Mixture Models, Latent Variables and the EM Algorithm

36-350, Data Mining, Fall 2009

30 November 2009

Contents

Reading: *Principles of Data Mining*, sections 9.2, 8.4 and 9.6, in that order.

1 From Kernel Density Estimates to Mixture Models

Last time, we looked at kernel density estimation, where we approximate the true distribution by sticking a small copy of a kernel pdf at each observed data point and adding them up. With enough data, this comes arbitrarily close to any (reasonable) probability density, but it does have some drawbacks. Statistically, it labors under the curse of dimensionality. Computationally, we have to remember *all* of the data points, which is a lot. We saw similar problems when we looked at fully non-parametric regression, and then saw that both could be ameliorated by using things like additive models, which impose more constraints than, say, unrestricted kernel smoothing. Can we do something like that with density estimation?

Additive modeling for densities is not as common as it is for regression — it's harder to think of times when it would be natural and well-defined[1] — but we can do something with a similar flavor, which is to use a **mixture model**. One way to motivate this is to realize that, in many problems, lots of our data points are very similar to each other, and the differences between them are really just matters of chance or noise. Rather than remembering all of those noisy details, why not collapse those data points, and just remember their common distribution?

More formally, we say that a distribution f is a **mixture** of K **component** distributions $f_1, f_2, \ldots f_K$ if

$$f(x) = \sum_{k=1}^{K} \lambda_k f_k(x) \tag{1}$$

with the λ_k being the **mixing weights**, $\lambda_k > 0$, $\sum_k \lambda_k = 1$. Eq. 1 is a complete stochastic model, so it gives us a recipe for generating new data points: first pick a distribution, with probabilities given by the mixing weights, and then generate one observation according to that distribution. Symbolically,

$$\begin{aligned} Z &\sim \mathrm{Mult}(\lambda_1, \lambda_2, \ldots \lambda_K) & (2)\\ X|Z &\sim f_Z & (3) \end{aligned}$$

where I've introduced the discrete random variable Z which says which component X is drawn from.

I haven't said what kind of distribution the f_ks are. In principle, we could make these completely arbitrary, and we'd still have a perfectly good mixture model. In practice, a lot of effort is given over to **parametric mixture** models, where the f_k are all from the same parametric family, but with different parameters — for instance they might all be Gaussians with different centers and variances, or all Poisson distributions with different means, or all power laws with different exponents. (It's not strictly necessary that they all be of the same kind.) We'll write the parameter, or parameter vector, of the k^{th} component as θ_k, so the model becomes

$$f(x) = \sum_{k=1}^{K} \lambda_k f(x; \theta_k) \tag{4}$$

The over-all parameter vector of the mixture model is thus $\theta = (\lambda_1, \lambda_2, \ldots \lambda_K, \theta_1, \theta_2, \ldots \theta_K)$.

Let's consider two extremes. When $K = 1$, we have a simple parametric distribution, of the usual sort, and density estimation reduces to estimating the

[1] Remember that the integral of a probability density over all space must be 1, while the integral of a regression function doesn't have to be anything in particular. If we had an additive density, $f(x) = \sum_j f_j(x_j)$, ensuring normalization is going to be very tricky; we'd need $\sum_j \int f_j(x_j) dx_1 dx_2 dx_p = 1$. It would be easier to ensure normalization while making the *log*-density additive, but that assumes the features are independent of each other.

parameters, by maximum likelihood or whatever else we feel like. On the other hand when $K = n$, the number of observations, we have gone back towards kernel density estimation. If K is fixed as n grows, we still have a parametric model, and avoid the curse of dimensionality, but a mixture of (say) ten Gaussians is more flexible than a single Gaussian — thought it may still be the case that the true distribution just can't be written as a ten-Gaussian mixture.

