Lecture topics:

- Different types of mixture models (cont'd)

- Estimating mixtures: the EM algorithm

Mixture models (cont'd)

Basic mixture model

Mixture models try to capture and resolve observable ambiguities in the data. E.g., an $m-$component Gaussian mixture model

$$P(\mathbf{x}; \theta) = \sum_{j=1}^{m} P(j) N(\mathbf{x}; \mu_j, \Sigma_j) \tag{1}$$

The parameters θ include the mixing proportions (prior distribution) $\{P(j)\}$, means of component Gaussians $\{\mu_j\}$, and covariances $\{\Sigma_j\}$. The notation $\{P(j)\}$ is a shorthand for $\{P(j), \ j = 1, \ldots, m\}$.

To generate a sample $\mathbf{x}$ from such a model we would first sample j from the prior distribution $\{P(j)\}$, then sample $\mathbf{x}$ from the selected component $N(\mathbf{x}; \mu_j, \Sigma_j)$. If we generated n samples, then we would get m potentially overlapping clusters of points, where each cluster center would correspond to one of the means μ_j and the number of points in the clusters would be approximately $n P(j)$. This is the type of structure in the data that the mixture model is trying to capture if estimated on the basis of observed $\mathbf{x}$ samples.

Student exam model: 1-year

We can model vectors of exam scores with mixture models. Each $\mathbf{x}$ is a vector of scores from a particular student and samples correspond to students. We expect that the population of students in a particular year consists of m different types (e.g., due to differences in background). If we expect each type to be present with an overall probability $P(j)$, then each student score is modeled as a mixture

$$P(\mathbf{x}|\theta) = \sum_{j=1}^{m} P(\mathbf{x}|\theta_j) P(j) \tag{2}$$

where we sum over the types (weighted by $P(j)$) since we don't know what type of student t is prior to seeing their exam score $\mathbf{x}_t$.

If there are n students taking the exam a particular year, then the likelihood of all the student score vectors, $D_1 = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, would be

$$L(D_1; \theta) = \prod_{t=1}^{n} P(\mathbf{x}_t|\theta) = \prod_{t=1}^{n} \left(\sum_{j=1}^{m} P(\mathbf{x}|\theta_j)P(j) \right) \tag{3}$$

Student exam model: K-years

Suppose now that we have student data from K years of offering the course. In year k we have n_k students who took the course. Let $\mathbf{x}_{k,t}$ denote the score vector for a student t in year k. Note that t is just an index to identify samples each year and the same index does not imply that the same student took the course multiple years. We can now assume that the number of student types as well as $P(\mathbf{x}|\theta_j)$ remain the same from year to year (the parameters θ_j are the same for all years). However, the population of students may easily change from year to year, and thus the prior probabilities over the types have to be set differently. Let $P(j|k)$ denote the prior probabilities over the types in year k (all of these would have to be estimated of course). Now, according to our mixture distribution, we expect example scores for students in year k be sampled from

$$P(\mathbf{x}|k, \theta) = \sum_{j=1}^{m} P(\mathbf{x}|\theta_j)P(j|k) \tag{4}$$

The likelihood of all the data, across K years, $D = \{D_1, \ldots, D_K\}$, is given by

$$L(D; \theta) = \prod_{k=1}^{K} \prod_{t=1}^{n_k} P(\mathbf{x}_{k,t}|k, \theta) = \prod_{k=1}^{K} \prod_{t=1}^{n_k} \left(\sum_{j=1}^{m} P(\mathbf{x}_{k,t}|\theta_j)P(j|k) \right) \tag{5}$$

The parameters θ here include the mixing portions $\{P(j|k)\}$ that change from year to year in addition to $\{\theta_j\}$.

Collaborative filtering

Mixture models are useful also in recommender systems. Suppose we have n users and m movies and our task is to recommend movies for users. The users have each rated a small fraction of movies and our task is to fill-in the rating matrix, i.e., provide a predicted rating for each user across all m movies. Such a prediction task is known as a collaborative filtering problem (see Figure 1).

Say the possible ratings are $r_{ij} \in \{1, \ldots, 5\}$ (i.e., how many stars you assign to each movie). We will use i to index users and j for movies; a rating r_{ij}, if provided, specifies how user

movies j

Figure 1: Partially observed rating matrix for a collaborative filtering task.

i rated movie j. Since only a small fraction of movies are rated by each user, we need a way to index these elements of the user/movie matrix: we say $(i, j) \in I_D$ if rating r_{ij} is available (observed). D denotes all the observed ratings.

