Expectation-Maximization for Mixture of Gaussians

Tanner Fiez

1 EM for Mixtures of Gaussians

This section concerns the implementation of the EM for Mixtures of Gaussians algorithm in python on the wine data set. The algorithm is given below, and there are a few key points to note. We work with log probabilities and use the log-sum-exp trick. Consider the likelihood of the multivariate Gaussian.

$$P(\underline{x_i}|C_j) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\underline{x_i}-\underline{\mu_j})^T\Sigma_j^{-1}(\underline{x_i}-\underline{\mu_j})}$$

We calculate the log probability of the multivariate Gaussian likelihood. In other words we calculate the following.

$$log(P(\underline{x_i}|C_j)) = -\frac{1}{2}(\underline{x_i}-\underline{\mu_j})^T\Sigma_j^{-1}(\underline{x_i}-\underline{\mu_j}) - \frac{d}{2}log(2\pi) - \frac{1}{2}log(|\Sigma|).$$

In calculating the posterior probabilities we again work with log probabilities. We will also employ the log-sum-trick in order to avoid underflows as below.

$$log(P(C_j|\underline{x_i})) = log(\frac{P(C_j)P(\underline{x_i}|C_j)}{\sum_{i'} P(C_{i'})P(\underline{x_i}|C_{i'})})$$

$$log(P(C_j|\underline{x_i})) = log(P(C_j)P(\underline{x_i}|C_j)) - log(\sum_{i'} P(C_{i'})P(\underline{x_i}|C_{i'}))$$

$$log(P(C_j|\underline{x_i})) = log(P(C_j)) + log(P(\underline{x_i}|C_j)) - log(\sum_{i'} e^{log(P(C_{i'})P(\underline{x_i}|C_{i'}))})$$

$$log(P(C_j|\underline{x_i})) = log(P(C_j)) + log(P(\underline{x_i}|C_j)) - log(\sum_{i'} e^{(log(P(C_{i'}))+log(P(\underline{x_i}|C_{i'})))})$$

Letting $a_{i'} = (log(P(C_{i'})) + log(P(\underline{x_i}|C_{i'})))$, the equation simplifies to the following.

$$log(P(C_j|\underline{x_i})) = log(P(C_j)) + log(P(\underline{x_i}|C_j)) - log(\sum_{i'} e^{a_{i'}})$$

Using the log-sum-exp trick the solution is reached.

$$log(P(C_j|\underline{x_i})) = log(P(C_j)) + log(P(\underline{x_i}|C_j)) - (a_{max} + log(\sum_{i'} e^{a_{i'}-a_{max}}))$$

We also check that use the log-sum-exp trick when calculating the complete log likelihood which is given by the following equation.

$$L = \sum_{i=1}^{n} log \sum_{j=1}^{k} P(C_j)P(\underline{x_i}|C_j)$$

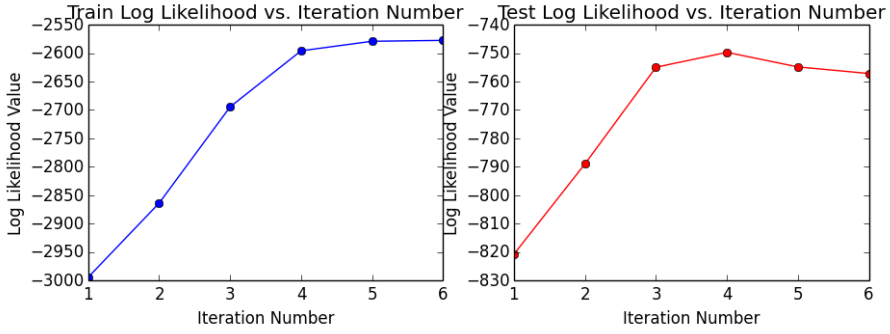
Pseudocode for the EM algorithm is provided below.