1.1 Identifiability

Before we set about trying to estimate our probability models, we need to make sure that they are identifiable — that if we have distinct representations of the model, they make distinct observational claims. It is easy to let there be too many parameters, or the wrong choice of parameters, and lose identifiability. If there *are* distinct representations which are observationally equivalent, we either need to change our model, change our representation, or fix on a *unique* representation by some convention.

- With additive regression, $\mathbf{E}[Y|X = x] = \alpha + \sum_j f_j(x_j)$, we can add arbitrary constants so long as they cancel out. That is, we get the same predictions from $\alpha + c_0 + \sum j f_j(x_j) + c_j$ when $c_0 = -\sum_j c_j$. This is another model of the same form, $\alpha' + \sum_j f'_j(x_j)$, so it's not identifiable. We dealt with this by imposing the convention that $\alpha = \mathbf{E}[Y]$ and $\mathbf{E}[f_j(X_j)] = 0$ — we picked out a favorite, convenient representation from the infinite collection of equivalent representations.

- Linear regression becomes unidentifiable with collinear features. Collinearity is a good reason to not use linear regression (i.e., we change the model.)

- Factor analysis is unidentifiable because of the rotation problem. Some people respond by trying to fix on a particular representation, others just ignore it.

Two kinds of identification problems are common for mixture models; one is trivial and the other is fundamental. The trivial one is that we can always swap the labels of any two components with no effect on anything observable at all — if we decide that component number 1 is now component number 7 and vice versa, that doesn't change the distribution of X at all. This **label degeneracy** can be annoying, especially for some estimation algorithms, but that's the worst of it.

A more fundamental lack of identifiability happens when mixing two distributions from a parametric family just gives us a third distribution from the same family. For example, suppose we have a single binary feature, say an indicator for whether someone will pay back a credit card. We might think there are two kinds of customers, with high- and low- risk of not paying, and try to represent this as a mixture of binomial distribution. If we try this, we'll see that we've gotten a *single* binomial distribution with an intermediate risk of repayment. A mixture of binomials is always just another binomial. In fact, a mixture of multinomials is always just another multinomial.

2 Probabilistic Clustering

Back when we talked about clustering, I suggested that, ideally, knowing which cluster an observation came from would tell us as much as possible about the distribution of features. This means that differences in features between points from the same clusters should just be matters of chance. This view *exactly* corresponds to mixture models like Eq. 1; the hidden variable Z I introduced above in just the cluster label.

One of the very nice things about probabilistic clustering is that Eq. 1 actually *claims* something about what the data looks like; it says that it follows a certain distribution. We can check whether it does, and we can check whether *new* data follows this distribution. If it does, great; if not, if the predictions systematically fail, then the model is wrong. We can compare different probabilistic clusterings by how well they predict (say under cross-validation). Contrast this with k-means or hierarchical clustering: they make no predictions, and so we have no way of telling if they are right or wrong. Consequently, comparing different non-probabilistic clusterings is a lot harder!

In particular, probabilistic clustering gives us a sensible way of answering the question “how many clusters?” The best number of clusters to use is the number which will best generalize to future data. If we don’t want to wait around to get new data, we can approximate generalization performance by cross-validation, or by any other adaptive model selection procedure.

3 Estimating Parametric Mixture Models

From baby stats., we remember that it’s generally a good idea to estimate distributions using maximum likelihood, when we can. How could we do that here?

Remember that the likelihood is the probability (or probability density) of observing our data, as a function of the parameters. Assuming independent samples, that would be

$$\prod_{i=1}^{n} f(x_i; \theta) \tag{5}$$

for observations $x_1, x_2, \ldots x_n$. As always, we’ll use the logarithm to turn multiplication into addition:

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^{n} \log f(x_i; \theta) && (6) \\ &= \sum_{i=1}^{n} \log \sum_{k=1}^{K} \lambda_k f(x_i; \theta_k) && (7) \end{aligned}$$

Let's try taking the derivative of this with respect to one parameter, say θ_j.

