CHAPTER 9

Clustering Using Probability Models

Clustering objects requires some notion of how similar they are. We have seen how to cluster using distance in feature space, which is a natural way of thinking about similarity. Another way to think about similarity is to ask whether two objects have high probability under the same probability model. This can be a convenient way of looking at things when it is easier to build probability models than it is to measure distances. It turns out to be a natural way of obtaining soft clustering weights (which emerge from the probability model). And it provides a framework for our first encounter with an extremely powerful and general algorithm, which you should see as a very aggressive generalization of k-means.

9.1 Mixture Models and Clustering

It is natural to think of clustering in the following way. The data was created by a collection of distinct probability models (one per cluster). For each data item, something (nature?) chose which model was to produce a point, and then an IID sample of that model produces the point. We see the points: we'd like to know what the models were, but (and this is crucial) we don't know which model produced which point. If we knew the models, it would be easy to decide which model produced which point. Similarly, if we knew which point went to which model, we could determine what the models were. One encounters this situation—or problems that can be mapped to this situation—again and again. It is very deeply embedded in clustering problems.

You should notice a resonance with k-means here. In k-means, if we knew the centers, which point belongs to which center would be easy; if we knew which point belongs to which center, the centers would be easy. We dealt with this situation quite effectively by repeatedly fixing one, then estimating the other. It is pretty clear that a natural algorithm for dealing with the probability models is to iterate between estimating which model gets which point, and the model parameters. This is the key to a standard, and very important, algorithm for estimation here, called **EM** (or **expectation maximization**, if you want the long version). I will develop this algorithm in two simple cases, and we will see it in a more general form later.

Notation: This topic lends itself to a glorious festival of indices, limits of sums and products, etc. I will do one example in quite gory detail; the other follows the same form, and for that we'll proceed more expeditiously. Writing the limits of sums or products explicitly is usually even more confusing than adopting a compact notation. When I write $\sum_i$ or $\prod_i$, I mean a sum (or product) over all values of i. When I write $\sum_{i,\hat{j}}$ or $\prod_{i,\hat{j}}$, I mean a sum (or product) over all values of i *except* for the jth item. I will write vectors, as usual, as $\mathbf{x}$; the ith such vector

D. Forsyth, *Applied Machine Learning*,
https://doi.org/10.1007/978-3-030-18114-7_9

in a collection is $\mathbf{x}_i$, and the kth component of the ith vector in a collection is x_{ik}. In what follows, I will construct a vector δ_i corresponding to the ith data item $\mathbf{x}_i$ (it will tell us what cluster that item belongs to). I will write δ to mean all the δ_i (one for each data item). The jth component of δ_i is δ_{ij}. When I write $\sum_{\delta_u}$, I mean a sum over all values that δ_u can take. When I write $\sum_{\delta}$, I mean a sum over all values that each δ can take. When I write $\sum_{\delta,\hat{\delta_v}}$, I mean a sum over all values that all δ can take, *omitting* all cases for the vth vector δ_v.

9.1.1 A Finite Mixture of Blobs

A blob of data points is quite easily modelled with a single normal distribution. Obtaining the parameters is straightforward (estimate the mean and covariance matrix with the usual expressions). Now imagine I have t blobs of data, and I know t. A normal distribution is likely a poor model, but I could think of the data as being produced by t normal distributions. I will assume that each normal distribution has a fixed, *known* covariance matrix Σ, but the mean of each is unknown. Because the covariance matrix is fixed, and *known*, we can compute a factorization $\Sigma = \mathcal{A}\mathcal{A}^T$. The factors must have full rank, because the covariance matrix must be positive definite. This means that we can apply $\mathcal{A}^{-1}$ to all the data, so that each blob covariance matrix (and so each normal distribution) is the identity.

Write μ_j for the mean of the jth normal distribution. We can model a distribution that consists of t distinct blobs by forming a weighted sum of the blobs, where the jth blob gets weight π_j. We ensure that $\sum_j \pi_j = 1$, so that we can think of the overall model as a probability distribution. We can then model the data as samples from the probability distribution

$$p(\mathbf{x}|\mu_1, \ldots, \mu_k, \pi_1, \ldots, \pi_k) = \sum_j \pi_j \left[\frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_j)^T(\mathbf{x} - \mu_j) \right) \right].$$

The way to think about this probability distribution is that a point is generated by first choosing one of the normal distributions (the jth is chosen with probability π_j), then generating a point from that distribution. This is a pretty natural model of clustered data. Each mean is the center of a blob. Blobs with many points in them have a high value of π_j, and blobs with a few points have a low value of π_j. We must now use the data points to estimate the values of π_j and μ_j (again, I am assuming that the blobs—and the normal distribution modelling each—have the identity as a covariance matrix). A distribution of this form is known as a **mixture of normal distributions**, and the π_j terms are usually called **mixing weights**.

Writing out the likelihood will reveal a problem: we have a product of many sums. The usual trick of taking the log will not work, because then you have a sum of logs of sums, which is hard to differentiate and hard to work with. A much more productive approach is to think about a set of hidden variables which tell us which blob each data item comes from. For the ith data item, we construct a vector δ_i. The jth component of this vector is δ_{ij}, where $\delta_{ij} = 1$ if $\mathbf{x}_i$ comes from blob (equivalently, normal distribution) j and zero otherwise. Notice there is exactly one 1 in δ_i, because each data item comes from one blob. I will write δ to mean all

the δ_i (one for each data item). Assume we know the values of these terms. I will write $\theta = (\mu_1, \ldots, \mu_k, \pi_1, \ldots, \pi_k)$ for the unknown parameters. Then we can write

$$p(\mathbf{x}_i|\delta_i, \theta) = \prod_j \left[\frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right]^{\delta_{ij}}$$

