Lecture 16: Mixture models

Roger Grosse and Nitish Srivastava

1 Learning goals

- Know what generative process is assumed in a mixture model, and what sort of data it is intended to model
- Be able to perform posterior inference in a mixture model, in particular
  - compute the posterior distribution over the latent variable
  - compute the posterior predictive distribution
- Be able to learn the parameters of a mixture model using the Expectation-Maximization (E-M) algorithm

2 Unsupervised learning

So far in this course, we've focused on supervised learning, where we assumed we had a set of training examples labeled with the correct output of the algorithm. We're going to shift focus now to unsupervised learning, where we don't know the right answer ahead of time. Instead, we have a collection of data, and we want to find interesting patterns in the data. Here are some examples:

- The neural probabilistic language model you implemented for Assignment 1 was a good example of unsupervised learning for two reasons:
  - One of the goals was to model the distribution over sentences, so that you could tell how "good" a sentence is based on its probability. This is unsupervised, since the goal is to model a distribution rather than to predict a target. This distribution might be used in the context of a speech recognition system, where you want to combine the evidence (the acoustic signal) with a prior over sentences.
  - The model learned word embeddings, which you could later analyze and visualize. The embeddings weren't given as a supervised signal to the algorithm, i.e. you didn't specify by hand which vector each word should correspond to, or which words should be close together in the embedding. Instead, this was learned directly from the data.

- You have a collection of scientific papers and you want to understand how the field's research focus has evolved over time. (For instance, maybe neural nets were a popular topic among AI researchers in the '80s and early '90s, then died out, and then came back with a vengeance around 2010.) You fit a model to a dataset of scientific papers which tries to identify different topics in the papers, and for each paper, automatically list the topics that it covers. The model wouldn't know what to call each topic, but you can attach labels manually by inspecting the frequent words in each topic. You then plot how often each topic is discussed in each year.

- You're a biologist interested in studying behavior, and you've collected lots of videos of bees. You want to "cluster" their behaviors into meaningful groups, so that you can look at each cluster and figure out what it means.

- You want to know how to reduce your energy consumption. You have a device which measures the total energy usage for your house (as a scalar value) for each hour over the course of a month. You want to decompose this signal into a sum of components which you can then try to match to various devices in your house (e.g. computer, refrigerator, washing machine), so that you can figure out which one is wasting the most electricity.

All of these are situations that call out for unsupervised learning. You don't know the right answer ahead of time, but just want to spot interesting patterns. In general, our strategy for unsupervised learning will be to formulate a probabilistic model which postulates certain unobserved random variables (called latent variables) which correspond to things we're interested in inferring. We then try to infer the values of the latent variables for all the data points, as well as parameters which relate the latent variables to the observations. In this lecture, we'll look at one type of latent variable model, namely mixture models.

3 Mixture models

In the previous lecture, we looked at some methods for learning probabilistic models which took the form of simple distributions (e.g. Bernoulli or Gaussian). But often the data we're trying to model is much more complex. For instance, it might be **multimodal**. This means that there are several different **modes**, or regions of high probability mass, and regions of smaller probability mass in between.

For instance, suppose we've collected the high temperatures for every day in March 2014 for both Toronto and Miami, Florida, but forgot to write down which city is associated with each temperature. The values are plotted in Figure 1; we can see the distribution has two modes.

In this situation, we might model the data in terms of a **mixture** of several **components**, where each component has a simple parametric form (such as a Gaussian). In other

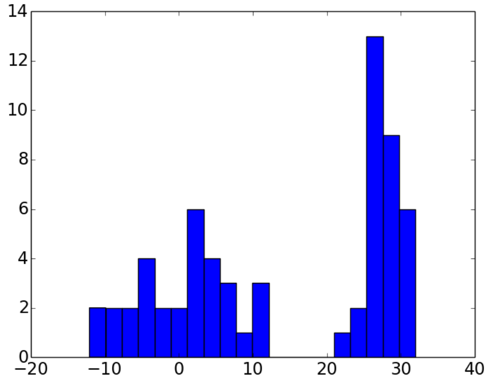

Figure 1: A histogram of daily high temperatures in °C for Toronto and Miami in March 2014. The distribution clearly has two modes.

words, we assume each data point belongs to one of the components, and we try to infer the distribution for each component separately. In this example, we happen to know that the two mixture components should correspond to the two cities. The model itself doesn't "know" anything about cities, though: this is just something we would read into it at the very end, when we analyze the results. In general, we won't always know precisely what meaning should be attached to the latent variables.

