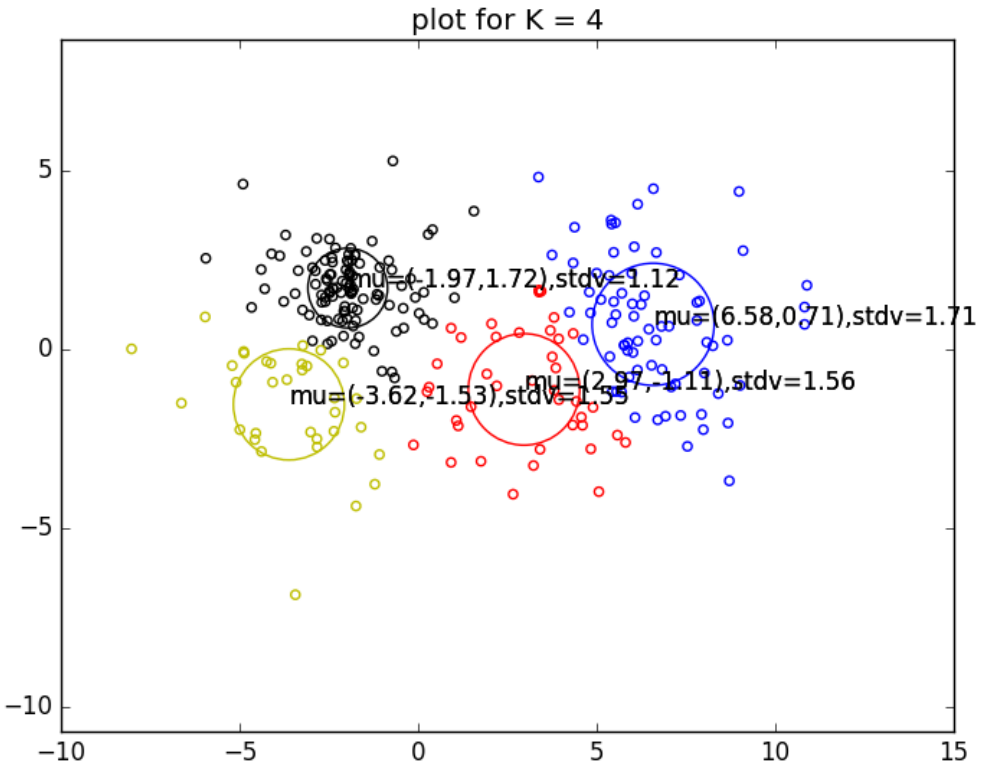
Best solution for K=2:
(LL = -1684.90)



Best solution for K=3:
(LL = -1329.66)



Best solution for K=4:
(LL = -1035.49)



b.

Code for E-Step:

```
def Estep(X,K,Mu,P,Var):
    n,d = np.shape(X) # n data points of dimension d
    post = np.zeros((n,K)) # posterior probabilities to compute
    LL = 0.0    # the LogLikelihood

    for u in xrange(len(X)):
        for j in xrange(K):
            post[u,j] = P[j]*Gaussian(X[u,:],Mu[j,:],Var[j])
        scaling_factor = np.sum(post[u,:])
        post[u,:] = post[u,:]/scaling_factor
        LL += np.log(scaling_factor)

    return (post,LL)
```

Code for M-Step:

```
def Mstep(X,K,Mu,P,Var,post):
    n,d = np.shape(X) # n data points of dimension d

    #code n_hat
    n_hat = np.sum(post, axis = 0).T

    #code pi (mixture proportions)
    P = n_hat/n

    #code mu
    Mu = np.divide(np.dot(post.T, X).T, n_hat).T

    for j in xrange (K):
        Var[j] = np.dot(np.linalg.norm(np.subtract(X, Mu[j,:]),
axis=1)**2, post[:,j]) / (d*n_hat[j])

    return (Mu,P,Var)
```

