ML: Homework 3
EM algorithms with Gaussian Mixtures

1. EM Implementation

1.1. Expectation Maximization Algorithm for Clustering

We consider a graphical model in which we have observations $\mathbf{x}^{(n)}$. Each of them belong to classes $\mathbf{z}^{(n)}$ which are unobserved. We represent a class in the form $z^{(n)} \in \{0,1\}^K$ such that $\sum_{k=1}^{K} z_k^{(n)} = 1$. There are a set of model parameters $\theta = (\pi, \mu, \Sigma)$. Here $\pi_k = Pr(z_k^{(n)} = 1)$. And $Pr(x^{(n)}|z_k^{(n)} = 1) \sim \mathcal{N}(\mu_k, \Sigma_k)$.

We first guess some θ_0 and iteratively improve it by the EM algorithm. For each θ_i, we have $\gamma_k^{(n)} = Pr(z_k^{(n)} = 1|x_k^{(n)}; \theta_i)$ Now we compute a new value θ_{i+1} by maximizing the expected log likelihood $\mathcal{L}(\theta) = E(\ln Pr(X, Z; \theta))$ where the expectation is taken over $Z \sim \gamma$. $\theta' = \arg\max_\theta \mathcal{L}(\theta)$. We solve the differential equations $\frac{\partial \mathcal{L}}{\pi_k}, \frac{\partial \mathcal{L}}{\mu_k}, \frac{\mathcal{L}}{\partial \Sigma_k}$ respectively to find it.

Written explicitly, the above translates to the following algorithm

1. **E Step**

$$\gamma_k^{(n)} = \frac{\pi_k \mathcal{N}(x^{(n)}; \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x^{(n)}; \mu_l, \Sigma_l)} \tag{1}$$

$$N_k = \sum_{n=1}^{N} \gamma_k^{(n)} \tag{2}$$

2. **M Step**

$$\mu_k' = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} x_n \tag{3}$$

$$\pi_k' = \frac{N_k}{N} \tag{4}$$

$$\Sigma_k' = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} \|(x^{(n)} - \mu_k')\|^2 \tag{5}$$

We implemented the above algorithm in python and used a minimum threshold on the change in expected log likelihood as convergence criterion. The expected log likelihood is given by

$$\mathcal{L}(\pi, \mu, \Sigma) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_k^{(n)} (\ln \pi_k + \ln \mathcal{N}(x^{(n)}, \mu_k, \Sigma_k)) \tag{6}$$

We used the log-sum-exp trick to compute its value. One notable thing we observed was the concentration of single classes onto a single point. In this case the covariance matrix tends to zero. We checked for this special case and restarted our algorithm from random initial θ. In some cases there were more than hundred restarts (especially when number of classes was large).

In figure 1 We observed how different choice of parameters affected our results. Here is a brief overview of key observations:

1. **Number of classes (K)**

   As we increase k, $\mathcal{L}$ increases monotonically, but we have overfitting for larger values of k. A suitable value for the given dataset turned out to be 2.

2. **Number of Data points (N)**

   The algorithm works better for larger values of n. For small values, it is more susceptible to noise and outliers and the results are not robust.

3. **Choice of initial parameters (θ_0)**

   Different initial parameters sometimes lead to different clusters, but sometimes results in very similar results. The output seems to be more sensitive to initial condition when the data set is not easily seperable, and when the value of K is not suitable for the dataset.

4. **Choice of convergence threshold**

   As expected, the number of iterations is larger for smaller values of the threshold. The output is bad for large values of the threshold, but results in essentialy same configuration beyond a certain value of the threshold. So the convergence pattern is slowed when the result comes close to the local optima.

2. EM Variances

Diagonal Covariance

If Σ_k is restricted to be diagonal, then the algorithm changes very slightly. In the step where we are computing $\Sigma'_k = \arg\min_{\Sigma'_k} \mathcal{L}(\pi', \mu', \Sigma)$, we take the partial derivative with respect to each component Σ_{kd} of Σ_k, By setting them equal to zero, we obtain the optimum value

$$\Sigma'_{kd} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_K^{(N)} (x^{(n)} - \mu_{kd})^2 \tag{7}$$

Where μ_{kd} is the d-th component of μ_k.