We can build on the previous discussion on mixture models. We can represent each movie as a distribution over "movie types" $z_m \in \{1, \ldots, K_m\}$. Similarly, a user is represented by a distribution over "user types" $z_u \in \{1, \ldots, K_u\}$. We do not assume that each movie corresponds to a single movie type across all users. Instead, we interpret the distribution over movie types as a bag of features corresponding to the movie and we resample from this bag in the context of each user. This is analogous to predicting exam scores for students in a particular year (we didn't assume that all the students had the same type). We also make the same assumption about user types, i.e., that the type is sampled from the "bag" for each rating, resulting potentially in different types for different movies. Since all the unobserved quantities are summed over, we do not explicitly assign any fixed movie/user type to a rating.

We imagine generating the rating r_{ij} associated with (i, j) element of the rating matrix as follows. Sample a movie type from $P(z_m|j)$, sample a user type from $P(z_u|i)$, then sample a rating r_{ij} with probability $P(r_{ij}|z_u, z_m)$. All the probabilities involved have to estimated from the available data. Note that we will resample movie and user types for each rating. The resulting mixture model for rating r_{ij} is given by

$$P(r_{ij}|i, j, \theta) = \sum_{z_u=1}^{K_u} \sum_{z_m=1}^{K_m} P(r_{ij}|z_u, z_m)P(z_u|i)P(z_m|j) \tag{6}$$

where the parameters θ refer to the mapping from types to ratings $\{P(r|z_u, z_m)\}$ and the user and movie specific distributions $\{P(z_u|i)\}$ and $\{P(z_m|j)\}$, respectively. The likelihood

of the observed data D is

$$L(D; \theta) = \prod_{(i,j) \in I_D} P(r_{ij}|i, j, \theta) \tag{7}$$

Users rate movies differently. For example, some users may only use a part of the scale (e.g., 3, 4 or 5) while others may be bi-modal, rating movies either very bad 1 or very good 5. We can improve the model by assuming that each user has a rating style $s \in \{1, \ldots, K_s\}$. The styles are assumed to be the same for all users, we just don't know how to assign each user to a particular style. The prior probability that any randomly chosen user would have style s is specified by $P(s)$. These parameters are common to all users. We also assume that the rating predictions $P(r_{ij}|z_u, z_m)$ associated with user/movie types now depend on the style s as well: $P(r_{ij}|z_u, z_m, s)$.

We have to be a bit careful in writing the likelihood of the data. All the ratings of one user have to come from one rating style s but we can sum over the possibilities. As a result, the likelihood of the observed ratings is modified to be

$$L'(D; \theta) = \prod_{i=1}^{n} \left[\sum_{s=1}^{K_s} P(s) \overbrace{\prod_{j:(i,j) \in I_D} \left(\sum_{z_u=1}^{K_u} \sum_{z_m=1}^{K_m} P(r_{ij}|z_u, z_m, s) P(z_u|i) P(z_m|j) \right)}^{\text{likelihood of user } i\text{'s ratings with style } s} \right] \tag{8}$$

The model does not actually involve that many parameters to estimate. There are exactly

$$\overbrace{(K_s - 1)}^{\{P(s)\}} + \overbrace{(5-1)K_u K_m K_s}^{\{P(r|z_u, z_m, s)\}} + \overbrace{n(K_u - 1)}^{\{P(z_u|i)\}} + \overbrace{m(K_m - 1)}^{\{P(z_m|j)\}} \tag{9}$$

independent parameters in the model.

A realistic model would include, in addition, a prediction of the "missing elements" in the rating matrix, i.e., a model of why the entry was missing (a user failed to rate a movie they had seen, not seen but could, chosen not to see, etc.).

Estimating mixtures: the EM-algorithm

We have seen a number of different types of mixture models. The advantage of mixture models lies in their ability to incorporate and resolve ambiguities in the data, especially

in terms of unidentified sub-groups. However, we can make use of them only if we can estimate such models easily from the available data.

Complete data. The simplest way to understand how to estimate mixture models is to start by pretending that we knew all the sub-typing (component) assignments for each available data point. This is analogous to knowing the label for each example in a classification context. We don't actually know these (they are unobserved in the data) but solving the estimation problem in this context will help us later on.