In order to represent this mathematically, we formulate the model in terms of **latent variables**, usually denoted z. These are variables which are *never* observed, and where we don't know the correct values in advance. They are roughly analogous to hidden units, in that the learning algorithm needs to figure out what they should represent, without a human specifying it by hand. Variables which are always observed, or even sometimes observed, are referred to as **observables**. In the above example, the city is the latent variable and the temperature is the observable.

In mixture models, the latent variable corresponds to the mixture component. It takes values in a discrete set, which we'll denote $\{1, \ldots, K\}$. (For now, assume K is fixed; we'll talk later about how to choose it.) In general, a mixture model assumes the data are generated by the following process: first we sample z, and then we sample the observables $\mathbf{x}$ from a distribution which depends on z, i.e.

$$p(z, \mathbf{x}) = p(z)\, p(\mathbf{x} \,|\, z).$$

In mixture models, $p(z)$ is always a multinomial distribution. $p(\mathbf{x} \,|\, z)$ can take a variety of parametric forms, but for this lecture we'll assume it's a Gaussian distribution. We refer to such a model as a **mixture of Gaussians.**

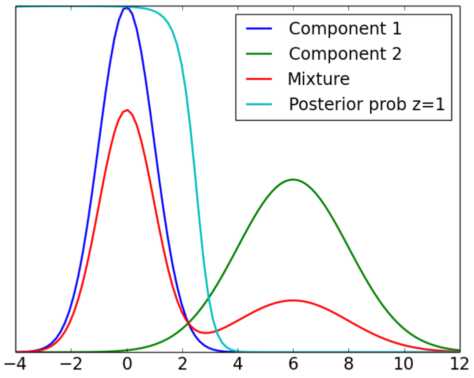

Figure 2: An example of a univariate mixture of Gaussians model.

Figure 2 shows an example of a mixture of Gaussians model with 2 components. It has the following generative process:

- With probability 0.7, choose component 1, otherwise choose component 2
- If we chose component 1, then sample x from a Gaussian with mean 0 and standard deviation 1
- If we chose component 2, then sample x from a Gaussian with mean 6 and standard deviation 2

This can be written in a more compact mathematical notation:

$$\begin{aligned} z &\sim \text{Multinomial}(0.7, 0.3) \qquad (1) \\ x \,|\, z = 1 &\sim \text{Gaussian}(0, 1) \qquad (2) \\ x \,|\, z = 2 &\sim \text{Gaussian}(6, 2) \qquad (3) \end{aligned}$$

For the general case,

$$\begin{aligned} z &\sim \text{Multinoimal}(\boldsymbol{\pi}) \qquad (4) \\ x \,|\, z = k &\sim \text{Gaussian}(\mu_k, \sigma_k). \qquad (5) \end{aligned}$$

Here, $\boldsymbol{\pi}$ is a vector of probabilities (i.e. nonnegative values which sum to 1) known as the **mixing proportions**.

In general, we can compute the probability density function (PDF) over $\mathbf{x}$ by **marginalizing out**, or summing out, z:

$$p(\mathbf{x}) = \sum_z p(z)\, p(\mathbf{x} \,|\, z) \tag{6}$$

$$= \sum_{k=1}^{K} \Pr(z = k)\, p(\mathbf{x} \,|\, z = k) \tag{7}$$

Note: Equations 6 and 7 are two different ways of writing the PDF; the first is more general (since it applies to other latent variable models), while the second emphasizes the meaning of the clustering model itself. In the example above, this gives us:

$$p(x) = 0.7 \cdot \text{Gaussian}(0, 1) + 0.3 \cdot \text{Gaussian}(6, 2). \tag{8}$$

This PDF is a **convex combination**, or **weighted average**, of the PDFs of the component distributions. The PDFs of the component distributions, as well as the mixture, are shown in Figure 2.

The general problem of grouping data points into clusters, where data points in the same cluster are more similar than data points in different clusters, is known as **clustering**. Learning a mixture model is one approach to clustering, but we should mention that there are a number of other approaches, most notably an algorithm called K-means[1].