(because $\delta_{ij} = 1$ means that $\mathbf{x}_i$ comes from blob j, so the terms in the product are a collection of 1's and the probability we want). We also have

$$p(\delta_{ij} = 1|\theta) = \pi_j$$

allowing us to write

$$p(\delta_i|\theta) = \prod_j [\pi_j]^{\delta_{ij}}$$

(because this is the probability that we select blob j to produce a data item; again, the terms in the product are a collection of 1's and the probability we want). This means that

$$p(\mathbf{x}_i, \delta_i|\theta) = \prod_j \left\{ \left[\frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right] \pi_j \right\}^{\delta_{ij}}$$

and we can write a log-likelihood. The data are the observed values of $\mathbf{x}$ and δ (remember, we pretend we know these; I'll fix this in a moment), and the parameters are the unknown values of $\mu_1, \ldots, \mu_k$ and $\pi_1, \ldots, \pi_k$. We have

$$\begin{aligned} \mathcal{L}(\mu_1, \ldots, \mu_k, \pi_1, \ldots, \pi_k; \mathbf{x}, \delta) &= \mathcal{L}(\theta; \mathbf{x}, \delta) \\ &= \sum_{ij} \left\{ \left[\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right] + \log \pi_j \right\} \delta_{ij} \\ &\quad + K, \end{aligned}$$

where K is a constant that absorbs the normalizing constants for the normal distributions. You should check this expression. I have used the δ_{ij} as a "switch"—for one term, $\delta_{ij} = 1$ and the term in curly brackets is "on," and for all others that term is multiplied by zero. The problem with all this is that we don't know δ. I will deal with this when we have another example.

9.1.2 Topics and Topic Models

We have already seen that word counts expose similarities between documents (Sect. 6.3). We now assume that documents with similar word counts will come from the same **topic** (mostly, a term of art for cluster used in the natural language processing community). A really useful model is to assume that words are conditionally independent, conditioned on the topic. This means that, once you know the topic, words are IID samples of a multinomial distribution that is given by the topic (the **word probabilities** for that topic). If it helps, you can think of the topic as multi-sided die with a different word on each face. Each document has one

topic. If you know the topic, you make a document by rolling this die—which is likely not a fair die—some number of times.

This model of documents has problems. Word order doesn't matter in this model nor does where a word appears in a document or what words are near in the document and what others are far away. We've already seen that ignoring word order, word position, and neighbors can still produce useful representations (Sect. 6.3). Despite its problems, this model clusters documents rather well, is easy to work with, and is the basis for more complex models.

A single document is a set of word counts that is obtained by (a) selecting a topic then (b) drawing words as IID samples from that topic. We now have a collection of documents, and we want to know (a) what topic each document came from and (b) the word probabilities for each topic. Now imagine we know which document comes from which topic. Then we could estimate the word probabilities using the documents in each topic by simply counting. In turn, imagine we know the word probabilities for each topic. Then we could tell (at least in principle) which topic a document comes from by looking at the probability each topic generates the document, and choosing the topic with the highest probability. This procedure should strike you as being very like k-means, though the details have changed.

To construct a probabilistic model more formally, we will assume that a document is generated in two steps. We will have t topics. First, we choose a topic, choosing the jth topic with probability π_j. Then we will obtain a set of words by repeatedly drawing IID samples from that topic, and record the count of each word in a count vector. Each topic is a multinomial probability distribution. The vocabulary is d-dimensional. Write $\mathbf{p}_j$ for the d-dimensional vector of word probabilities for the jth topic. Now write $\mathbf{x}_i$ for the ith vector of word counts (there are N vectors in the collection). We assume that words are generated independently, conditioned on the topic. Write x_{ik} for the kth component of $\mathbf{x}_i$, and so on. Notice that $\mathbf{x}_i^T\mathbf{1}$ is the sum of entries in $\mathbf{x}_i$, and so the number of words in document i. Then the probability of observing the counts in $\mathbf{x}_i$ when the document was generated by topic j is

$$p(\mathbf{x}_i|\mathbf{p}_j) = \left(\frac{(\mathbf{x}_i^T\mathbf{1})!}{\prod_v x_{iv}!}\right)\prod_u p_{ju}^{x_{iu}}.$$

We can now write the probability of observing a document. Again, we write $\theta = (\mathbf{p}_1, \ldots, \mathbf{p}_t, \pi_1, \ldots, \pi_t)$ for the vector of unknown parameters. We have

$$\begin{aligned} p(\mathbf{x}_i|\theta) &= \sum_l p(\mathbf{x}_i|\text{topic is l})p(\text{topic is l}|\theta) \\ &= \sum_l \left[\left(\frac{(\mathbf{x}_i^T\mathbf{1})!}{\prod_v x_{iv}!}\right)\prod_u p_{lu}^{x_{iu}}\right]\pi_l. \end{aligned}$$

This model is widely called a **topic model**; be aware that there are many kinds of topic model, and this is a simple one. The expression should look unpromising, in a familiar way. If you write out a likelihood, you will see a product of sums; and if you write out a log-likelihood, you will see a sum of logs of sums. Neither is enticing. We could use the same trick we used for a mixture of normals. Write

$\delta_{ij} = 1$ if $\mathbf{x}_i$ comes from topic j, and $\delta_{ij} = 0$ otherwise. Then we have

$$p(\mathbf{x}_i|\delta_{ij} = 1, \theta) = \left[\left(\frac{(\mathbf{x}_i^T \mathbf{1})!}{\prod_v x_{iv}!} \right) \prod_u p_{ju}^{x_{iu}} \right]$$

(because $\delta_{ij} = 1$ means that $\mathbf{x}_i$ comes from topic j). This means we can write

$$p(\mathbf{x}_i|\delta_i, \theta) = \prod_j \left\{ \left[\left(\frac{(\mathbf{x}_i^T \mathbf{1})!}{\prod_v x_{iv}!} \right) \prod_u p_{ju}^{x_{iu}} \right] \right\}^{\delta_{ij}}$$