Qualitatively, this restriction means that the clusters we'll obtain will have their axes oriented parallel to the axes. We shown this in Figure 2

K-means

The K-means algorithm is a iterative hard-assignment algorithm that starts with a random assignment of points to clusters and iterates it to obtain better clusters. At each step we compute the center of mass of cluster k

$$\mu_k = \frac{1}{N_k} \sum_{z_{nk}=1} x_n \tag{8}$$

Where $N_k = \sum_{n=1}^{N} z_{nk}$ is the number of elements in class k. At the iteration, we assign z'_n to be class $k = \arg\min_k \|\mu_k - x_n\|$ After termination of K-means, we use the center of mass of the clusters as the initial parameters μ_k, the covariance of the points as the initial values Σ_k and the fraction of points N_k/N in each cluster as the initial π_k. The result is shown in Figure 3

Comparison

When we enforce the diagonal matrix constraint, the equipotential ellipses have major and minor axes parallel to the x and y axes, as expected. Hence the output does not capture clusters which are diagonally distributed. The algorithm also converges with smaller number of iterations.

Restricting to diagonal covariance also makes the algorithm more sensitive to the initial conditions, this is expected since it is unable to find good clusters.

When we used k-means as the initial condition, we get better local maxima. So it solves the problem of suitable initial conditions. But the output of the EM is still sensitive to the initial conditions of the k-means.

3. EM Model Selection

3.1. Model Space

In this section we consider a space of models. Specifically, we parametrize the following: the number of classes K from 1 through 5, and co-variance matrices between *diagonal* and *full* matrix.

3.2. Choices

To generate a gaussian mixture model for each parametrization, we make the following choices: We initialize with k-mean clustering, because earlier results shown that using k-mean clustering as initial guess gives better fits. We only run the gaussian-mixture once instead of running it multiple times and picking the best one. Running it multiple times will give a better fit, however, after experimenting with the parameters, we observed the following: For most of the runs, although the parameters of the gaussian mixture are different, the performance of these mixtures (under average log-likelihood) is similar. We suspect that the performance of these mixtures under K-fold cross validation will likely to be different, but since cross-validation is expensive to perform, we cannot afford to re-run the gaussian mixture fitting multiple times to the point we notice a significant improvement.

3.3. Model ranking under average log-likelihood

Here we rank the models according to the average log-likelihood. Table 1 documents the best models for each data set.

In short, average log-likelihood encourages overfitting. Here, the best models are the ones with the most parameters, with K 5 and Σ the full co-variance matrix. We can see a trend of log likelihood increasing in Graph 4 as K increase, and we also note that the log-likelihood is higher for the full covariance matrix than the diagonal one. We also note that the small data-set has a higher avg-log-likelihood performance than the large data-set, this is because we can over-fit better with only 5 classes to a small data-set than to a big data-set. We elect not to show the figures for the gaussian mixtures here, as they are overfitted and offers little insights to the underlying model.

Data Set	avg-log-lkh	best K	best Σ
data1-large	-3.247	5	full
data1-small	-2.685	5	full
data2-large	-2.675	5	full
data2-small	-2.263	5	full
data3-large	-3.594	5	full
data3-small	-3.348	5	full

Table 1. Best models under ranking via avg-log-likelihood

3.4. Model ranking under K-fold cross validation

In K-fold cross validation, we divide the data into K partitions of roughly equal size. The model is trained K times on the partitions, each time with one partition as the held-out set, and the rest of the partitions as training set. The performance of the model is measured as the average avg-log-likelihood over the K different held-out sets. We first run the tests under 5-fold cross validation, documented in Table 2. We then run the tests under 2-fold cross validation, documented in Table 3.

In short, cross validation deals with overfitting well. Several observations are important: Firstly, comparing to the avg-log-likelihood scores without cross-validation, the scores here are overall lower, meaning with cross-validation we cannot overfit as much to the data. Secondly, we note that for majority of the cases (beside data2 under 5-fold), the small data-set is fitted with fewer classes. Because with a small data-set at the training phase, we know very little about the distribution of the samples, thus it is much safer to use fewer classes to fit these few points. By constrast, for a large data-set, we are more certain about the distribution of the samples, and we can afford to fit more classes. Thirdly, we note that with cross-validation, the score is higher for the big data-set. This is due to given more points, one can more accurately construct a model performs better in the held-out set. This is the opposite without cross-validation, as we noted that the score was higher for the small data-set there. Finally, we note that the K value is higher for the 5-fold cross validation than the 2-fold. This is again due to during the training phase we have more points in the 5-fold than in the 2-fold, thus we can afford to fit more classes in the 5-fold. We plot the performance of the different parameters in graph 5, where we see the performance increases at first with more parameters, then decreases from overfitting. We also show the figures for the best-fits of the data-sets in figure 6.

