# Mixture Models and the EM Algorithm

Padhraic Smyth,
Department of Computer Science
University of California, Irvine

## 1 Finite Mixture Models

Say we have a data set $D = \{\underline{x}_1, \ldots, \underline{x}_N\}$ where $\underline{x}_i$ is a $d$-dimensional vector measurement. A flexible model for $p(\underline{x}_i)$ is a finite mixture model with $K$ components:

$$p(\underline{x}_i|\Theta) = \sum_{k=1}^{K} \alpha_k \, p_k(\underline{x}_i|z_{ik} = 1, \theta_k) \tag{1}$$

where:

- The sum over $k$ above is in effect just an application of the law of total probability where we are summing out over the random variable $z_i$ (but with some extra notation, e.g., using binary indicator variables $z_{ik}$ for convenience).
- $z_i = (z_{i1}, \ldots, z_{iK})$ is a vector of $K$ binary indicator variables that are mutually exclusive and exhaustive (i.e., one and only one of the $z_{ik}$'s is equal to 1, and the others are 0). $z_i$ plays the role of an indicator random variable representing the identity of the mixture component that generated $\underline{x}_i$. In unsupervised learning we observe the $\underline{x}$'s and not the $z$'s, and the $z_i$'s are considered to be hidden or latent.
- The $p_k(\underline{x}|z_{ik} = 1, \theta_k)$ are *mixture components*, $1 \leq k \leq K$. Each is a density or distribution defined over $p(\underline{x}_i)$, with parameters $\theta_k$. For example, each of the components could be a Gaussian multivariate density function, each with its own mean vector $\underline{\mu}_k$ and covariance vector $\Sigma_k$. In general the components can be any distribution or density function defined on $\underline{x}$, and the components need not all have the same functional form.
- When we condition on $z_{ik} = 1$ in each of the components in the sum above we are evaluating the component density at $\underline{x}_i$ assuming that it was generated by component $k$. Note that an implicit assumption in finite mixture models, in the generative sense, is that each data point $\underline{x}_i$ was generated by just one of the $K$ components. The generative model has two steps: (1) $z_i \sim p(z) = (\alpha_1, \ldots, \alpha_K)$, and (2) $\underline{x}_i \sim p_k(\underline{x}_i|z_{ik} = 1, \theta_k)$.

- The $\alpha_k = p(z = k) = p(z_{ik} = 1)$ are the mixture weights, representing the probability that a randomly selected $\underline{x}_i$ was generated by component $k$, where $\sum_{k=1}^{K} \alpha_k = 1$. Note that $p(z = k) = p(z_{ik} = 1)$ because $p(z_{ik} = 1)$ is the *marginal* probability that a randomly selected $\underline{x}$ was generated by component $k$, i.e., $p(z_{ik})$ is not conditioned on knowing $\underline{x}_i$.

The complete set of parameters for a mixture model with $K$ components is

$$\Theta = \{\alpha_1, \ldots, \alpha_K, \theta_1, \ldots, \theta_K\}$$

with $\sum_{k=1}^{K} \alpha_k = 1, \alpha_k \geq 0$.

Mixture models are generally useful in a few different contexts:

- One general application is in *density estimation*: they allow us to build complex models out of simple parts. For example, a mixture of $K$ multivariate Gaussians may have up to $K$ modes, allowing us to model multimodal densities.
- A second motivation for using mixture models is where there is *an underlying true categorical variable* $z$, but we cannot directly measure it: a well-known example is pulling fish from a lake and measuring their weight $x_i$, where there are known to be $K$ types of fish in the lake but the types were not measured in the data we were given. In this situation the $z_i$'s correspond to some actual real-world phenomenon that could in principle be measured but that wasn't.
- A third motivation, similar to the second, is where we believe their might be $K$ underlying groups in the data, each characterized by different parameters, e.g., $K$ sets of customers which we wish to infer from purchasing data $\underline{x}_i$. This is often referred to as model-based clustering: there is not necessarily any true underlying interpretation to the $z$'s, so this tends to be more exploratory in nature than in the second case.

## 2 Gaussian Mixture Models

For $\underline{x}_i \in \mathcal{R}^d$ we can define a Gaussian mixture model by making each of the $K$ components a Gaussian density with parameters $\underline{\mu}_k$ and $\Sigma_k$. Each component is a multivariate Gaussian density

$$p_k(\underline{x}_i|\theta_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(\underline{x}_i - \underline{\mu}_k)^t \Sigma_k^{-1} (\underline{x}_i - \underline{\mu}_k)}$$

with its own parameters $\theta_k = \{\underline{\mu}_k, \Sigma_k\}$.