Let's begin with the simple Gaussian mixture model in Eq.(1),

$$P(\mathbf{x}; \theta) = \sum_{j=1}^{m} P(j) N(\mathbf{x}; \mu_j, \Sigma_j) \tag{10}$$

and pretend that each observation $\mathbf{x}_1, \ldots, \mathbf{x}_n$ also had information about the component that was responsible for generating it, i.e., we also observed $j_1, \ldots, j_n$. This additional component information is convenient to include in the form of 0-1 assignments $\delta(j|t)$ where $\delta(j_t|t) = 1$ and $\delta(j|t) = 0$ for all $j \neq j_t$. The log-likelihood of this *complete data* is

$$
\begin{aligned}
l(\mathbf{x}_1, \ldots, \mathbf{x}_n, j_1, \ldots, j_n; \theta) &= \sum_{t=1}^{n} \log \left[P(j_t) N(\mathbf{x}_t; \mu_{j_t}, \Sigma_{j_t}) \right] \tag{11} \\
&= \sum_{t=1}^{n} \sum_{j=1}^{m} \delta(j|t) \log \left[P(j) N(\mathbf{x}_t; \mu_j, \Sigma_j) \right] \tag{12} \\
&= \sum_{j=1}^{m} \left(\sum_{t=1}^{n} \delta(j|t) \right) \log P(j) \\
&\quad + \sum_{j=1}^{m} \left(\sum_{t=1}^{n} \delta(j|t) \log N(\mathbf{x}_t; \mu_j, \Sigma_j) \right) \tag{13}
\end{aligned}
$$

It's important to note that in trying to maximize this log-likelihood, all the Gaussians can be estimated separately from each other. Put another way, because our pretend observations are "complete", we can estimate each component from only data pertaining to it; the problem of resolving which component should be responsible for which data points is not

present. As a result, the maximizing solution can be written in closed form:

$$\hat{P}(j) = \frac{\hat{n}(j)}{n}, \quad \text{where} \ \ \hat{n}(j) = \sum_{t=1}^{n} \delta(j|t) \tag{14}$$

$$\hat{\mu}_j = \frac{1}{\hat{n}(j)} \sum_{t=1]}^{n} \delta(j|t)\mathbf{x}_t \tag{15}$$

$$\hat{\Sigma}_j = \frac{1}{\hat{n}(j)} \sum_{t=1]}^{n} \delta(j|t)(\mathbf{x}_t - \hat{\mu}_j)(\mathbf{x}_t - \hat{\mu}_j)^T \tag{16}$$

In other words, the prior probabilities simply recover the empirical fractions from the "observed" $j_1, \ldots, j_n$, and each Gaussian component is estimated in the usual way (evaluating the empirical mean and the covariance) based on data points explicitly assigned to that component. So, the estimation of mixture models would be very easy if we knew the assignments $j_1, \ldots, j_n$.

Incomplete data. What changes if we don't know the assignments? We can always guess what the assignments should be based on the current setting of the parameters. Let $\theta^{(l)}$ denote the current (initial) parameters of the mixture distribution. Using these parameters, we can evaluate for each data point x_t the posterior probability that it was generated from component j:

$$P(j|\mathbf{x}_t, \theta^{(l)}) = \frac{P^{(l)}(j)N(\mathbf{x}_t; \mu_j^{(l)}, \Sigma_j^{(l)})}{\sum_{j'=1}^{m} P^{(l)}(j')N(\mathbf{x}_t; \mu_{j'}^{(l)}, \Sigma_{j'}^{(l)})} = \frac{P^{(l)}(j)N(\mathbf{x}_t; \mu_j^{(l)}, \Sigma_j^{(l)})}{P(\mathbf{x}_t; \theta^{(l)})} \tag{17}$$

Instead of using the 0-1 assignments $\delta(j|t)$ of data points to components we can use "soft assignments" $p^{(l)}(j|t) = P(j|\mathbf{x}_t, \theta^{(l)})$. By substituting these in the above closed form estimating equations we get an iterative algorithm for estimating Gaussian mixtures. The algorithm is iterative since the soft posterior assignments were evaluated based on the current setting of the parameters $\theta^{(l)}$ and may have to revised later on (once we have a better idea of where the clusters are in the data).

The resulting algorithm is known as the *Expectation Maximization algorithm* (EM for short) and applies to all mixture models and beyond. For Gaussian mixtures, the EM-algorithm can be written as

> **(Step 0)** Initialize the Gaussian mixture, i.e., specify $\theta^{(0)}$. A simple initialization consists of setting $P^{(0)}(j) = 1/m$, equating each $\mu_j^{(0)}$ with a randomly chosen data point, and making all $\Sigma_j^{(0)}$ equal to the overall data covariance.

(**E-step**) Evaluate the posterior assignment probabilities $p^{(l)}(j|t) = P(j|\mathbf{x}_t, \theta^{(l)})$ based on the current setting of the parameters $\theta^{(l)}$.