Algorithm 1 EM-algorithm

function EM-MIX(x,k,n,d)
  Initialization:
  $P(C_i) \leftarrow \frac{1}{k}$ for $i = 1, \cdots, k$ ▷ *Initialize component priors to uniform distribution*
  $\underline{\mu_i} \leftarrow x[\text{RandomInt}[0, \text{n})]$ for $i = 1, \cdots, k$ ▷ *Initialize component means to random data point*
  $\sigma^2_{i,j} \leftarrow \frac{\text{range(x[:,i])}}{j}$ for $i = 1, \cdots, d$ for $j = 1, \cdots, k$ ▷ *Initialize variances to fixed fraction of range*
  iteration $\leftarrow 1$ ▷ *Iteration number*
  $m \leftarrow 100$ ▷ *Upper iterations bound*
  $L_I \leftarrow -\infty$ ▷ *Initial log-likelihood*
  Main Program:
  while *iteration* $< m$ **do**
    $i \leftarrow 0$
    for $i < n$ **do**
      $P(C_j|\underline{x_i}) = \frac{P(C_j)P(\underline{x_i}|C_j)}{\sum_{i'} P(C_{i'})P(\underline{x_i}|C_{i'})}$ for $j = 1, \cdots, k$ ▷ *E–step: update posterior probabilities*
    $i \leftarrow 0$
    for $i < k$ **do**
      $P(C_i) = \frac{1}{n}\sum_{j=1}^{n} P(C_i|\underline{x_j})$ ▷ *M–step: update prior probabilities*
      $\underline{\mu_i} = \frac{\sum_{j=1}^{n} \underline{x_j} P(C_i|\underline{x_j})}{\sum_{j=1}^{n} P(C_i|\underline{x_j})}$ ▷ *M–step: update component means*
      $\underline{\sigma_i^2} = \frac{\sum_{j=1}^{n} (\underline{x_j} - \underline{\mu_i})^2 P(C_i|\underline{x_j})}{\sum_{j=1}^{n} P(C_i|\underline{x_j})}$ ▷ *M–step: update component variances*
      $L_N = \sum_{i=1}^{n} log \sum_{j=1}^{k} P(C_j)P(\underline{x_i}|C_j)$ ▷ *Update the complete log likelihood*
    if $\frac{|(L_N - L_I)| \cdot 100}{|L_I|} < 0.1$ **then** ▷ *Check for convergence*
      break
    else
      $L_I \leftarrow L_N$
    iteration $\leftarrow$ iteration $+ 1$
  Outputs: $P(C)$, μ, σ, $P(C|x)$, L_N

1.1 Train and Test Likelihood

In this section the EM algorithm is trained on the wine.train dataset and the log likelihood is recorded after each iteration. At the same time that the log likelihood is calculated for the training data, the log likelihood of the wine.test data set is calculated using the current model from the EM algorithm on the training set. The plots of this are provided below.



There are a few things to notice in these plots. One is that the train log likelihood is monotonically increasing which is a guarantee from the EM algorithm. We can also see that the test log likelihood is not monotonically increasing. It is not guaranteed to be so, when using the training data model but is a good sign of generalization that it is increases for the majority of the plot. This run of the algorithm took 6

iterations to converge, but using random seeds for the means of the clusters this varies from run to run. During my testing I typically saw between 5-10 iterations before convergence was reached.

1.2 Methods of Initialization

Experimenting with the method of initialization for the mean of each Gaussian in each cluster I found the method of using random examples, meaning choosing a random data point and initializing it to the cluster mean to be the best method. This method converged much faster and with a slightly higher likelihood than if I used random values. One way I tried random values was to set each cluster mean to the average over all the data points. I also tried adding some noise to this term for each of the cluster means. Still I found the random examples to be the most effective. I think the reason for this is that each each data point is close to one of the true means. So some of the initializations will be close to the true mean. Also data points that are not close to an initialization that was close to a true mean will clearly be further apart, which will lead to faster corrections to the means of the other clusters.

1.3 Random Seeds Likelihood

In this section I ran the EM algorithm on the wine.train data set and at each iteration recorded the the log likelihood of the training set as well as the log likelihood of the test set using the model obtained from the training set.

Table 1: Final log likelihoods for EM with 10 Random Seeds.