Data Set	avg avg-log-lkh	best K	best Σ
data1-large	-3.413	3	diag
data1-small	-3.610	1	diag
data2-large	-2.837	2	full
data2-small	-3.470	4	full
data3-large	-3.704	3	diag
data3-small	-3.799	1	full

Table 2. Best models under 5-fold cross validation

Data Set	avg avg-log-lkh	best K	best Σ
data1-large	-3.370	2	full
data1-small	-3.630	1	full
data2-large	-2.876	2	full
data2-small	-3.119	3	diag
data3-large	-3.699	3	full
data3-small	-3.761	1	full

Table 3. Best models under 2-fold cross validation

4. Mystery Set

For the mystery data set, we gave our best attempt. That means selected from the space of models the best model under 5-fold cross-validation. The best fits are shown in Figure 7. We also used the 2-fold cross-validation to fit the data, but the results are similar. Overall, we feel there are too few points in the mystery set to give a accurate fit.

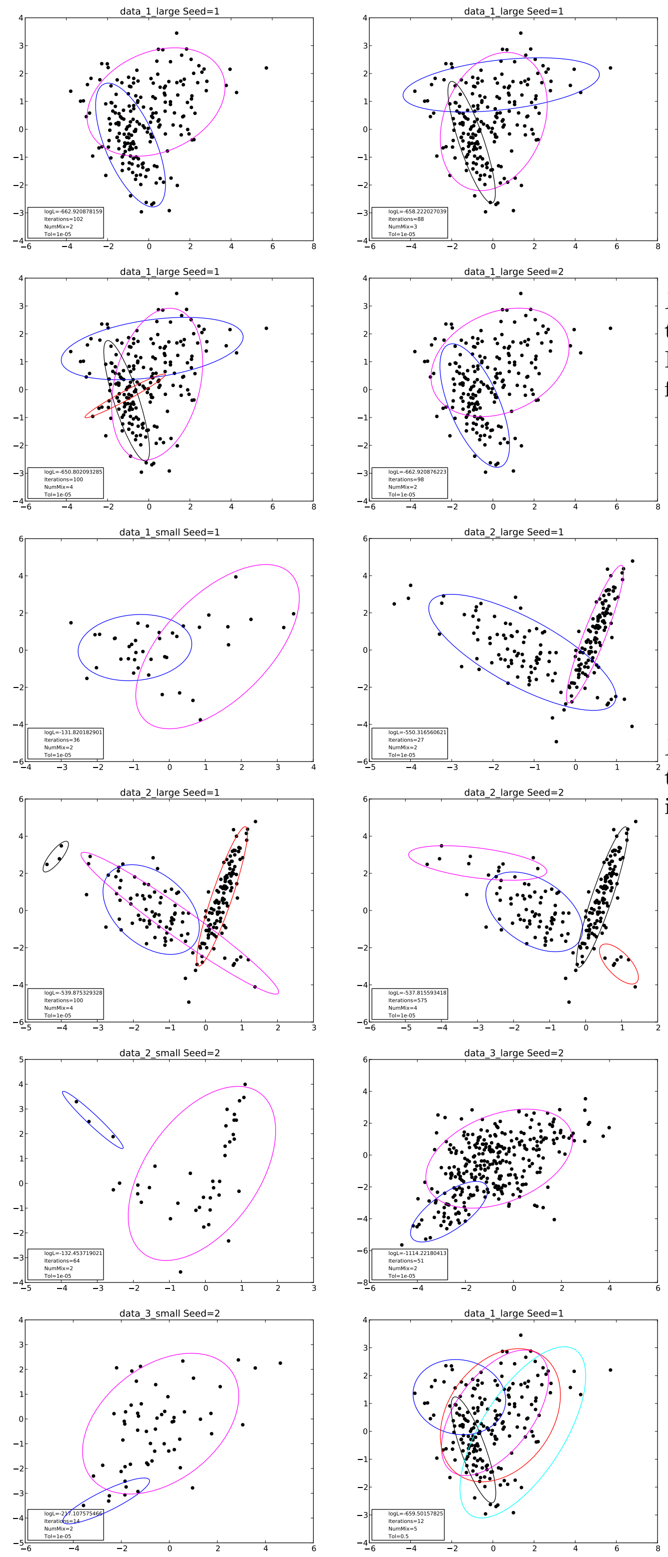

Figure 1. The above plots show the effect of parameters, i.e. number of classes, number of data points, choice of initial parameters (seed for random number generator) on the output

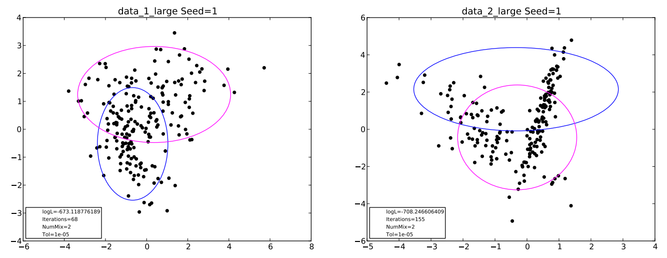

Figure 2. The above plots show the outputs obtained when the co-variance matrix were restricted to diagonal ones. Data set one is fitting somewhat, while dataset 2 is not fitting at all because it's classes are diagonally aligned.