(**M-step**) Update the parameters according to

$$P^{(l+1)}(j) = \frac{\hat{n}(j)}{n}, \text{ where } \hat{n}(j) = \sum_{t=1}^{n} p^{(l)}(j|t) \tag{18}$$

$$\mu_j^{(l+1)} = \frac{1}{\hat{n}(j)} \sum_{t=1}^{n} p^{(l)}(j|t)\mathbf{x}_t \tag{19}$$

$$\Sigma_j^{(l+1)} = \frac{1}{\hat{n}(j)} \sum_{t=1}^{n} p^{(l)}(j|t)(\mathbf{x}_t - \mu_j^{(l+1)})(\mathbf{x}_t - \mu_j^{(l+1)})^T \tag{20}$$

Perhaps surprisingly, this iterative algorithm is guaranteed to converge and each iteration increases the log-likelihood of the data. In other words, if we write

$$l(D; \theta^{(l)}) = \sum_{t=1}^{n} P(\mathbf{x}_t; \theta^{(l)}) \tag{21}$$

then

$$l(D; \theta^{(0)}) < l(D; \theta^{(1)}) < l(D; \theta^{(2)}) < \dots \tag{22}$$

until convergence. The main downside of this algorithm is that we are only guaranteed to converge to a locally optimal solution where $d/d\theta\, l(D; \theta) = 0$. In other words, there could be a setting of the parameters for the mixture distribution that leads to a higher log-likelihood of the data[1]. For this reason, the algorithm is typically run multiple times (recall the random initialization of the means) so as to ensure we find a reasonably good solution, albeit perhaps only locally optimal.

Example. Consider a simple mixture of two Gaussians. Figure 2 demonstrates how the EM-algorithm changes the Gaussian components after each iteration. The ellipsoids specify one standard deviation distances from the Gaussian mean. The mixing proportions $P(j)$ are not visible in the figure. Note that it takes many iterations for the algorithm to resolve how to properly assign the data points to mixture components. At convergence, the assignments are still soft (not 0-1) but nevertheless clearly divide the responsibilities of the two Gaussian components across the clusters in the data.

---

[1]In fact, the highest likelihood for Gaussian mixtures is always ∞. This happens when one of the Gaussians concentrates around a single data point. We do not look for such solutions, however, and they can be removed by constraining the covariance matrices or via regularization. The real comparison is to a non-trivial mixture that achieves the highest log-likelihood.
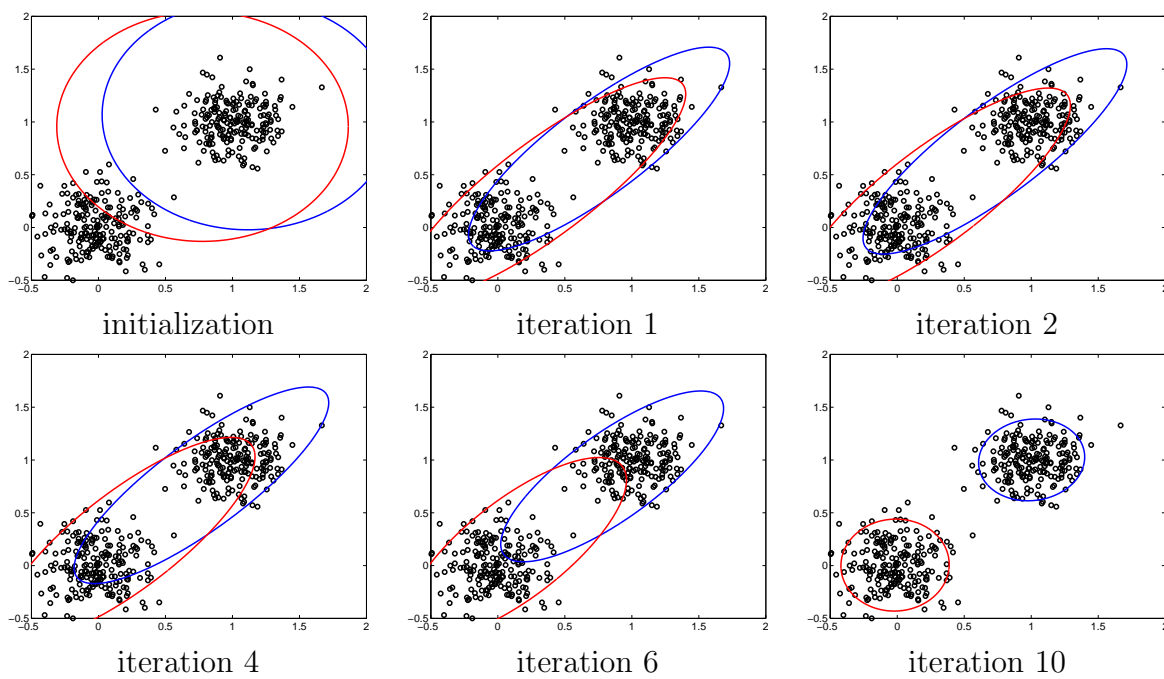
Figure 2: An example of the EM algorithm with a two-component mixture of Gaussians model.