$$\begin{aligned}
\frac{\partial \ell}{\partial \theta_j} &= \sum_{i=1}^{n} \frac{1}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \lambda_j \frac{\partial f(x_i;\theta_j)}{\partial \theta_j} && (8)\\
&= \sum_{i=1}^{n} \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \frac{1}{f(x_i;\theta_j)} \frac{\partial f(x_i;\theta_j)}{\partial \theta_j} && (9)\\
&= \sum_{i=1}^{n} \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \frac{\partial \log f(x_i;\theta_j)}{\partial \theta_j} && (10)
\end{aligned}$$

If we just had an ordinary parametric model, on the other hand, the derivative of the log-likelihood would be

$$\sum_{i=1}^{n} \frac{\partial \log f(x_i;\theta_j)}{\partial \theta_j} \tag{11}$$

So maximizing the likelihood for a mixture model is like doing a *weighted* likelihood maximization, where the weight of x_i depends on cluster, being

$$w_{ij} = \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \tag{12}$$

The problem is that these weights depend on the parameters we are trying to estimate!

Let's look at these weights w_{ij} a bit more. Remember that λ_j is the probability that the hidden class variable Z is j, so the numerator in the weights is the joint probability of getting $Z = j$ and $X = x_i$. The denominator is the marginal probability of getting $X = x_i$, so the ratio is the conditional probability of $Z = j$ given $X = x_i$,

$$w_{ij} = \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} = p(Z = j|X = x_i;\theta) \tag{13}$$

If we try to estimate the mixture model, then, we're doing weighted maximum likelihood, with weights given by the posterior cluster probabilities. These, to repeat, depend on the parameters we are trying to estimate, so there seems to be a vicious circle.

But, as the saying goes, one man's vicious circle is another man's successive approximation procedure. We have in fact seen something very much like this particular vicious circle already in cluster, in the form of the k-means algorithm. There, we want to assign each point to the cluster with the closest center (like calculating $p(Z = j|X = x_i;\theta)$ given θ), but a cluster's center depends on which points belong to it (like finding θ by maximizing the weighted likelihood).

In k-means, we solved the difficulty by alternating between assigning points to clusters and finding cluster centers. You could imagine doing something similar for mixture models: start with an initial guess about the component distributions; find out which component each point is most likely to have come from; re-estimate the components using only the points assigned to it, etc., until things converge. This corresponds to taking all the weights w_{ij} to be either 0 or 1; it also corresponds to a "hard" clustering procedure like k-means. However, it does not maximize the likelihood, since we've seen that to do so we need fractional weights.

What's called the EM algorithm is simply the obvious refinement of the hard assignment strategy.

1. Start with guesses about the mixture components $\theta_1, \theta_2, \ldots \theta_K$ and the mixing weights $\lambda_1, \ldots \lambda_K$.
2. Until nothing changes very much:
   (a) Using the current parameter guesses, calculate the weights w_{ij} (E-step)
   (b) Using the current weights, maximize the weighted likelihood to get new parameter estimates (M-step)
3. Return the final parameter estimates (including mixing proportions) and cluster probabilities

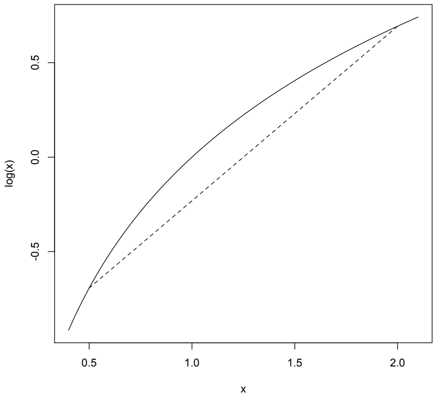
The M in "M-step" and "EM" stands for "maximization", which is pretty transparent. The E stands for "expectation", because it gives us the conditional probabilities of different values of Z, and probabilities are expectations of indicator functions. (In fact in some early applications, Z was binary, so one really was computing the expectation of Z.) The whole thing is also called the "expectation-maximization" algorithm.