(because $\delta_{ij} = 1$ means that $\mathbf{x}_i$ comes from topic j, so the terms in the product are a collection of 1's and the probability we want). We also have

$$p(\delta_{ij} = 1|\theta) = \pi_j$$

(because this is the probability that we select topic j to produce a data item), allowing us to write

$$p(\delta_i|\theta) = \prod_j [\pi_j]^{\delta_{ij}}$$

(again, the terms in the product are a collection of 1's and the probability we want). This means that

$$p(\mathbf{x}_i, \delta_i|\theta) = \prod_j \left[\left(\frac{(\mathbf{x}_i^T \mathbf{1})!}{\prod_v x_{iv}!} \right) \prod_u \left(p_{ju}^{x_{iu}} \right) \pi_j \right]^{\delta_{ij}}$$

and we can write a log-likelihood. The data are the observed values of $\mathbf{x}$ and δ (remember, we pretend we know these for the moment), and the parameters are the unknown values collected in θ. We have

$$\mathcal{L}(\theta; \mathbf{x}, \delta) = \sum_i \left\{ \sum_j \left[\sum_u x_{iu} \log p_{ju} + \log \pi_j \right] \delta_{ij} \right\} + K,$$

where K is a term that contains all the

$$\log \left(\frac{(\mathbf{x}_i^T \mathbf{1})!}{\prod_v x_{iv}!} \right)$$

terms. This is of no interest to us, because it doesn't depend on any of our parameters. It takes a fixed value for each dataset. You should check this expression, noticing that, again, I have used the δ_{ij} as a "switch"—for one term, $\delta_{ij} = 1$ and the term in curly brackets is "on," and for all others that term is multiplied by zero. The problem with all this, as before, is that we don't know δ_{ij}. But there is a recipe.

9.2 The EM Algorithm

There is a straightforward, natural, and very powerful recipe for estimating θ for both models. In essence, we will average out the things we don't know. But this average will depend on our estimate of the parameters, so we will average, then re-estimate parameters, then re-average, and so on. If you lose track of what's going on here, think of the example of k-means with soft weights (Sect. 8.2.2; this is close to what the equations for the case of a mixture of normals will boil down to). In this analogy, the δ tell us which cluster center a data item came from. Because we don't know the values of the δ, we assume we have a set of cluster centers; these allow us to make an estimate of the δ; then we use this estimate to re-estimate the centers; and so on.

This is an instance of a general recipe. Recall we wrote θ for a vector of parameters. In the mixture of normals case, θ contained the means and the mixing weights; in the topic model case, it contained the topic distributions and the mixing weights. Assume we have an estimate of the value of this vector, say $\theta^{(n)}$. We could then compute $p(\delta|\theta^{(n)}, \mathbf{x})$. In the mixture of normals case, this is a guide to which example goes to which cluster. In the topic case, it is a guide to which example goes to which topic.

We could use this to compute the expected value of the likelihood with respect to δ. We compute

$$\mathcal{Q}(\theta; \theta^{(n)}) = \sum_{\delta} \mathcal{L}(\theta; \mathbf{x}, \delta) p(\delta|\theta^{(n)}, \mathbf{x}) = \mathbb{E}_{p(\delta|\theta^{(n)}, \mathbf{x})}[\mathcal{L}(\theta; \mathbf{x}, \delta)]$$

(where the sum is over all values of δ). Notice that $Q(\theta; \theta^{(n)})$ is a *function* of θ (because $\mathcal{L}$ was), but now does not have any unknown δ terms in it. This $Q(\theta; \theta^{(n)})$ encodes what we know about δ.

For example, assume that $p(\delta|\theta^{(n)}, \mathbf{x})$ has a single, narrow peak in it, at (say) $\delta = \delta^0$. In the mixture of normals case, this would mean that there is one allocation of points to clusters that is significantly better than all others, given $\theta^{(n)}$. For this example, $Q(\theta; \theta^{(n)})$ will be approximately $\mathcal{L}(\theta; \mathbf{x}, \delta^0)$.

Now assume that $p(\delta|\theta^{(n)}, \mathbf{x})$ is about uniform. In the mixture of normals case, this would mean that any particular allocation of points to clusters is about as good as any other. For this example, $Q(\theta; \theta^{(n)})$ will average $\mathcal{L}$ over all possible δ values with about the same weight for each.

We obtain the next estimate of θ by computing

$$\theta^{(n+1)} = \underset{\theta}{\text{argmax}}\ Q\left(\theta; \theta^{(n)}\right)$$

and iterate this procedure until it converges (which it does, though I shall not prove that). The algorithm I have described is extremely general and powerful, and is known as **expectation maximization** or (more usually) **EM**. The step where

we compute $Q(\theta;\theta^{(n)})$ is called the **E-step**; the step where we compute the new estimate of θ is known as the **M-step**.

One trick to be aware of: it is quite usual to ignore additive constants in the log-likelihood, because they have no effect. When you do the E-step, taking the expectation of a constant gets you a constant; in the M-step, the constant can't change the outcome. As a result, additive constants may disappear without notice (they do so regularly in the research literature). In the mixture of normals example, below, I've tried to keep track of them; for the mixture of multinomials, I've been looser.

9.2.1 Example: Mixture of Normals: The E-step

Now let us do the actual calculations for a mixture of normal distributions. The E-step requires a little work. We have

$$\mathcal{Q}\left(\theta;\theta^{(n)}\right) = \sum_{\delta} \mathcal{L}(\theta;\mathbf{x},\delta)p\left(\delta|\theta^{(n)},\mathbf{x}\right).$$

If you look at this expression, it should strike you as deeply worrying. There are a very large number of different possible values of δ. In this case, there are t^N cases (there is one δ_i for each data item, and each of these can have a one in each of t locations). It isn't obvious how we could compute this average.