4 Posterior inference

Before we talk about learning a mixture model, let's first talk about **posterior inference**. Here, we assume we've already chosen the parameters of the model. We want to infer, given a data point $\mathbf{x}$, which component it is likely to belong to. More precisely, we want to infer the posterior distribution $p(z \,|\, \mathbf{x})$. Just like in Bayesian parameter estimation, we can infer the posterior distribution using Bayes' Rule:

$$p(z \,|\, \mathbf{x}) \propto p(z)\, p(\mathbf{x} \,|\, z). \tag{9}$$

Recall that $\propto$ means "equal up to a constant." In other words, we can evaluate the right-hand side for all values of z, and then renormalize so that the values sum to 1.

[1] `http://en.wikipedia.org/wiki/K-means_clustering`

Example 1. Suppose we observe $x = 2$ in the model described above. We compute each of the required terms:

$$\begin{aligned}
\Pr(z = 1) &= 0.7 \\
\Pr(z = 2) &= 0.3 \\
p(x \,|\, z = 1) &= \text{Gaussian}(2; 0, 1) \\
&\approx 0.054 \\
p(x \,|\, z = 2) &= \text{Gaussian}(2; 6, 2) \\
&\approx 0.027
\end{aligned}$$

Therefore,

$$\begin{aligned}
\Pr(z = 1 \,|\, x) &= \frac{\Pr(z = 1)\, p(x \,|\, z = 1)}{\Pr(z = 1)\, p(x \,|\, z = 1) + \Pr(z = 2)\, p(x \,|\, z = 2)} \\
&= \frac{0.7 \cdot 0.054}{0.7 \cdot 0.054 + 0.3 \cdot 0.027} \\
&\approx 0.824
\end{aligned}$$

What if we repeat this calculation for different values of x? Figure 2 shows the posterior probability $\Pr(z = 1 \,|\, x)$ as a function of x.

Sometimes only some of the observables are actually observed. The others are said to be **missing**. One of the important uses of probabilistic models is to make predictions about the missing data given the observed data. You'll see an example of this in Assignment 3, namely **image completion**. There, you observe a subset of the pixels in an image, and want to make predictions about the rest of the image.

Suppose, for instance, that our observations are two-dimensional, and that we observe x_1 and want to make predictions about x_2. We can do this using the posterior predictive distribution:

$$p(x_2 \,|\, x_1) = \sum_z p(z \,|\, x_1)\, p(x_2 \,|\, z, x_1). \tag{10}$$

Sometimes we make the assumption that the x_i values are **conditionally independent given** z. This means that, if we are told the value of z, then x_1 and x_2 are independent. However, if z is unknown, then x_1 and x_2 are *not* independent, because x_1 gives information about z, which gives information about x_2. (For instance, the pixels in one half of an image are clearly not independent of the pixels in the other half. But maybe they are roughly independent, conditioned on a detailed description of everything going on in the image.)

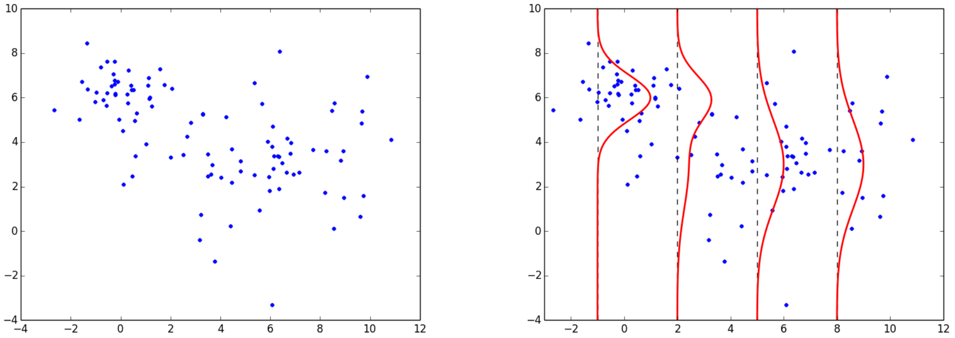

Figure 3: **Left:** Samples from the mixture of Gaussians model of Example 2. **Right:** The PDF of the posterior predictive distribution $p(x_2 \,|\, x_1)$, for various values of x_1.