3.1 More about the EM Algorithm

The EM algorithm turns out to be a general way of maximizing the likelihood when some variables are unobserved, and hence useful for other things besides mixture models. So in this section, where I try to explain why it works, I am going to be a bit more general abstract. (Also, it will actually cut down on notation.) I'll pack the whole sequence of observations $x_1, x_2, \ldots x_n$ into a single variable d (for "data"), and likewise the whole sequence of $z_1, z_2, \ldots z_n$ into h (for "hidden"). What we want to do is maximize

$$\ell(\theta) = \log p(d; \theta) = \log \sum_{h} p(d, h; \theta) \tag{14}$$

This is generally hard, because even if $p(d, h; \theta)$ has a nice parametric form, that is lost when we sum up over all possible values of h (as we saw above). The essential trick of the EM algorithm is to maximize not the log likelihood, but a *lower bound* on the log-likelihood, which is more tractable; we'll see that this lower bound is sometimes tight, i.e., coincides with the actual log-likelihood, and in particular does so at the global optimum.



```
curve(log(x),from=0.4,to=2.1)
segments(0.5,log(0.5),2,log(2),lty=2)
```

Figure 1: The logarithm is a concave function, i.e., the curve connecting any two points lies above the straight line doing so. Thus the average of logarithms is less than the logarithm of the average.

We can introduce an arbitrary[2] distribution on h, call it $q(h)$, and we'll

$$\begin{aligned}\ell(\theta) &= \log \sum_h p(d,h;\theta) && (15)\\ &= \log \sum_h \frac{q(h)}{q(h)} p(d,h;\theta) && (16)\\ &= \log \sum_h q(h) \frac{p(d,h;\theta)}{q(h)} && (17)\end{aligned}$$

So far so trivial.

Now we need a geometric fact about the logarithm function, which is that its curve is concave: if we take any two points on the curve and connect them by a straight line, the curve lies above the line (Figure 1). Algebraically, this means that

$$w \log t_1 + (1-w) \log t_2 \leq \log w t_1 + (1-w) t_2 \tag{18}$$

for any $0 \leq w \leq 1$, and any points $t_1, t_2 > 0$. Nor does this just hold for two points: for any r points $t_1, t_2, \ldots t_r > 0$, and any set of non-negative weights

[2] Well, almost arbitrary; it shouldn't give probability zero to value of h which has positive probability for all θ.

$\sum_{i=1}^{r} w_r = 1$,

$$\sum_{i=1}^{r} w_i \log t_i \leq \log \sum_{i=1}^{r} w_i t_i \tag{19}$$

In words: the log of the average is at least the average of the logs. This is called **Jensen's inequality**. So

$$\begin{aligned}\log \sum_h q(h) \frac{p(d,h;\theta)}{q(h)} &\geq \sum_h q(h) \log \frac{p(d,h;\theta)}{q(h)} &(20)\\ &\equiv J(q,\theta) &(21)\end{aligned}$$

We are bothering with this because we hope that it will be easier to maximize this lower bound on the likelihood than the actual likelihood, and the lower bound is reasonably tight. As to tightness, suppose that $q(h) = p(h|d;\theta)$. Then

$$\frac{p(d,h;\theta)}{q(h)} = \frac{p(d,h;\theta)}{p(h|d;\theta)} = \frac{p(d,h;\theta)}{p(h,d;\theta)/p(d;\theta)} = p(d;\theta) \tag{22}$$

no matter what h is. So with that choice of q, $J(q,\theta) = \ell(\theta)$ and the lower bound is tight. Also, since $J(q,\theta) \leq \ell(\theta)$, this choice of q maximizes J for fixed θ.

Here's how the EM algorithm goes in this formulation.