But notice

$$p(\delta|\theta^{(n)},\mathbf{x}) = \frac{p(\delta,\mathbf{x}|\theta^{(n)})}{p(\mathbf{x}|\theta^{(n)})}$$

and let us deal with numerator and denominator separately. For the numerator, notice that the $\mathbf{x}_i$ and the δ_i are independent, identically distributed samples, so that

$$p(\delta,\mathbf{x}|\theta^{(n)}) = \prod_i p(\delta_i,\mathbf{x}_i|\theta^{(n)}).$$

The denominator is slightly more work. We have

$$\begin{aligned} p\left(\mathbf{x}|\theta^{(n)}\right) &= \sum_{\delta} p\left(\delta,\mathbf{x}|\theta^{(n)}\right) \\ &= \sum_{\delta}\left[\prod_i p\left(\delta_i,\mathbf{x}_i|\theta^{(n)}\right)\right] \\ &= \prod_i\left[\sum_{\delta_i} p\left(\delta_i,\mathbf{x}_i|\theta^{(n)}\right)\right]. \end{aligned}$$

You should check the last step; one natural thing to do is check with $N = 2$ and

$t = 2$. This means that we can write

$$\begin{aligned} p(\delta|\theta^{(n)}, \mathbf{x}) &= \frac{p\left(\delta, \mathbf{x}|\theta^{(n)}\right)}{p\left(\mathbf{x}|\theta^{(n)}\right)} \\ &= \frac{\prod_i p\left(\delta_i, \mathbf{x}_i|\theta^{(n)}\right)}{\prod_i \left[\sum_{\delta_i} p\left(\delta_i, \mathbf{x}_i|\theta^{(n)}\right)\right]} \\ &= \prod_i \frac{p\left(\delta_i, \mathbf{x}_i|\theta^{(n)}\right)}{\sum_{\delta_i} p\left(\delta_i, \mathbf{x}_i|\theta^{(n)}\right)} \\ &= \prod_i p\left(\delta_i|\mathbf{x}_i, \theta^{(n)}\right). \end{aligned}$$

Now we need to look at the log-likelihood. We have

$$\mathcal{L}(\theta; \mathbf{x}, \delta) = \sum_{ij} \left\{ \left[\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right] + \log \pi_j \right\} \delta_{ij} + K.$$

The K term is of no interest—it will result in a constant—but we will try to keep track of it. To simplify the equations we need to write, I will construct a t dimensional vector $\mathbf{c}_i$ for the ith data point. The jth component of this vector will be

$$\left\{ \left[\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right] + \log \pi_j \right\}$$

so we can write

$$\mathcal{L}(\theta; \mathbf{x}, \delta) = \sum_i \mathbf{c}_i^T \delta_i + K.$$

Now all this means that

$$\begin{aligned} \mathcal{Q}(\theta; \theta^{(n)}) &= \sum_\delta \mathcal{L}(\theta; \mathbf{x}, \delta) p(\delta|\theta^{(n)}, \mathbf{x}) \\ &= \sum_\delta \left(\sum_i \mathbf{c}_i^T \delta_i + K \right) p(\delta|\theta^{(n)}, \mathbf{x}) \\ &= \sum_\delta \left(\sum_i \mathbf{c}_i^T \delta_i + K \right) \prod_u p(\delta_u|\theta^{(n)}, \mathbf{x}) \\ &= \sum_\delta \left(\mathbf{c}_1^T \delta_1 \prod_u p(\delta_u|\theta^{(n)}, \mathbf{x}) + \ldots \mathbf{c}_N^T \delta_N \prod_u p(\delta_u|\theta^{(n)}, \mathbf{x}) \right). \end{aligned}$$

We can simplify further. We have that $\sum_{\delta_i} p(\delta_i|\mathbf{x}_i, \theta^{(n)}) = 1$, because this is a probability distribution. Notice that, for any index v

$$\begin{aligned} \sum_\delta \left(\mathbf{c}_v^T \delta_v \prod_u p(\delta_u|\theta^{(n)}, \mathbf{x}) \right) &= \sum_{\delta_v} \left(\mathbf{c}_v^T \delta_v p(\delta_v|\theta^{(n)}, \mathbf{x}) \right) \left[\sum_{\delta,\, \hat{\delta_v}} \prod_{u,\hat{v}} p(\delta_u|\theta^{(n)}, \mathbf{x}) \right] \\ &= \sum_{\delta_v} \left(\mathbf{c}_v^T \delta_v p(\delta_v|\theta^{(n)}, \mathbf{x}) \right). \end{aligned}$$

So we can write

$$\begin{aligned}\mathcal{Q}(\theta;\theta^{(n)}) &= \sum_{\delta}\mathcal{L}(\theta;\mathbf{x},\delta)p(\delta|\theta^{(n)},\mathbf{x})\\ &= \sum_{i}\left[\sum_{\delta_i}\mathbf{c}_i^T\delta_i p(\delta_i|\theta^{(n)},\mathbf{x})\right]+K\\ &= \sum_{i}\left[\left(\sum_{j}\left\{\left[\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T(\mathbf{x}_i-\mu_j)\right)+\log\pi_j\right]w_{ij}\right\}\right)\right]+K,\end{aligned}$$

where

$$\begin{aligned}w_{ij} &= 1p(\delta_{ij}=1|\theta^{(n)},\mathbf{x})+0p(\delta_{ij}=0|\theta^{(n)},\mathbf{x})\\ &= p(\delta_{ij}=1|\theta^{(n)},\mathbf{x}).\end{aligned}$$