Example 2. Consider the following two-dimensional mixture of Gaussians model, where x_1 and x_2 are conditionally independent given z:

$$\begin{aligned} z &\sim \text{Multinomial}(0.4, 0.6) \\ x_1 \,|\, z = 1 &\sim \text{Gaussian}(0, 1) \\ x_2 \,|\, z = 1 &\sim \text{Gaussian}(6, 1) \\ x_1 \,|\, z = 2 &\sim \text{Gaussian}(6, 2) \\ x_2 \,|\, z = 2 &\sim \text{Gaussian}(3, 2) \end{aligned}$$

The distribution is plotted in Figure 3. Suppose we observe $x_1 = 3$. We can compute the posterior distribution just like in the previous example:

$$\begin{aligned} \Pr(z = 1 \,|\, x_1) &= \frac{\Pr(z = 1)\, p(x \,|\, z = 1)}{\Pr(z = 1)\, p(x \,|\, z = 1) + \Pr(z = 2)\, p(x \,|\, z = 2)} \\ &= \frac{0.4 \cdot 0.0175}{0.4 \cdot 0.0175 + 0.6 \cdot 0.0431} \\ &= 0.213 \end{aligned}$$

Using the posterior, we compute the posterior predictive distribution:

$$\begin{aligned} p(x_2 \,|\, x_1) &= \Pr(z = 1 \,|\, x_1)\, p(x_2 \,|\, z = 1) + \Pr(z = 2 \,|\, x_1)\, p(x_2 \,|\, z = 2) \\ &= 0.213 \cdot \text{Gaussian}(x_2; 6, 1) + 0.787 \cdot \text{Gaussian}(x_2; 3, 2) \end{aligned}$$

Hence, the posterior predictive distribution $p(x_2 \,|\, x_1)$ is a mixture of two Gaussians, but with different mixing proportions than the original mixture model. Figure 3 shows the posterior predictive distribution for various values of x_1.

5 Learning

Now let's see how to learn the parameters of a mixture model. This doesn't immediately seem to have much to do with inference, but it'll turn out we need to do inference repeatedly in order to learn the parameters.

So far, we've been a bit vague about exactly what are the parameters of the Gaussian mixture model. We need two sets of parameters:

- The mean μ_k and standard deviation σ_k associated with each component k. (Other kinds of mixture models will have other sets of parameters.)
- The **mixing proportions** π_k, defined as $\Pr(z = k)$.

In the last lecture, we discussed three methods of parameter learning: maximum likelihood (ML), the maximum a-posteriori approximation (MAP), and the full Bayesian (FB) approach. For mixture models, we're going to focus on the first two. (There are fully Bayesian methods which look very much like the algorithm we'll discuss, but these are beyond the scope of the class.)

5.1 Log-likelihood derivatives

As always, if we want to compute the ML estimate, it's a good idea to start by computing the log-likelihood derivatives. Sometimes we can solve the problem analytically by setting the derivatives to 0, and otherwise, we can apply gradient ascent. In this section, we'll derive the log-likelihood derivatives. However, we won't end up using them directly; instead, we compute them as a way of motivating the E-M algorithm, which we discuss in the next section. If you're feeling lost, feel free to skip this section for now; we won't need it until next week, when we talk about Boltzmann machines.

Now we compute the log-likelihood derivative with respect to a parameter θ; this could stand for any of the parameters we want to learn, such as the mixing proportions or the mean or standard deviation of one of the components.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(\mathbf{x}) &= \frac{\mathrm{d}}{\mathrm{d}\theta}\log\sum_z p(z,\mathbf{x}) && (11)\\
&= \frac{\frac{\mathrm{d}}{\mathrm{d}\theta}\sum_z p(z,\mathbf{x})}{\sum_{z'} p(z',\mathbf{x})} && (12)\\
&= \frac{\sum_z \frac{\mathrm{d}}{\mathrm{d}\theta} p(z,\mathbf{x})}{\sum_{z'} p(z',\mathbf{x})} && (13)\\
&= \frac{\sum_z p(z,\mathbf{x})\,\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(z,\mathbf{x})}{\sum_{z'} p(z',\mathbf{x})} && (14)\\
&= \sum_z \left(\frac{p(z,\mathbf{x})}{\sum_{z'} p(z',\mathbf{x})}\right)\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(z,\mathbf{x}) && (15)\\
&= \sum_z p(z\,|\,\mathbf{x})\,\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(z,\mathbf{x}) && (16)\\
&= \mathbb{E}_{p(z\,|\,\mathbf{x})}\left[\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(z,\mathbf{x})\right] && (17)
\end{aligned}$$