## 3 Learning Mixture Models from Data

To fit a mixture model to data we can use maximum likelihood (we can also be Bayesian if we wish, but its simpler to start with maximum likelihood). Assuming for simplicity that the data points $\underline{x}_i$ are conditionally independent given the model and its parameters $\Theta$, we have

$$l(\Theta) \;=\; P(D|\Theta) \;=\; \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \alpha_k \, p_k(\underline{x}_i | z_{ik} = 1, \theta_k)\right)$$

where $\alpha_k = p(z_{ik} = 1)$ is the marginal (unconditional) probability that a randomly selected $\underline{x}$ was generated by component $k$. If we take partial derivatives of this log-likelihood and set them to 0 we get a set of coupled non-linear equations. For example, if the component parameters $\theta_k$ were known and we were just learning the $\alpha_k$'s, we have

$$\frac{\partial l(\Theta)}{\partial \alpha_j} \;=\; \sum_{i=1}^{N} \frac{p_j(\underline{x}_i | z_{ij} = 1, \theta_j)}{\sum_{k=1}^{K} p_j(\underline{x}_i | z_{ij} = 1, \theta_k)\alpha_k}, \quad 1 \le j \le K.$$

Setting these to 0, we get $K$ non-linear equations (since the $\alpha$'s sum to 1 there we would also need a Lagrangian term here to enforce this constraint). We could try to solve these equations directly, for example by using iterative local gradient descent (or ascent). Gradient-based methods are certainly a valid approach for learning the parameters of mixture models, but comes with the cost of having to set learning rates, etc.. In addition in this context we would need to enforce the constraint (during gradient search) that the $\alpha_k$'s are non-negative and sum to 1.

A widely-used alternative in this context is the Expectation-Maximization (EM) algorithm. The EM algorithm is an iterative algorithm for doing "local ascent" of the likelihood (or log-likelihood) function. It is usually easy to implement, it enforces parameter constraints automatically, and it does not require the specification of a step-size (in some sense the step-size is implicit at each iteration of the algorithm). Below we discuss the EM algorithm for mixture models, focusing primarily on Gaussian mixtures. It is important to note however that EM is a much more general procedure and is broadly applicable to maximizing the likelihood in any problems where there is missing data (for mixture models the missing data are the $z_i$ indicators for component membership for each data point $\underline{x}_i$).

## 4 The EM Algorithm for Mixture Models

### 4.1 Outline of the EM Algorithm for Mixture Models

The EM algorithm is an iterative algorithm that starts from some initial estimate of the parameter set $\Theta$ or the membership weights (e.g., random initialization) and then proceed to iteratively update the parameter estimates until convergence is detected. Each iteration consists of an E-step and an M-step.

In the E-step the algorithm computes the expected log-likelihood with respect to the probability of the $\underline{z}_i$'s conditioned on the $\underline{x}_i$'s and the current values of the parameters. For mixture models the expected value $E[z_{ik}] = p(z_{ik} = 1|\underline{x}_i, \Theta)$ (since the $z_{ik}$'s are binary). This is computed for all $N$ data points for each of the $K$ components, using Bayes rule (more details below).

In the M-step the algorithm computes new parameter values that maximize the expected log-likelihood, given the $N \times K$ matrix of $p(z_{ik} = 1|\underline{x}_i, \theta_k)$ values produced by the E-Step.

Both the E-step and M-Step are usually straightforward to compute for mixture models, typically scaling linearly in both $N$ and $K$. The fact that the E and M steps can be computed and implemented in code in a straightforward manner is an appealing property of the EM algorithm and is one of the reasons it is very popular in practice for fitting mixture models.

## 4.2 The E-Step for Mixture Models

In the E-Step, given a current set of parameters $\Theta$, we compute the "membership weight" of data point $\underline{x}_i$ in component $k$ as

$$w_{ik} \;=\; p(z_{ik}=1|\underline{x}_i,\Theta) \;=\; \frac{\alpha_k \cdot p_k(\underline{x}_i|z_k,\theta_k)}{\sum_{m=1}^K \alpha_m \cdot p_m(\underline{x}_i|z_m,\theta_m)}, \quad 1 \le k \le K, \quad 1 \le i \le N.$$

This follows from a direct application of Bayes rule in the context of Equation 1.For Gaussian mixture models the component density functions $p_k(\underline{x}_i|\ldots)$ are Gaussian multivariate densities—if we were using a mixture model with different components, the components would have different functional forms, but the general equation above for computing mixture weights has the same general form.

These membership weights can be stored as an $N \times K$ matrix where each row sums to 1 and contains the membership weights for data vector $\underline{x}_i$.

The membership weights reflect our uncertainty, given $\underline{x}_i$ and $\Theta$, about which of the $K$ components generated vector $\underline{x}_i$. Note that we are assuming in our generative mixture model that each $\underline{x}_i$ was generated by a single component—so these probabilities reflect our uncertainty about which component $\underline{x}_i$ came from, not any "mixing" in the generative process (there is a different type of mixture model that allows for such mixing, referred to as "admixture models", which have been adapted in machine learning as topic models, used for modeling documents as combinations of components consisting of multinomial distributions over a vocabulary of words).

## 4.3 The M-Step for Gaussian Mixture Models

Given the membership weights from the E-step we can use the membership weights and the data to calculate new parameter values. Let $N_k = \sum_{i=1}^N w_{ik}$, i.e., the sum of the membership weights for the $k$th component—this is the effective number of data points assigned to component $k$.