1. Start with an initial guess $\theta^{(0)}$ about the components and mixing weights.
2. Until nothing changes very much
   (a) E-step: $q^{(t)} = \operatorname{argmax}_q J(q,\theta^{(t)})$
   (b) M-step: $\theta^{(t+1)} = \operatorname{argmax}_\theta J(q^{(t)},\theta)$
3. Return final estimates of θ and q

The E and M steps are now nice and symmetric; both are about maximizing J. It's easy to see that, after the E step,

$$J(q^{(t)},\theta^{(t)}) \geq J(q^{(t-1)},\theta^{(t)}) \tag{23}$$

and that, after the M step,

$$J(q^{(t)},\theta^{(t+1)}) \geq J(q^{(t)},\theta^{(t)}) \tag{24}$$

Putting these two inequalities together,

$$\begin{aligned}J(q^{(t+1)},\theta^{(t+1)}) &\geq J(q^{(t)},\theta^{(t)}) &(25)\\ \ell(\theta^{(t+1)}) &\geq \ell(\theta^{(t)}) &(26)\end{aligned}$$

So each EM iteration can only improve the likelihood, guaranteeing convergence to a local maximum. Since it only guarantees a local maximum, it's a good idea to try a few different initial values of $\theta^{(0)}$ and take the best. All of which, again, is very like k-means.

We saw above that the maximization in the E step is just computing the posterior probability $p(h|d;\theta)$. What about the maximization in the M step?

$$\sum_h q(h) \log \frac{p(d,h;\theta)}{q(h)} = \sum_h q(h) \log p(d,h;\theta) - \sum_h q(h) \log q(h) \tag{27}$$

The second sum doesn't depend on θ at all, so it's irrelevant for maximizing, giving us back the optimization problem from the last section. This confirms that using the lower bound from Jensen's inequality hasn't yielded a different algorithm!

3.2 Further Reading on and Applications of EM

Like the textbook's, my presentation of the EM algorithm draws *heavily* on Neal and Hinton (1998).

Because it's so general, the EM algorithm is applied to *lots* of problems with missing data or latent variables. Traditional estimation methods for factor analysis, for example, can be replaced with EM. (Arguably, some of the older methods *were* versions of EM.) A common problem in time-series analysis and signal processing is that of "filtering" or "state estimation": there's an unknown signal S_t, which we want to know, but all we get to observe is some noisy, corrupted measurement, $X_t = h(S_t) + \eta_t$. (A historically important example of a "state" to be estimated from noisy measurements is "Where is our rocket and which way is it headed?" — see McGee and Schmidt, 1985.) This is solved by the EM algorithm, with the signal as the hidden variable; Fraser (2008) gives a really good introduction to such models and how they use EM.

Instead of just doing mixtures of densities, one can also do mixtures of predictive models, say mixtures of regressions, or mixtures of classifiers. The hidden variable Z here controls which regression function to use. A general form of this is what's known as a **mixture-of-experts** model (Jordan and Jacobs, 1994; Jacobs, 1997) — each predictive model is an "expert", and there can be a quite complicated set of hidden variables determining which expert to use when.

The EM algorithm is so useful and general that it has in fact been re-invented multiple times. The name "EM algorithm" comes from the statistics of mixture models in the late 1970s; in the time series literature it's been known since the 1960s as the "Baum-Welch" algorithm.

3.3 Topic Models and Probabilistic LSA

Mixture models over words provide an alternative to latent semantic indexing for document analysis. Instead of finding the principal components of the bag-of-words vectors, the idea is as follows. There are a certain number of **topics** which documents in the corpus can be about; each topic corresponds to a distribution over words. The distribution of words in a document is a mixture of the topic distributions. That is, one can generate a bag of words by first picking a topic according to a multinomial distribution (topic i occurs with probability λ_i), and then picking a word from that topic's distribution. The distribution of topics varies from document to document, and this is what's used, rather than projections on to the principal components, to summarize the document. This idea was, so far as I can tell, introduced by Hofmann (1999), who estimated everything by EM. **Latent Dirichlet allocation**, due to Blei and collaborators (Blei *et al.*, 2003) is an important variation which smoothes the topic distributions; there is a CRAN package called `lda`. Blei and Lafferty (2009) is a good recent review paper of the area.