This is pretty cool — the derivative of the marginal log-probability $p(\mathbf{x})$ is simply the expected derivative of the *joint* log-probability, where the expectation is with respect to the posterior distribution. This applies to any latent variable model, not just mixture models, since our derivation didn't rely on anything specific to mixture models.

That derivation was rather long, but observe that each of the steps was a completely mechanical substitution, except for the step labeled (14). In that step, we used the formula

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\log p(z,\mathbf{x}) = \frac{1}{p(z,\mathbf{x})}\frac{\mathrm{d}}{\mathrm{d}\theta}p(z,\mathbf{x}). \qquad (18)$$

You're probably used to using this formula to *get rid of* the log inside the derivative. We instead used it to *introduce* the log. We did this because it's much easier to deal with derivatives of log probabilities than derivatives of probabilities. This trick comes up quite a lot in probabilistic modeling, and we'll see it twice more: once when we talk about Boltzmann machines, and once when we talk about reinforcement learning.

If we can compute the expectation (17) summed over all training cases, that gives us the log-likelihood gradient.

> **Example 3.** Let's return to the mixture model from Example 1, where we're considering the training case $x = 2$. Let's compute the log-likelihood gradient with respect

to μ_1, the mean of the first mixture component.

$$\begin{aligned}
\frac{\partial}{\partial \mu_1} \log p(x) &= \mathbb{E}_{p(z \mid x)} \left[\frac{\partial}{\partial \mu_1} \log p(z, x) \right] && (19) \\
&= \mathbb{E}_{p(z \mid x)} \left[\frac{\partial}{\partial \mu_1} \log p(z) + \frac{\partial}{\partial \mu_1} \log p(x \mid z) \right] && (20) \\
&= \Pr(z = 1 \mid x) \left[\frac{\partial}{\partial \mu_1} \log \Pr(z = 1) + \frac{\partial}{\partial \mu_1} \log p(x \mid z = 1) \right] + \\
&\quad + \Pr(z = 2 \mid x) \left[\frac{\partial}{\partial \mu_1} \log \Pr(z = 2) + \frac{\partial}{\partial \mu_1} \log p(x \mid z = 2) \right] && (21) \\
&= 0.824 \cdot \left[0 + \frac{\partial}{\partial \mu_1} \log \mathrm{Gaussian}(x; \mu_1, \sigma_1) \right] + 0.176 \cdot [0 + 0] && (22) \\
&= 0.824 \cdot \frac{x - \mu_1}{\sigma_1^2} && (23) \\
&= 0.824 \cdot 2 && (24) \\
&= 1.648 && (25)
\end{aligned}$$

In the step (22), all of the terms except $\frac{\partial}{\partial \mu_1} \log p(x \mid z = 1)$ are zero because changing the parameters of the first component doesn't affect any of the other probabilities involved.

This only covers a single training case. But if we take the sum of this formula over all training cases, we find:

$$\frac{\partial \ell}{\partial \mu_1} = \sum_{i=1}^{N} \Pr(z^{(i)} = 1 \mid x^{(i)}) \cdot \frac{x^{(i)} - \mu_1}{\sigma_1^2} \qquad (26)$$

So essentially, we're taking a weighted sum of the conditional probability gradients for all the training cases, where each training case is weighted by its probability of belonging to the first component.

Since we can compute the log-likelihood gradient, we can maximize the log-likelihood using gradient ascent. This is, in fact, one way to learn the parameters of a mixture of Gaussians. However, we're not going to pursue this further, since we'll see an even more elegant and efficient learning algorithm for mixture models, called Expectation-Maximization.