Code for mixGauss:

```
def mixGauss(X,K,Mu,P,Var):
    n,d = np.shape(X) # n data points of dimension d
    post = np.zeros((n,K)) # posterior probabilities

    prevLL = -1e18
    curLL = -1e17
    LL = []
    while True:
        prevLL = curLL
        (post,curLL) = Estep(X,K,Mu,P,Var)
        LL.append(curLL)
        print curLL
        if (curLL-prevLL) <= 1e-6*np.abs(curLL): break
        (Mu,P,Var) = Mstep(X,K,Mu,P,Var,post)

    return (Mu,P,Var,post,np.array(LL))
```

Test the E-step with this code:

```
K = 3
(Mu, P, Var) = p3.init(X,K,fixedmeans=True)
(Mu,P,Var,post,LL) = p3.mixGauss(X,K,Mu,P,Var)
print LL
```

Returns a log likelihood of -1331.67489, as expected.

c., d.

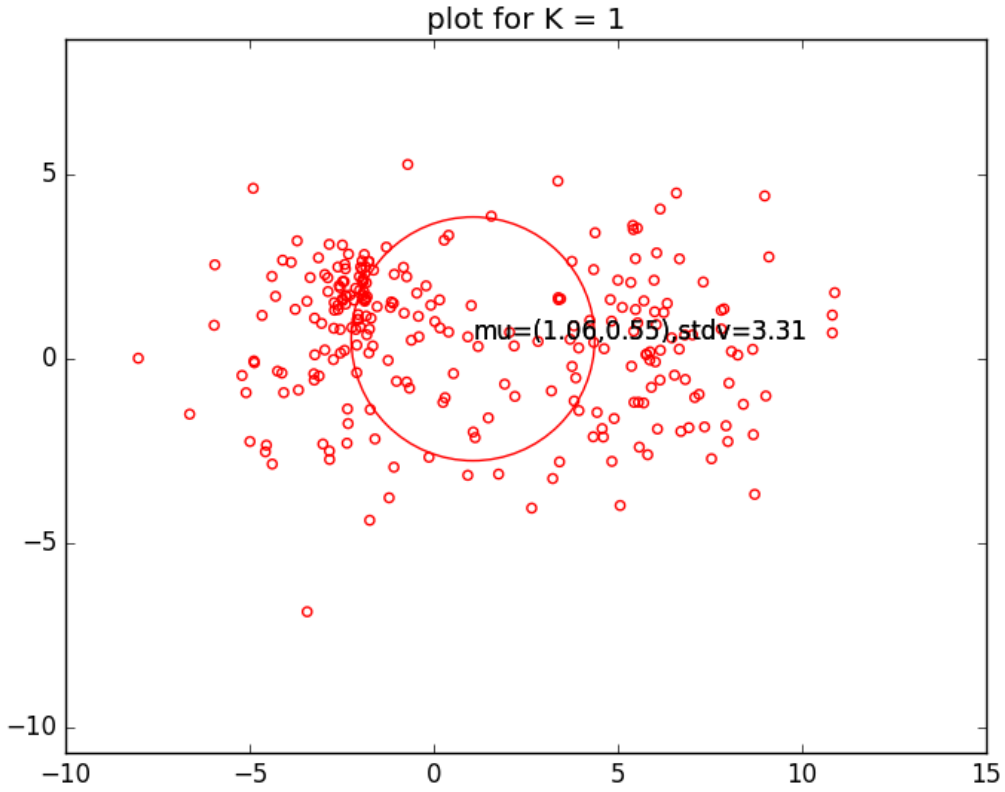
Code:

```
X = p3.readData('toy_data.txt')

for K in [1,2,3,4]:
      print "Calculating for K=", K
      (Mu, P, Var) = p3.init(X,K)
      (Mu,P,Var,post, LL) = p3.mixGauss(X,K,Mu,P,Var)
      for i in xrange(len(LL)-1):
            assert LL[i] >= LL[i+1], "Log-Likelihoods not monotonically
increasing!"
      p3.plot2D(X,K,Mu,P,Var,post,"plot for K = %d" % K)
```

I confirmed that the LL values that my algorithm returns are indeed always monotonically increasing. Running the EM algorithm with the *(Mu, P, Var) = init(X,K,fixedmeans=True)* initialization and *K= 3*, I got the expected values -1331.67489 as the initial LL and then -1138.89248 as the final LL.