Now

$$\begin{aligned}p(\delta_{ij}=1|\theta^{(n)},\mathbf{x}) &= \frac{p(\mathbf{x},\delta_{ij}=1|\theta^{(n)})}{p(\mathbf{x}|\theta^{(n)})}\\ &= \frac{p(\mathbf{x},\delta_{ij}=1|\theta^{(n)})}{\sum_l p(\mathbf{x},\delta_{il}=1|\theta^{(n)})}\\ &= \frac{p(\mathbf{x}_i,\delta_{ij}=1|\theta^{(n)})\prod_{u,\hat{i}}p(\mathbf{x}_u,\delta_u|\theta)}{\left(\sum_l p(\mathbf{x},\delta_{il}=1|\theta^{(n)})\right)\prod_{u,\hat{i}}p(\mathbf{x}_u,\delta_u|\theta)}\\ &= \frac{p(\mathbf{x}_i,\delta_{ij}=1|\theta^{(n)})}{\sum_l p(\mathbf{x},\delta_{il}=1|\theta^{(n)})}.\end{aligned}$$

If the last couple of steps puzzle you, remember we obtained $p(\mathbf{x},\delta|\theta)=\prod_i p(\mathbf{x}_i,\delta_i|\theta)$. Also, look closely at the denominator; it expresses the fact that the data must have come from somewhere. So the main question is to obtain $p(\mathbf{x}_i,\delta_{ij}=1|\theta^{(n)})$. But

$$\begin{aligned}p(\mathbf{x}_i,\delta_{ij}=1|\theta^{(n)}) &= p(\mathbf{x}_i|\delta_{ij}=1,\theta^{(n)})p(\delta_{ij}=1|\theta^{(n)})\\ &= \left[\frac{1}{\sqrt{(2\pi)^d}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T(\mathbf{x}_i-\mu_j)\right)\right]\pi_j.\end{aligned}$$

Substituting yields

$$p(\delta_{ij}=1|\theta^{(n)},\mathbf{x})=\frac{\left[\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T(\mathbf{x}_i-\mu_j)\right)\right]\pi_j}{\sum_k\left[\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_k)^T(\mathbf{x}_i-\mu_k)\right)\right]\pi_k}=w_{ij}.$$

9.2.2 Example: Mixture of Normals: The M-step

The M-step is more straightforward. Recall

$$\mathcal{Q}(\theta;\theta^{(n)})=\left(\sum_{ij}\left\{\left[\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T(\mathbf{x}_i-\mu_j)\right)\right]+\log\pi_j\right\}w_{ij}+K\right)$$

and we have to maximize this with respect to μ and π, and the terms w_{ij} are known. This maximization is easy. We compute

$$\mu_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}}{\sum_i w_{ij}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}}{N}.$$

You should check these expressions. When you do so, remember that, because π is a probability distribution, $\sum_j \pi_j = 1$ (otherwise you'll get the wrong answer). You need to either use a Lagrange multiplier or set one probability to (1 − all others).

9.2.3 Example: Topic Model: The E-step

We need to work out two steps. The E-step requires a little calculation. We have

$$\begin{aligned}
\mathcal{Q}(\theta; \theta^{(n)}) &= \sum_{\delta} \mathcal{L}(\theta; \mathbf{x}, \delta) p(\delta | \theta^{(n)}, \mathbf{x}) \\
&= \sum_{\delta} \left(\sum_{ij} \left\{ \left[\sum_u x_{iu} \log p_{ju} \right] + \log \pi_j \right\} \delta_{ij} \right) p(\delta | \theta^{(n)}, \mathbf{x}) \\
&= \left(\sum_{ij} \left\{ \left[\sum_k x_{i,k} \log p_{j,k} \right] + \log \pi_j \right\} w_{ij} \right).
\end{aligned}$$

Here the last two steps follow from the same considerations as in the mixture of normals. The $\mathbf{x}_i$ and δ_i are IID samples, and so the expectation simplifies as in that case. If you're uncertain, rewrite the steps of Sect. 9.2.1. The form of this Q function is the same as that (a sum of $\mathbf{c}_i^T \delta_i$ terms, but using a different expression for $\mathbf{c}_i$). In this case, as above,

$$\begin{aligned}
w_{ij} &= 1 p(\delta_{ij} = 1 | \theta^{(n)}, \mathbf{x}) + 0 p(\delta_{ij} = 0 | \theta^{(n)}, \mathbf{x}) \\
&= p(\delta_{ij} = 1 | \theta^{(n)}, \mathbf{x}).
\end{aligned}$$

Again, we have

$$\begin{aligned}
p(\delta_{ij} = 1 | \theta^{(n)}, \mathbf{x}) &= \frac{p(\mathbf{x}_i, \delta_{ij} = 1 | \theta^{(n)})}{p(\mathbf{x}_i | \theta^{(n)})} \\
&= \frac{p(\mathbf{x}_i, \delta_{ij} = 1 | \theta^{(n)})}{\sum_l p(\mathbf{x}_i, \delta_{il} = 1 | \theta^{(n)})}
\end{aligned}$$

and so the main question is to obtain $p(\mathbf{x}_i, \delta_{ij} = 1|\theta^{(n)})$. But

$$\begin{aligned} p(\mathbf{x}_i, \delta_{ij} = 1|\theta^{(n)}) &= p(\mathbf{x}_i|\delta_{ij} = 1, \theta^{(n)})p(\delta_{ij} = 1|\theta^{(n)}) \\ &= = \left[\prod_k p_{j,k}^{x_k}\right] \pi_j. \end{aligned}$$

Substituting yields

$$p(\delta_{ij} = 1|\theta^{(n)}, \mathbf{x}) = \frac{\left[\prod_k p_{j,k}^{x_k}\right] \pi_j}{\sum_l \left[\prod_k p_{l,k}^{x_k}\right] \pi_l}.$$

9.2.4 Example: Topic Model: The M-step

The M-step is more straightforward. Recall

$$\mathcal{Q}(\theta; \theta^{(n)}) = \left(\sum_{ij} \left\{ \left[\sum_k x_{i,k} \log p_{j,k} \right] + \log \pi_j \right\} w_{ij} \right)$$

and we have to maximize this with respect to μ and π, and the terms w_{ij} are known. This maximization is easy, but remember that the probabilities sum to one, so you need to either use a Lagrange multiplier or set one probability to (1 − all others). You should get

$$\mathbf{p}_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}}{\sum_i \mathbf{x}_i^T \mathbf{1} w_{ij}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}}{N}.$$

You should check these expressions by differentiating and setting to zero.