5.2 Expectation-Maximization

As mentioned above, one way to fit a mixture of Gaussians model is with gradient ascent. But a mixture of Gaussians is built out of multinomial and Gaussian distributions, and we have closed-form solutions for maximum likelihood for these distributions. Shouldn't we be able to make use of these closed-form solutions when fitting mixture models?

Look at the log-likelihood derivative $\partial \ell / \partial \mu_1$ from Example 3, Eqn 26. It's a bit clunky to write out the conditional probabilities every time, so let's introduce the variables $r_k^{(i)} = \Pr(z^{(i)} = 1 \,|\, x^{(i)})$, which are called **responsibilities**. These say how strongly a data point "belongs" to each component. Eqn 26 can then be written in the more concise form

$$\frac{\partial \ell}{\partial \mu_1} = \sum_{i=1}^{N} r_k^{(i)} \cdot \frac{x^{(i)} - \mu_1}{\sigma_1^2} \tag{27}$$

It would be tempting to set this partial derivative to 0, and then solve for μ_1. This would give us:

$$\mu_1 = \frac{\sum_{i=1}^{N} r_1^{(i)} x^{(i)}}{\sum_{i=1}^{N} r_1^{(i)}}. \tag{28}$$

In other words, it's a weighted average of the training cases (essentially, the "center of gravity"), where the weights are given by the responsibilities. The problem is, we can't just solve for μ_1 in Eqn 27, because the responsibilities $r_k^{(i)}$ *depend* on μ_1. When we apply Eqn 28, the derivative in Eqn 26 won't be 0, because the posterior probabilities will have changed!

But that's OK. Even though the update Eqn 28 doesn't give the optimal parameters, it still turns out to be quite a good update. This is the basis of the **Expectation-Maximization (E-M) algorithm**. This algorithm gets its name because the update we just performed involves two steps: computing the responsibilities, and applying the maximum likelihood update with those responsibilities. More precisely, we apply the following algorithm:

Repeat until converged:

E-step.[2] Compute the expectations, or responsibilities, of the latent variables:

$$r_k^{(i)} \leftarrow \Pr(z^{(i)} = k \,|\, x^{(i)}) \tag{29}$$

M-step. Compute the maximum likelihood parameters given these responsibilities:

$$\boldsymbol{\theta} \leftarrow \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{k=1}^{K} r_k^{(i)} \left[\log \Pr(z^{(i)} = k) + \log p(\mathbf{x}^{(i)} \,|\, z^{(i)} = k) \right] \tag{30}$$

Typically, only one of the two terms within the bracket will depend on any given parameter, which simplifes the computations.

[2] This is called the E-step, or Expectation step, for the following reason: it's common to represent the mixture component z with a 1-of-K encoding, i.e. a K-dimensional vector where the *kth* component is 1 if the data point is assigned to cluster k, and zero otherwise. In this representation, the responsibilities are simply the expectations of the latent variables.

Generally, it takes a modest number of iterations (e.g. 20-50) to get close to a local optimum. Now let's derive the E-M algorithm for Gaussian mixture models.

Example 4. Let's derive the full E-M update for Gaussian mixtures. The E-step simply consists of computing the posterior probabilities, as we did in Example 1:

$$\begin{aligned} r_k^{(i)} &\leftarrow \Pr(z^{(i)} = k \mid x^{(i)}) \\ &= \frac{\Pr(z^{(i)} = k)\, p(x^{(i)} \mid z^{(i)} = k)}{\sum_{k'} \Pr(z^{(i)} = k')\, p(x^{(i)} \mid z^{(i)} = k')} \\ &= \frac{\pi_k \cdot \text{Gaussian}(x^{(i)}; \mu_k, \sigma_k)}{\sum_{k'} \pi_{k'} \cdot \text{Gaussian}(x^{(i)}; \mu_{k'}, \sigma_{k'})} \end{aligned}$$

Now let's turn to the M-step. Consider the mixing proportions π_k first. Before we go through the derivation, we can probably guess what the solution should be. A good estimate of the probability of an outcome is the proportion of times it appears, i.e. $\pi_k = N_k/N$, where N_k is the count for outcome k, and N is the number of observations. If we replace N_k with the sum of the responsibilities (which can be thought of as "fractional counts"), we should wind up with

$$\pi_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} r_k^{(i)}. \tag{31}$$

Let's see if we were right. Observe that the mixing proportions don't affect the second term inside the sum in Eqn 30, so we only need to worry about the first term, $\log \Pr(z^{(i)} = k)$. We want to maximize

$$\sum_{i=1}^{N} \sum_{k=1}^{K} r_k^{(i)} \log \Pr(z^{(i)} = k) = \sum_{i=1}^{N} \sum_{k=1}^{K} r_k^{(i)} \log \pi_k. \tag{32}$$

subject to the normalization constraint $\sum_k \pi_k = 1$. We can do this by computing the Lagrangian and setting its partial derivatives to zero. (If you don't know what the Lagrangian is, you can skip this step. We're not going to need it again.)