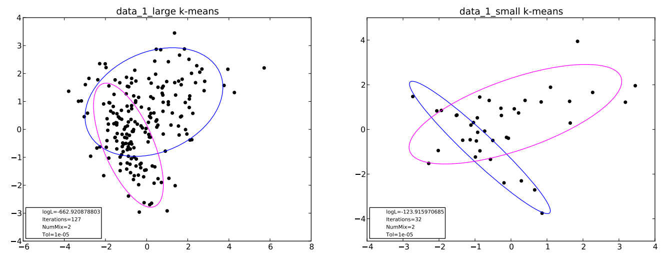

Figure 3. The above plots show the outputs obtained when the initial parameters were chosen using K-means. There is not much noticable features of the output.

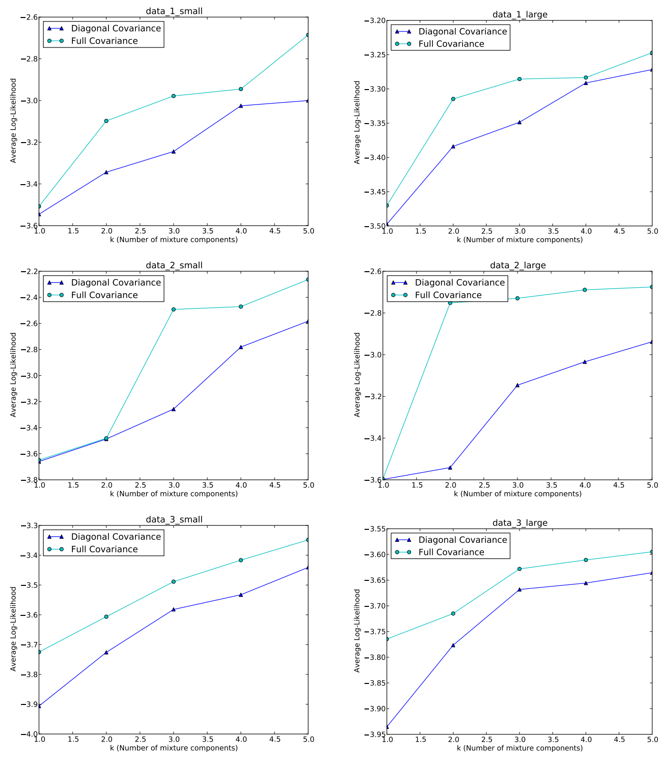

Figure 4. Scores under avg-log-likelihood for different parameters

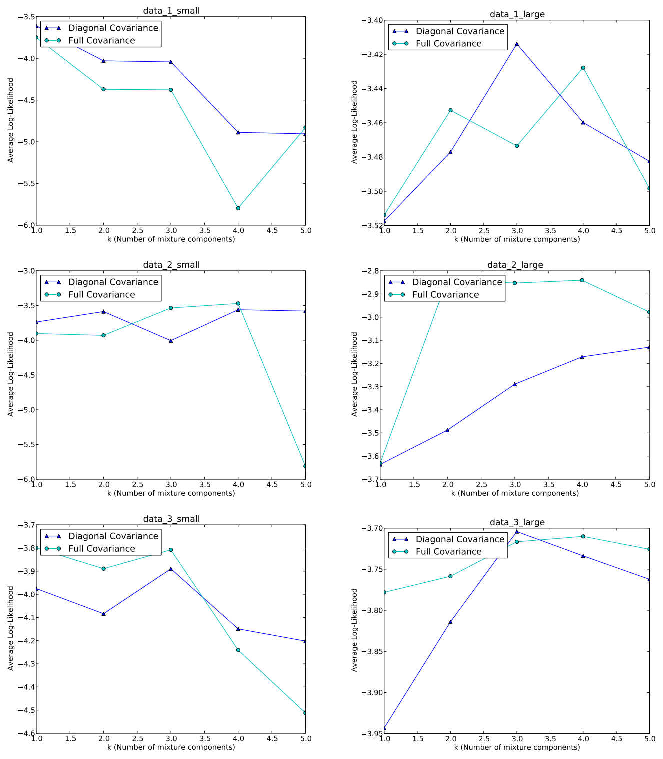