Our new estimate of the mixture weights is

$$\alpha_k^{new} = \frac{N_k}{N}, \quad 1 \le k \le K.$$

Our new estimates of the component means are

$$\underline{\mu}_k^{new} = \frac{1}{N_k} \sum_{i=1}^N w_{ik} \cdot \underline{x}_i \quad 1 \le k \le K.$$

The updated mean is calculated in a manner similar to how we could compute a standard empirical average, except that the $i$th data vector $\underline{x}_i$ has a fractional weight $w_{ik}$. Note that this is a vector equation since $\underline{\mu}_k^{new}$ and $\underline{x}_i$ are both $d$-dimensional vectors.

Finally, the new estimates of the component covariances are

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} w_{ik} \cdot (\underline{x}_i - \underline{\mu}_k^{new})(\underline{x}_i - \underline{\mu}_k^{new})^t \quad 1 \leq k \leq K.$$

Again we get an equation that is similar in form to how we would normally compute an empirical covariance matrix, except that the contribution of each data point is weighted by $w_{ik}$. Note that this is a matrix equation of dimensionality $d \times d$ on each side.

After we have computed all of the new parameters, the M-step is complete and we can now go back and recompute the membership weights in the E-step, then recompute the parameters again in the E-step, and continue updating the parameters in this manner. Each pair of E and M steps is considered to be one iteration.

## 5 Initialization and Convergence Issues for EM

The EM algorithm can be started by either initializing the algorithm with a set of initial parameters and then conducting an E-step, or by starting with a set of initial weights and then doing a first M-step. The initial parameters or weights can be chosen randomly (e.g. select $K$ random data points as initial means and select the covariance matrix of the whole data set for each of the initial $K$ covariance matrices) or could be chosen via some heuristic method (such as by using the k-means algorithm to cluster the data first and then defining weights based on k-means memberships).

The algorithm can be halted by detecting convergence (or trying to detect convergence given that there is no 100 percent surefire way to do this). One option to is to detect when the average of the membership weights (across all $N \times K$ weights) is changing by less than some amount (e.g., by $10^{-6}$) from one iteration to the next. Another option would be to halt when the value of the log-likelihood appears not to be changing in a significant manner from one iteration to the next. Note that the log-likelihood (under the IID assumption) is defined as follows:

$$\log l(\Theta) \;=\; \sum_{i=1}^{N} \log p(\underline{x}_i|\Theta) \;=\; \sum_{i=1}^{N} \left( \log \sum_{k=1}^{K} \alpha_k p_k(\underline{x}_i|z_k, \theta_k) \right)$$

where $p_k(\underline{x}_i|z_k, \theta_k)$ is the Gaussian density for the $k$th mixture component.

## 6 The $K$-means Algorithm

The $K$-means algorithm is another algorithm for clustering real-valued data. It is based on minimizing the sum of Euclidean distances between each point and its assigned cluster, rather than on a probabilistic model. The algorithm takes as input an $N \times d$ data matrix (with real-valued entries), a value for $K$, and operates as follows:

1. Initialize by randomly selecting $K$ mean vectors, e.g., pick $K$ data vectors (rows) randomly from the input data matrix

2. Assign each of the $N$ data vectors to the cluster corresponding to which of the $K$ clusters means it is closest to, where distance is measured as Euclidean distance in the $d$-dimensional input space.

3. For each cluster $k$, compute its new mean as the mean (average) of all the data vectors that were assigned to this cluster in Step 2.

4. Check for convergence. An easy way to determine convergence is to execute Step 2 and check if any of the data points change cluster assignments relative to their assignment on the previous iteration. If not, exit; if 1 or more points change cluster assignment, continue to Step 3.

The $K$-means algorithm can be viewed as a heuristic search algorithm for finding the cluster assignments that minimize the total sum of squares, namely the sum of the squared Euclidean distances from each of the $N$ data points to a cluster center. Finding the optimal solution is NP-hard, so $K$-means may converge to local minima. For this reason it can be useful to start the algorithm with multiple random starting conditions, and select the solution with the minimum sum of squares score over different runs.

The $K$-means algorithm can also be thought of as a simpler non-probabilistic alternative to Gaussian mixtures. $K$-means has no explicit notion of cluster covariances. One can "reduce" Gaussian mixture clustering to $K$-means if one were to (a) fix a priori all the covariances for the $K$ components to be the identity matrix (and not update them during the M-step), and (b) during the E-step, for each data vector, assign a membership probability of 1 for the component it is most likely to belong to, and 0 for all the other memberships (in effect make a "hard decision" on component membership at each iteration).

## Additional Reading

The new Murphy text (early 2021) has relevant material in Chapter 3.7 on Mixture models, Chapter 5.7.2 on the EM Algorithm, and Chapter 21.4 on Clustering using mixture models. See also the Background Reading section on mixtures and EM on the class Website.