9.2.5 EM in Practice

The algorithm we have seen is amazingly powerful; I will use it again, ideally with less notation. One could reasonably ask whether it produces a "good" answer. Slightly surprisingly, the answer is yes. The algorithm produces a local maximum of $p(\mathbf{x}|\theta)$, the likelihood of the data conditioned on parameters. This is rather surprising because we engaged in all the activity with δ to avoid directly dealing with this likelihood (which in our cases was an unattractive product of sums). I did not prove this, but it's true anyway. I have summarized the general algorithm, and the two instances we studied, in boxes below for reference. There are some practical issues.

Procedure: 9.1 *EM*

Given a model with parameters θ, data $\mathbf{x}$, and missing data δ, which gives rise to a log-likelihood $\mathcal{L}(\theta; \mathbf{x}, \delta) = \log P(\mathbf{x}, \delta|\theta)$ and some initial estimate of parameters $\theta^{(1)}$, iterate

- **The E-step:** Obtain

$$\mathcal{Q}(\theta; \theta^{(n)}) = \mathbb{E}_{p(\delta|\theta^{(n)}, \mathbf{x})}[\mathcal{L}(\theta; \mathbf{x}, \delta)].$$

- **The M-step:** Compute

$$\theta^{(n+1)} = \underset{\theta}{\text{argmax}}\ Q(\theta; \theta^{(n)}).$$

Diagnose convergence by testing the size of the update to θ.

Procedure: 9.2 *EM for Mixtures of Normals: E-step*

Assume $\theta^{(n)} = (\mu_1, \ldots, \mu_t, \pi_1, \ldots, \pi_t)$ is known. Compute weights w_{ij} linking the ith data item to the jth cluster center, using

$$w_{ij}^{(n)} = \frac{\left[\exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mu_j^{(n)}\right)^T\left(\mathbf{x}_i - \mu_j^{(n)}\right)\right)\right]\pi_j^{(n)}}{\sum_k \left[\exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mu_k^{(n)}\right)^T\left(\mathbf{x}_i - \mu_k^{(n)}\right)\right)\right]\pi_k^{(n)}}.$$

Procedure: 9.3 *EM for Mixtures of Normals: M-step*

Assume $\theta^{(n)} = (\mu_1, \ldots, \mu_t, \pi_1, \ldots, \pi_t)$ and weights w_{ij} linking the ith data item to the jth cluster center are known. Then estimate

$$\mu_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}^{(n)}}{\sum_i w_{ij}^{(n)}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}^{(n)}}{N}.$$

Procedure: 9.4 *EM for Topic Models: E-step*

Assume $\theta^{(n)} = (\mathbf{p}_1, \ldots, \mathbf{p}_t, \pi_1, \ldots, \pi_t)$ is known. Compute weights $w_{ij}^{(n)}$ linking the ith data item to the jth cluster center, using

$$w_{ij}^{(n)} = \frac{\left[\prod_k \left(p_{j,k}^{(n)}\right)^{x_k}\right] \pi_j^{(n)}}{\sum_l \left[\prod_k \left(p_{j,k}^{(n)}\right)^{x_k}\right] \pi_l^{(n)}}.$$

Procedure: 9.5 *EM for Topic Models: M-step*

Assume $\theta^{(n)} = (\mathbf{p}_1, \ldots, \mathbf{p}_t, \pi_1, \ldots, \pi_t)$ and weights $w_{ij}^{(n)}$ linking the ith data item to the jth cluster center are known. Then estimate

$$\mathbf{p}_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}^{(n)}}{\sum_i \mathbf{x}_i^T \mathbf{1} w_{ij}^{(n)}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}^{(n)}}{N}.$$

First, how many cluster centers should there be? Mostly, the answer is a practical one. We are usually clustering data for a reason (vector quantization is a really good reason), and then we search for a k that yields the best results. Second, how should one start the iteration? This depends on the problem you want to solve, but for the two cases I have described, a rough clustering using k-means usually provides an excellent start. In the mixture of normals problem, you can take the cluster centers as initial values for the means, and the fraction of points in each cluster as initial values for the mixture weights. In the topic model problem, you can cluster the count vectors with k-means, use the overall counts within a cluster to get an initial estimate of the multinomial model probabilities, and use the fraction of documents within a cluster to get mixture weights. You need to be careful here, though. You really don't want to initialize a topic probability with a zero value for any word (otherwise no document containing that word can ever go into the cluster, which is a bit extreme). For our purposes, it will be enough to allocate a small value to each zero count, then adjust all the word probabilities to be sure they sum to one. More complicated approaches are possible.

Third, we need to avoid numerical problems in the implementation. Notice that you will be evaluating terms that look like

$$\frac{\pi_k e^{-(\mathbf{x}_i - \mu_k)^T (\mathbf{x}_i - \mu_k)/2}}{\sum_u \pi_u e^{-(\mathbf{x}_i - \mu_u)^T (\mathbf{x}_i - \mu_u)/2}}.$$

Imagine you have a point that is far from all cluster means. If you just blithely exponentiate the negative distances, you could find yourself dividing zero by zero, or a tiny number by a tiny number. This can lead to trouble. There's an easy alternative. Find the center the point is closest to. Now subtract the square of this distance (d^2_{min} for concreteness) from all the distances. Then evaluate

$$\frac{\pi_k e^{-\left[(\mathbf{x}_i-\mu_k)^T(\mathbf{x}_i-\mu_k)-d^2_{\text{min}}\right]/2}}{\sum_u \pi_u e^{-\left[(\mathbf{x}_i-\mu_u)^T(\mathbf{x}_i-\mu_u)-d^2_{\text{min}}\right]/2}}$$

which is a better way of estimating the same number (notice the $e^{-d^2_{\text{min}}/2}$ terms cancel top and bottom).