Figure 5. Scores under 5-fold cross validation for different parameters

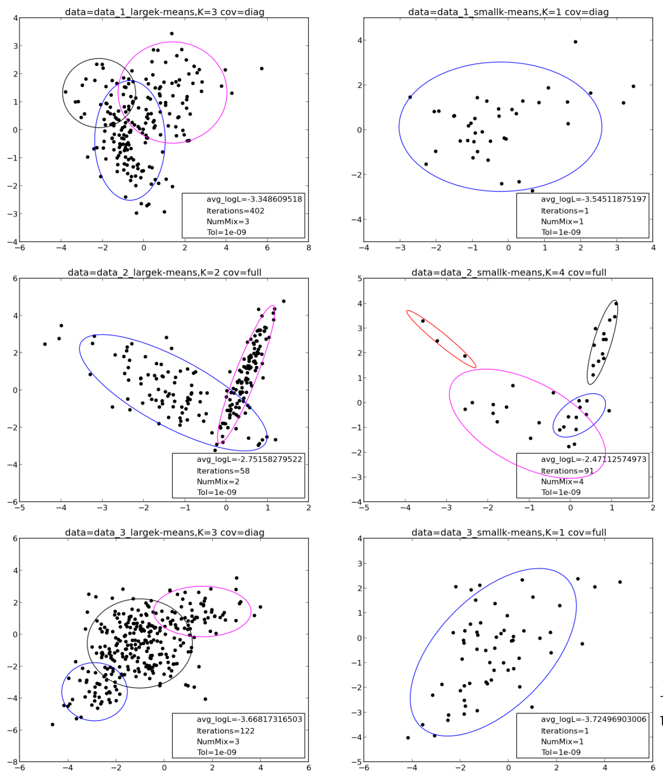

Figure 6. Best mixture of gaussian fits for each data-set under 5-fold validation

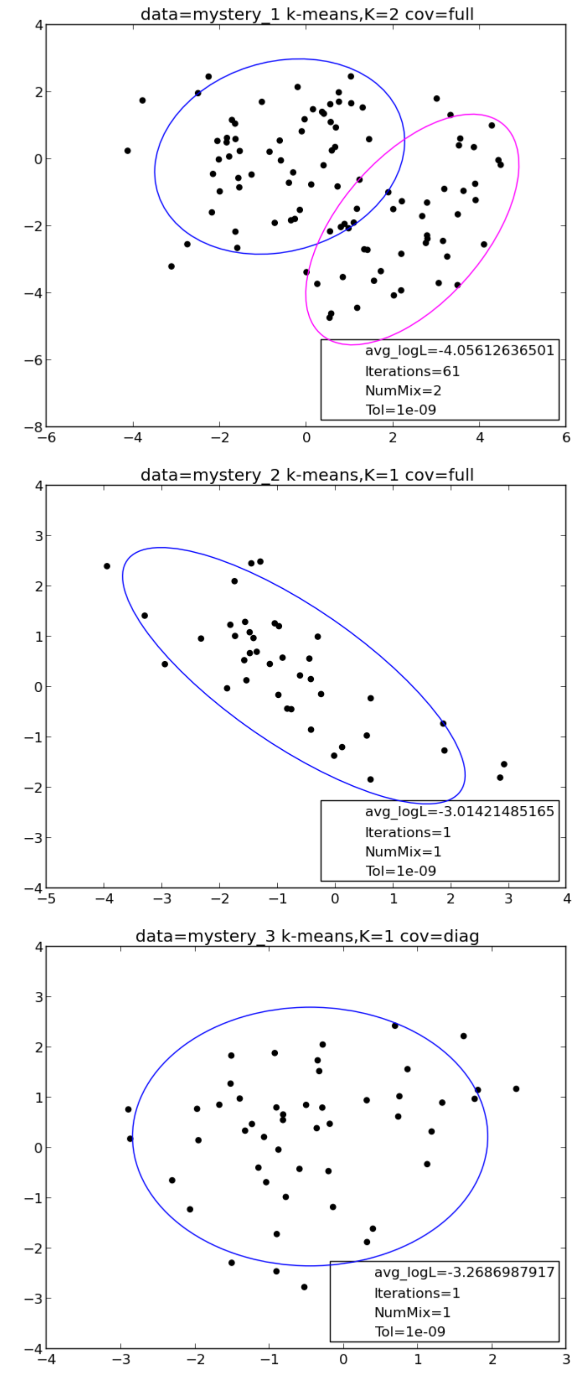

Figure 7. Best mixture of gaussian fits for each data-set under 5-fold validation