Simulation	Final Train Log Likelihood	Final Test Log Likelihood
1	-2572.25	-753.06
2	-2574.14	-748.70
3	-2576.98	-742.07
4	-2778.75	-809.35
5	-2577.03	-758.76
6	-2571.85	-756.30
7	-2571.93	-756.05
8	-2576.93	-742.13
9	-2572.36	-756.34
10	-3063.97	-843.77

Looking at the results there is a group of simulations very close on the training likelihood. These are all in the -2570's. The test set likelihoods corresponding to this group are also similar. There were a couple of outliers including simulation 4 and 10. This is to be somewhat expected with the random seeds using random examples for the cluster means. It also shows why it can be important to run the algorithm more than once with different initializations and to use the initialization that leads to the highest log likelihood.

1.4 EM Clustering vs. True Clustering

To infer the most likely clustering for each point in the training data, I use the final posterior probabilities. For each data point I find which cluster gives the highest posterior probability, then I add the data point to a dictionary for that cluster. To compare with the wine-true.train, I then find for each cluster which label is most frequent for the data points in the cluster. The data points in the cluster that do not have the most frequent label in the cluster are then marked as incorrect predictions. Running 10 simulations again with random seeds, I provide a table of the accuracy of the clustering.

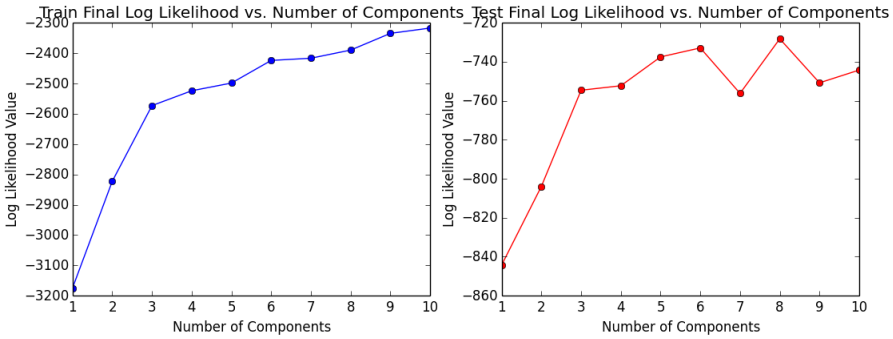
Table 2: Classification Accuracy with 10 Random Seeds.

Simulation	Final Train Log Likelihood	Classification Accuracy
1	-2576.90	88.73
2	-2574.37	93.66
3	-2570.02	97.18
4	-2572.86	95.77
5	-2572.92	94.36
6	-2770.36	69.01
7	-2781.52	69.71
8	-2573.07	95.77
9	-2571.91	95.77
10	-2577.55	92.95

This table shows a few interesting things. The first is that a higher log likelihood will most likely correspond to a higher classification accuracy. It also shows how different initializations can impact the final results, so it is worth running the simulation more than once while evaluating the log likelihood with different simulations.

1.5 Varying Number of Clusters

This section concerns varying the number of clusters when running the EM algorithm and observing the effect that this has on the log likelihood of both the training and test data sets. Below is a plot of the final log likelihood versus the number of components for both the training set and the test set. Also to reiterate, this is the log likelihood from training on the wine.train and calculating the log likelihood on the train data, then using the trained model to calculate the log likelihood of the test set.



This plot shows a few interesting things. First note that the final log likelihood is not guaranteed to be increasing with the number of components in part due to the fact I am using random seeds, but in this iteration is was. Another important observation is the log likelihood is increasing on the training set, although the major gains are made within the first 3 components. The elbow of the plot is at 3 components and that is generally a metric used to pick the number of components for clustering algorithms. This is as we would expect since we have knowledge that there are 3 classes. This means that within the three classes we could separate them out to make more precise groupings by similarity, but this would give hardly any more information which is one reason why 3 classes are clearly sufficient.

Now looking at the final likelihoods of the test set, we can see that many of the same characteristics of the training plot are exhibited. The most important of these is the fact that major gain in the log likelihood is used up to 3 components. In other words, the elbow of the plot is at 3 which is showing again that the true number of components is likely 3. As the number of components goes up we can see that the likelihood is generally staying the same but with minimal dips and increases, which is due to over-fitting as the number of clusters increases.