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{k=1}^{K} r_k^{(i)} \log \pi_k + \lambda \left(1 - \sum_{k=1}^{K} \pi_k \right) \tag{33}$$

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{\sum_{i=1}^{N} r_k^{(i)}}{\pi_k} - \lambda \tag{34}$$

Setting the partial derivative to zero, we see that

$$\lambda = \frac{\sum_{i=1}^{N} r_k^{(i)}}{\pi_k} \tag{35}$$

for *each* k. For this to be true, π_k must be proportional to $\sum_{i=1}^N r_k^{(i)}$. Therefore,

$$\pi_k \leftarrow \frac{\sum_{i=1}^N r_k^{(i)}}{\sum_{k'=1}^K \sum_{i=1}^N r_{k'}^{(i)}} = \frac{\sum_{i=1}^N r_k^{(i)}}{N}, \tag{36}$$

confirming our original guess, Eqn 31.

Finally, let's turn to the Gaussian parameters μ_k and σ_k. Observe that these parameters don't affect the first term inside the sum in Eqn 30, so we only need to worry about the second term, $\log p(x^{(i)} \mid z^{(i)} = k)$. Also, the parameters associated with mixture component k only affect the data points assigned to component k. Therefore, we want to maximize

$$\sum_{i=1}^N r_k^{(i)} \log p(x^{(i)} \mid z^{(i)} = k) = \sum_{i=1}^N r_k^{(i)} \log \text{Gaussian}(x^{(i)}; \mu_k, \sigma_k). \tag{37}$$

But this is exactly the maximum likelihood estimation problem for Gaussians, which we solved in the previous lecture. (The only difference here is the use of responsibilities.) Therefore, the updates are

$$\mu_k \leftarrow \frac{1}{\sum_{i=1}^N r_k^{(i)}} \sum_{i=1}^N r_k^{(i)} x^{(i)} \tag{38}$$

$$\sigma_k \leftarrow \frac{1}{\sum_{i=1}^N r_k^{(i)}} \sum_{i=1}^N r_k^{(i)} (x^{(i)} - \mu_k)^2 \tag{39}$$

These updates are visualized for the temperature data in Figure 4.

This example demonstrates a general recipe for deriving the M-step: apply the standard maximum likelihood formulas, except weight each data point according to its responsibility. This is true in general for mixture models, which is all we're going to talk about in this class. However, the E-M algorithm is much more far-ranging than this, and can be applied to many more latent variable models than just mixtures.

Despite our heuristic justification of the algorithm, it actually has quite strong guarantees. It's possible to show that each iteration of the algorithm increases the log-likelihood. (Note that this fact is not at all obvious just from the log-likelihood gradient (Eqn 17), since the algorithm takes such big steps that the first-order Taylor approximation around the current parameters certainly won't be very accurate.) Even though this is the only week of the class that doesn't involve neural nets, Professor Hinton left his footprints here as well: he and Radford Neal showed that each step of the algorithm is maximizing a particular lower bound on the log-likelihood. This analysis lets you generalize E-M to cases where you can only approximately compute the posterior over latent variables.