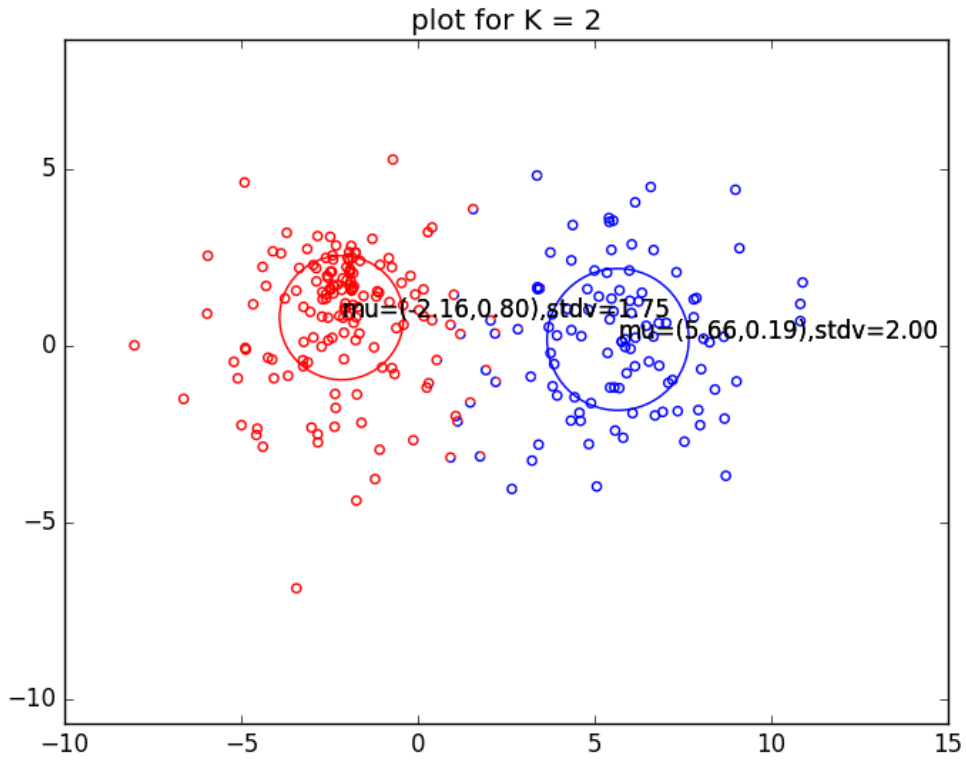
Plots:
For each K, the solutions that achieved the highest log-likelihood is below:

For K=1:
(LL= -1307.22)



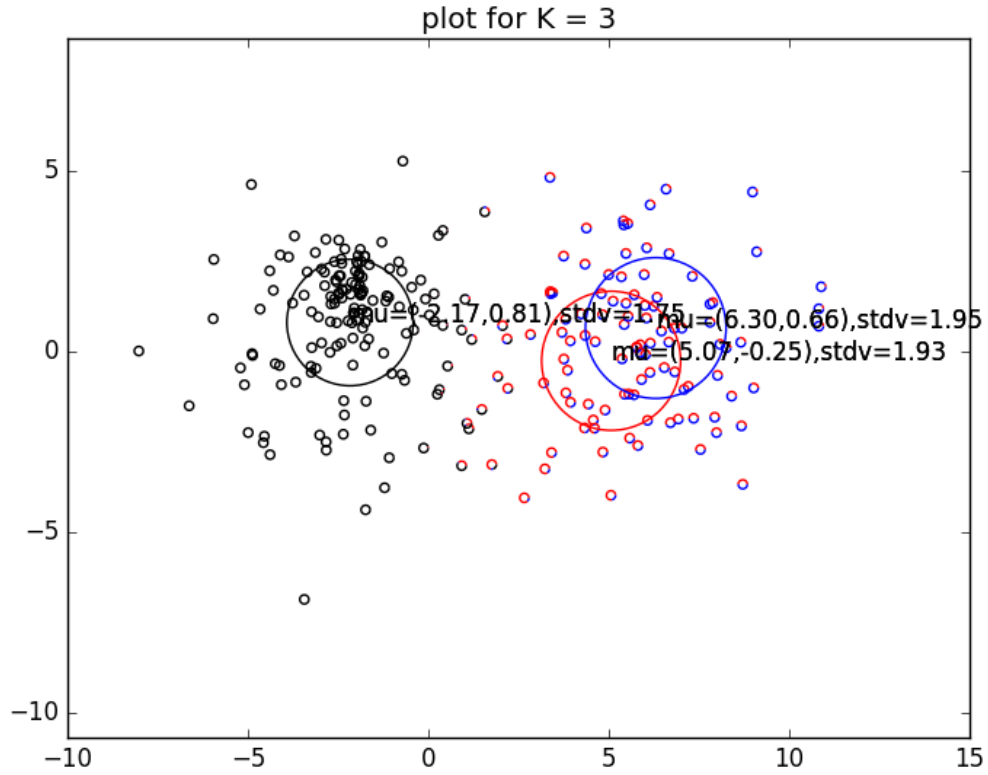
For K=2:

(LL = -1175.71)

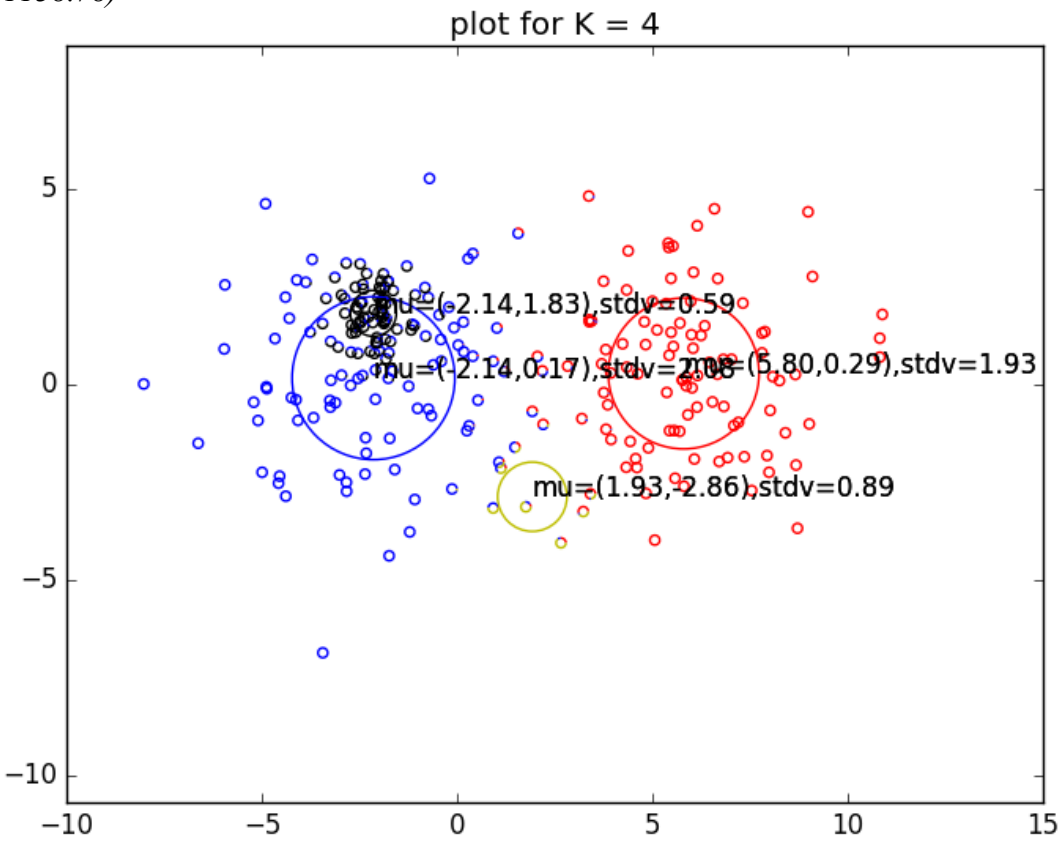


For K=3:

(LL = -1175.35)



For K=4:
(LL = -1138.78)



Comparison of K-means and the mixture solutions:

- They differ for K=3 and K=4.
- In K-means, the clusters do not overlap with each other, while in the EM, we see clusters on top of each other, overlapping. In other words, K-means solutions are hard clusters. More specifically, looking at the points on the plot, we can see one circle having 3 colors in it, representing mixtures from different Gaussians. So, a point can belong to different clusters with different weights. That's a softer assignment compared to K-means.
- EM captures more possible configurations of Gaussian mixtures; it is more comprehensive compared to K-means. K-means is not flexible.

e.
Overall, as we increase K, the log likelihood increases. But more specifically, from K=1 to K=2, we see a big drop in the log-likelihood, and then from K=2 to K=3 we see a small amount of drop and then even a smaller amount from K=3 to K=4. As we increase the number of Gaussians, the gain decreases, *ie. the log-likehood improves less*. In more general terms, the gain from introducing complexity into the model diminishes.

BIC:

- Best **K = 3**, BIC score = **-1166.498**.
- **Yes,** the criterion selects the correct number of clusters for the toy data.

Part 2: Mixture models for matrix completion

a. When we integrate the mixture density over all unobserved coordinate values, the zero values won't contribute anything, because each coordinate value is independent. Therefore, we know we will get the same log likelihood value. This is why we can use the exact same formula from part 1 despite having missing values.

d.

Code:

Code for Estep Part 2

```
def Estep_part2(X,K,Mu,P,Var):
    n,d = np.shape(X) # n data points of dimension d
    post = np.zeros((n,K)) # posterior probabilities to compute
    LL = 0.0    # the LogLikelihood

    for u in xrange(len(X)):
        for j in xrange(K):
        post[u,j] = np.log(P[j]) + Gaussian_part2(X[u,:],Mu[j,:],Var[j])
        log_c = np.max(post[u,:])
        log_scaling_factor = np.log(np.sum(np.exp(post[u,:] - log_c)))
        post[u,:] = post[u,:] - log_c - log_scaling_factor
        LL += log_scaling_factor + log_c

    return (np.exp(post),LL)
```

Code for Mstep Part 2

```
def Mstep_part2(X,K,Mu,P,Var,post, minVariance=0.25):
    n,d = np.shape(X) # n data points of dimension d

    #code n_hat
    n_hat = np.sum(post, axis = 0).T

    #code pi (mixture proportions)
    P = n_hat/n

    delta = (X > 0).T

    #code mu
    candidate_Mu = np.divide(np.dot(post.T, X), np.dot(delta,post).T)
    decision_value = np.dot(delta, post)

    for j in xrange (K):
        for l in xrange(d):
            if decision_value[l, j] >= 1:
                Mu[j, l] = candidate_Mu[j, l]
        Var[j] = max(0.25,
                     np.dot(np.sum(np.multiply(np.subtract(X,
Mu[j,:])**2, delta.T), axis=1),
```

```
                                post[:,j]) / np.dot(np.sum(delta.T, axis=1),
post[:,j]))

    return (Mu,P,Var)
```

Code for mixGauss Part2

```
def mixGauss_part2(X,K,Mu,P,Var):
    n,d = np.shape(X) # n data points of dimension d
    post = np.zeros((n,K)) # posterior probs tbd

    prevLL = -1e18
    curLL = -1e17
    LL = []
    while True:
        prevLL = curLL
        (post,curLL) = Estep_part2(X,K,Mu,P,Var)
        LL.append(curLL)
        print curLL
        if (curLL-prevLL) <= 1e-6*np.abs(curLL): break
        (Mu,P,Var) = Mstep_part2(X,K,Mu,P,Var,post)

    return (Mu,P,Var,post,np.array(LL))
```

e. LL of best mixture: **-1375095.6021**

f. Below is the mathematical expression for completing a particular row x_C

$$x_i^{(t)} = \Sigma\, \mu_i^{(j)}\, p(y^{(t)} = j \mid x^{(t)}), \text{ for } i \notin C_t$$

g. RMSE = 0.49