The last problem is more substantial. EM will get to a local minimum of $p(\mathbf{x}|\theta)$, but there might be more than one local minimum. For clustering problems, the usual case is there are lots of them. One doesn't really expect a clustering problem to have a single best solution, as opposed to a lot of quite good solutions. Points that are far from all clusters are a particular source of local minima; placing these points in different clusters yields somewhat different sets of cluster centers, each about as good as the other. It's not usual to worry much about this point. A natural strategy is to start the method in a variety of different places (use k-means with different start points), and choose the one that has the best value of Q when it has converged.

Remember This: *You should use the same approach to choosing the number of cluster centers with EM as you use with k-means (try a few different values, and see which yields the most useful clustering). You should initialize an EM clusterer with k-means, but be careful of initial probabilities that are zero when initializing a topic model. You should be careful when computing weights, as it is easy to have numerical problems. Finally, it's a good idea to start EM clustering at multiple start points.*

However, EM isn't magic. There are problems where computing the expectation is hard, typically because you have to sum over a large number of cases which don't have the nice independence structure that helped in the examples I showed. There are strategies for dealing with this problem—essentially, you can get away with an approximate expectation—but they're beyond our reach at present.

There is an important, rather embarrassing, secret about EM. In practice, it isn't usually that much better as a clustering algorithm than k-means. You can only really expect improvements in performance if it is really important that many points can make a contribution to multiple cluster centers, and this doesn't happen very often. For a dataset where this does apply, the data itself may not really be an IID draw from a mixture of normal distributions, so the weights you compute are only approximate. Usually, it is smart to start EM with k-means. Nonetheless, EM is an algorithm you should know, because it is very widely applied in other

situations, and because it can cluster data in situations where it isn't obvious how you compute distances.

Remember This: *EM clusterers aren't much better than k-means clusterers, but EM is very general. It is a procedure for estimating the parameters of a probability model in the presence of missing data; this is a scenario that occurs in many applications. In clustering, the missing data was which data item belonged to which cluster.*

9.3 You Should

9.3.1 Remember These Terms

9.3.2 Remember These Facts

9.3.3 Remember These Procedures

9.3.4 Be Able to

- Use EM to cluster points using a mixture of normals model.
- Cluster documents using EM and a topic model.

Problems

9.1. You will derive the expressions for the M-step for mixture of normal clustering. Recall

$$\mathcal{Q}(\theta;\theta^{(n)}) = \left(\sum_{ij} \left\{ \left[\left(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j) \right) \right] + \log \pi_j \right\} w_{ij} + K \right)$$

and we have to maximize this with respect to μ and π, and the terms w_{ij} are known. Show that

$$\mu_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}}{\sum_i w_{ij}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}}{N}$$

maximize $\mathcal{Q}$. When you do so, remember that, because π is a probability distribution, $\sum_j \pi_j = 1$ (otherwise you'll get the wrong answer). You need to either use a Lagrange multiplier or set one probability to (1 − all others).

9.2. You will derive the expressions for the M-step for topic models. Recall

$$\mathcal{Q}(\theta;\theta^{(n)}) = \left(\sum_{ij} \left\{ \left[\sum_k x_{i,k} \log p_{j,k} \right] + \log \pi_j \right\} w_{ij} \right)$$

and we have to maximize this with respect to μ and π, and the terms w_{ij} are known. Show that

$$\mathbf{p}_j^{(n+1)} = \frac{\sum_i \mathbf{x}_i w_{ij}}{\sum_i \mathbf{x}_i^T \mathbf{1} w_{ij}}$$

and

$$\pi_j^{(n+1)} = \frac{\sum_i w_{ij}}{N}.$$

When you do so, remember that, because π is a probability distribution, $\sum_j \pi_j = 1$ (otherwise you'll get the wrong answer). Furthermore, the $\mathbf{p}_j$ are all probability distributions. You need to either use Lagrange multipliers or set one probability to (1 − all others).

Programming Exercises

9.3. Image segmentation is an important application of clustering. One breaks an image into k segments, determined by color, texture, etc. These segments are obtained by clustering image pixels by some representation of the image around the pixel (color, texture, etc.) into k clusters. Then each pixel is assigned to the segment corresponding to its cluster center.

(a) Obtain a color image represented as three arrays (red, green, and blue). You should look for an image where there are long scale color gradients (a sunset is a good choice). Ensure that this image is represented so the darkest pixel takes the value $(0, 0, 0)$ and the lightest pixel takes the value $(1, 1, 1)$. Now assume the pixel values have covariance the identity matrix. Cluster its pixels into 10, 20, and 50 clusters, modelling the pixel values as a mixture of normal distributions and using EM. Display the image obtained by replacing each pixel with the mean of its cluster center. What do you see?

(b) The weights linking an image to a cluster center can be visualized as an image. For the case of 10 cluster centers, construct a figure showing the weights linking each pixel to each cluster center (all 10 images). You should notice that the weights linking a given pixel to each cluster center do not vary very much. Why?

(c) Now repeat the previous two subexercises, but now using $0.1 \times \mathcal{I}$ as the covariance matrix. Show the new set of weight maps. What has changed, and why?

(d) Now estimate the covariance of pixel values by assuming that pixels are normally distributed (this is somewhat in tension with assuming they're distributed as a mixture of normals, but it works). Again, cluster the image's pixels into 10, 20, and 50 clusters, modelling the pixel values as a mixture of normal distributions and using EM, but now assuming that each normal distribution has the covariance from your estimate. Display the image obtained by replacing each pixel with the mean of its cluster center. Compare this result from the result of the first exercise. What do you see?