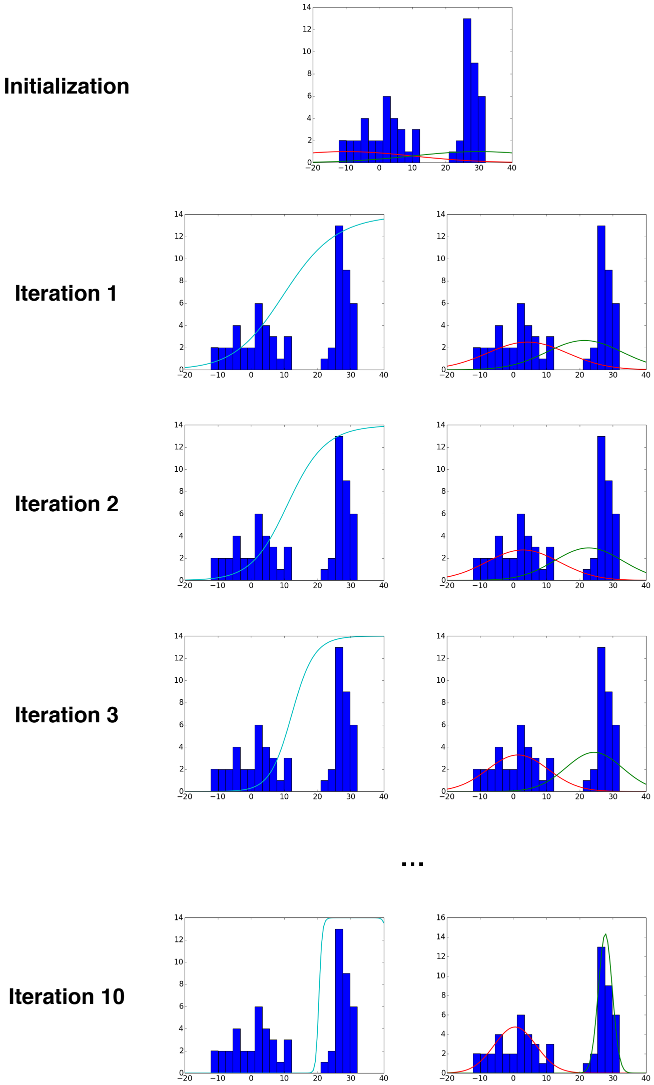

Figure 4: Visualization of the steps of the E-M algorithm applied to the temperature data. **Left:** E-step. The posterior probability $\Pr(z = 1 \mid x)$ is shown, as a function of x. **Right:** M-step. See Example 4 for the derivations of the update rules.

5.3 Odds and ends

Just a few more points about learning mixture models. First, how do you choose K, the number of components? If you choose K to small, you'll underfit the data, whereas if you choose it too large, you can overfit. One solution is: K is just a hyperparameter, and you can choose it using a validation set. I.e., you would choose the value of K which maximizes the average log-likelihood on the validation set. (There is also a very elegant framework called Bayesian nonparametrics which automatically adjusts the model complexity without requiring a validation set, but that is beyond the scope of the class.)

Second, how should you initialize the clusters? This is a tricky point, because some initializations are very bad. If you initialize all the components to the same parameters, they will never break apart because of symmetries in the model. If you choose a really bad initialization for one of the components, it may "die out" during E-M because the component doesn't assign high likelihood to any data points. There's no strategy that's guaranteed to work, but one good option is to initialize the different clusters to have random means and very broad standard deviation, just like we did for the temperatures example. Another option is to initialize the cluster assignments using K-means, another (non-probabilistic) clustering algorithm which tends to be easier to fit.

Finally, we discussed learning parameters with maximum likelihood. What about the MAP approximation? It turns out we can apply the E-M algorithm almost without modification. The only difference is that in the M-step, we apply the MAP updates rather than the ML ones. We'll walk you through this in Assignment 3.

6 Summary

- A mixture model is a kind of probabilistic model that assumes the data were generated by the following process:
  - Choose one of the mixture components at random.
  - Sample the data point from the distribution associated with that mixture component.
- If the mixture components have a nice form, we can do exact inference in the model. This includes:
  - inferring the posterior distribution over mixture components for a given data point
  - making predictions about missing input dimensions given the observed ones
- Mixture models have two sets of parameters:
  - the mixing proportions

  - the parameters associated with each component distribution

- The log-likelihood gradient is the expected gradient of the joint log-probability, where the expectation is with respect to the posterior over latent variables

- The E-M algorithm is a more efficient way to learn the parameters. This alternates between two steps:

  - E-step: Compute the responsibilities (defined as the posterior probabilities)
  - M-step: Compute the maximum likelihood parameters for each component, where each data point is weighted by the cluster's responsibility