4 Non-parametric Mixture Modeling

We could replace the M step of EM by some other way of estimating the distribution of each mixture component. This could be a fast-but-crude estimate of parameters (say a method-of-moments estimator if that's simpler than the MLE), or it could even be a non-parametric density estimator of the type we talked about last time. (Similarly for mixtures of regressions, etc.) Issues of dimensionality re-surface now, as well as convergence: because we're not, in general, increasing J at each step, it's harder to be sure that the algorithm will in fact converge. This is an active area of research.

5 R

There are several R packages which implement mixture models. The `mclust` package (`http://www.stat.washington.edu/mclust/`) is pretty much standard for Gaussian mixtures. One of the most recent and powerful is `mixtools` (Benaglia *et al.*, 2009), which, in addition to classic mixtures of parametric densities, handles mixtures of regressions and some kinds of non-parametric mixtures. The `FlexMix` package (Leisch, 2004) is (as the name implies) very good at flexibly handling complicated situations, though you have to do some programming to take advantage of this.

6 Exercises

Not to hand in.

1. Work through the E- step and M- step for a mixture of two Poisson distributions, *without* looking at the textbook. Then check your answer against section 8.4

2. Code up the EM algorithm for a mixture of K Gaussians, following section 8.4 in the textbook. Simulate data from $K = 3$ Gaussians. How well does your code assign data-points to components if you give it the actual

Gaussian parameters as your initial guess? If you give it other initial parameters?

3. Could you use mixture models to design a recommendation engine?

References

Benaglia, Tatiana, Didier Chauveau, David R. Hunter and Derek S. Young (2009). “mixtools: An R Package for Analyzing Mixture Models.” *Journal of Statistical Software*, **32**. URL `http://www.jstatsoft.org/v32/i06`.

Blei, David M. and John D. Lafferty (2009). “Topic Models.” In *Text Mining: Theory and Applications* (A. Srivastava and M. Sahami, eds.). London: Taylor and Francis. URL `http://www.cs.princeton.edu/~blei/papers/BleiLafferty2009.pdf`.

Blei, David M., Andrew Y. Ng and Michael I. Jordan (2003). “Latent Dirichlet Allocation.” *Journal of Machine Learning Research*, **3**: 993–1022. URL `http://jmlr.csail.mit.edu/papers/v3/blei03a.html`.

Fraser, Andrew M. (2008). *Hidden Markov Models and Dynamical Systems*. Philadelphia: SIAM Press. URL `http://www.siam.org/books/ot107/`.

Hofmann, Thomas (1999). “Probabilistic Latent Semantic Analysis.” In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference [UAI 1999]* (Kathryn Laskey and Henri Prade, eds.), pp. 289–296. San Francisco: Morgan Kaufmann. URL `http://www.cs.brown.edu/~th/papers/Hofmann-UAI99.pdf`.

Jacobs, Robert A. (1997). “Bias/Variance Analyses of Mixtures-of-Experts Architectures.” *Neural Computation*, **9**: 369–383.

Jordan, Michael I. and Robert A. Jacobs (1994). “Hierarchical Mixtures of Experts and the EM Algorithm.” *Neural Computation*, **6**: 181–214.

Leisch, Friedrich (2004). “FlexMix: A General Framework for Finite Mixture Models and Latent Class Regression in R.” *Journal of Statistical Software*, **11**. URL `http://www.jstatsoft.org/v11/i08`.

McGee, Leonard A. and Stanley F. Schmidt (1985). *Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry*. Tech. Rep. 86847, NASA Technical Memorandum. URL `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860003843_1986003843.pdf`.

Neal, Radford M. and Geoffrey E. Hinton (1998). “A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants.” In *Learning in Graphical Models* (Michael I. Jordan, ed.), pp. 355–368. Dordrecht: Kluwer Academic. URL `http://www.cs.toronto.edu/~radford/em.abstract.html`.