9.4. If you have a careful eye, or you chose a picture fortunately, you will have noticed that the previous exercise can produce image segments that have many connected components. For some applications, this is fine, but for others, we want segments that are compact clumps of pixels. One way to achieve this is to represent each pixel with 5D vector, consisting of its RG and B values *and* its x and y coordinates. You then cluster these 5D vectors.

(a) Obtain a color image represented as three arrays (red, green, and blue). You should look for an image where there are many distinct colored objects (for example, a bowl of fruit). Ensure that this image is represented so the darkest pixel takes the value $(0, 0, 0)$ and the lightest pixel takes the value $(1, 1, 1)$. Represent the x and y coordinates of each pixel using the range 0 to 1 as well. Now assume the pixel RGB values have covariance 0.1 times the identity matrix, there is zero covariance between position and color, and the coordinates have covariance σ times the identity matrix where σ is a parameter we will modify. Cluster your image's pixels into 20, 50, and 100 clusters, with $\sigma = (0.01, 0.1, 1)$ (so 9 cases). Again, model the pixel values as a mixture of normal distributions and using EM. For each case, display the image obtained by replacing each pixel with the mean of its cluster center. What do you see?

9.5. EM has applications that don't look like clustering at first glance. Here is one. We will use EM to reject points that don't fit a line well (if you haven't seen least squares line fitting, this exercise isn't for you).

(a) Construct a dataset of 10 2D points which are IID samples from the following mixture distribution. Draw the x coordinate from the uniform distribution on the range $[0, 10]$. With probability 0.8, draw ξ a normal random variable with mean 0 and standard deviation 0.001 and form the y coordinate as $y = x + \xi$. With probability 0.2, draw the y coordinate from the uniform distribution on the range $[0, 10]$. Plot this dataset—you should see about eight points on a line with about two scattered points.

(b) Fit a least squares line to your dataset, and plot the result. It should be bad, because the scattered points may have a significant effect on the line. If you were unlucky, and drew a sample where there were no scattered points or where this line fits well, keep drawing datasets until you get one where the fit is poor.

(c) We will now use EM to fit a good line. Write $N(\mu, \sigma)$ for a normal distribution with mean μ and standard deviation σ, and $U(0, 10)$ for the uniform distribution on the range $0 - 10$. Model the y coordinate using the mixture model $P(y|a, b, \pi, x) = \pi N(ax + b, 0.001) + (1 - \pi)U(0, 10)$. Now associate a variable δ_i with the ith data point, where $\delta_i = 1$ if the data point comes from the line model and $\delta_i = 0$ otherwise. Write an expression for $P(y_i, \delta_i|a, b, \pi, x)$.

(d) Assume that $a^{(n)}$, $b^{(n)}$, and $\pi^{(n)}$ are known. Show that

$$Q\left(a, b, \pi; a^{(n)}, b^{(n)}, \pi^{(n)}\right) = -\sum_i w_i \frac{(ax_i + b - y_i)^2}{20.001^2} + (1 - w_i)(1/10) + K$$

(where K is a constant). Here

$$w_i = \mathbb{E}_{P(\delta_i|a^{(n)}, b^{(n)}, \pi^{(n)}, x)}[\delta_i].$$

(e) Show that

$$w_i = P(\delta_i|a^{(n)}, b^{(n)}, \pi^{(n)}, x) = \frac{\pi^{(n)} e^{-\frac{(a^{(n)}x_i + b^{(n)} - y_i)^2}{20.001^2}}}{\pi^{(n)} e^{-\frac{(a^{(n)}x_i + b^{(n)} - y_i)^2}{20.001^2}} + (1 - \pi^{(n)})\frac{1}{10}}.$$

(f) Now implement an EM algorithm using this information, and estimate the line for your data. You should try multiple start points. Do you get a better line fit? Why?

9.6. This is a fairly ambitious exercise. We will use the document clustering method of Sect. 9.1.2 to identify clusters of documents, which we will associate with topics. The 20 newsgroups dataset is a famous text dataset. It consists of posts collected from 20 different newsgroups. There are a variety of tricky data issues that this presents (for example, what aspects of the header should one ignore? should one reduce words to their stems, so "winning" goes to "win," "hugely" to "huge," and so on?). We will ignore these issues, and deal with a cleaned up version of the dataset. This consists of three items each for train and test: a document-word matrix, a set of labels, and a map. You can find this cleaned up version of the dataset at http://qwone.com/~jason/20Newsgroups/. You should look for the cleaned up version, identified as `20news-bydate-matlab.tgz` on that page. The usual task is to label a test article with which newsgroup it came from. The document-word matrix is a table of counts of how many times a particular word appears in a particular document. The collection of words is very large (53,975 distinct words), and most words do not appear in most documents, so most entries of this matrix are zero. The file `train.data` contains this matrix for a collection of training data; each row represents a distinct document (there are 11,269), and each column represents a distinct word.

(a) Cluster the rows of this matrix, using the method of Sect. 9.1.2, to get a set of cluster centers which we will identify as topics. **Hint:** Clustering all these points is a bit of a performance; check your code on small subsets of the data first, because the size of this dataset means that clustering the whole thing will be slow.

(b) You can now think of each cluster center as a document "type." Assume you have k clusters (topics). Represent each document by a k-dimensional vector. Each entry of the vector should be the negative log probability of the document under that cluster model. Now use this information to build a classifier that identifies the newsgroup using the vector. You'll need to use the file `train.label`, which will tell you what newsgroup a particular item comes from. I advise you to use a randomized decision forest, but other choices are plausible. Evaluate your classifier using the test data (`test.data` and `test.label`).