Chapter 20

Mixture Models

20.1 Two Routes to Mixture Models

20.1.1 From Factor Analysis to Mixture Models

In factor analysis, the origin myth is that we have a fairly small number, q of real variables which happen to be unobserved ("latent"), and the much larger number p of variables we do observe arise as linear combinations of these factors, plus noise. The mythology is that it's possible for us (or for Someone) to *continuously* adjust the latent variables, and the distribution of observables likewise changes continuously. What if the latent variables are not continuous but ordinal, or even categorical? The natural idea would be that each value of the latent variable would give a different distribution of the observables.

20.1.2 From Kernel Density Estimates to Mixture Models

We have also previously looked at kernel density estimation, where we approximate the true distribution by sticking a small ($\frac{1}{n}$ weight) 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

---

[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

do things to restrict density estimation. For instance, instead of putting a copy of the kernel at *every* point, we might pick a small number $K \ll n$ of points, which we feel are somehow typical or representative of the data, and put a copy of the kernel at each one (with weight $\frac{1}{K}$). This uses less memory, but it ignores the other data points, and lots of them are probably very similar to those points we're taking as prototypes. The differences between prototypes and many of their neighbors are 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? Different regions of the data space will have different shared distributions, but we can just combine them.

20.1.3 Mixture Models

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

$$f(x) = \sum_{k=1}^{K} \lambda_k f_k(x) \tag{20.1}$$

with the λ_k being the **mixing weights**, $\lambda_k > 0$, $\sum_k \lambda_k = 1$. Eq. 20.1 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,

$$
\begin{aligned}
Z &\sim \text{Mult}(\lambda_1, \lambda_2, \ldots \lambda_K) \tag{20.2}\\
X|Z &\sim f_Z \tag{20.3}
\end{aligned}
$$

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 kth component as θ_k, so the model becomes

$$f(x) = \sum_{k=1}^{K} \lambda_k f(x; \theta_k) \tag{20.4}$$

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 parameters, by maximum likelihood or whatever else we feel like. On the other hand when

---

independent of each other.

$K = n$, 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. So we have our usual bias-variance or accuracy-precision trade-off — using many components in the mixture lets us fit many distributions very accurately, with low approximation error or bias, but means we have more parameters and so we can't fit any one of them as precisely, and there's more variance in our estimates.

20.1.4   Geometry

In Chapter 18, we looked at principal components analysis, which finds linear structures with q space (lines, planes, hyper-planes, ...) which are good approximations to our p-dimensional data, $q \ll p$. In Chapter 19, we looked at factor analysis, where which imposes a statistical model for the distribution of the data around this q-dimensional plane (Gaussian noise), and a statistical model of the distribution of representative points on the plane (also Gaussian). This set-up is implied by the mythology of linear continuous latent variables, but can arise in other ways.

Now, we know from geometry that it takes $q+1$ points to define a q-dimensional plane, and that in general any $q + 1$ points on the plane will do. This means that if we use a mixture model with $q + 1$ components, we will also get data which clusters around a q-dimensional plane. Furthermore, by adjusting the mean of each component, and their relative weights, we can make the global mean of the mixture whatever we like. And we can even match the covariance matrix of any q-factor model by using a mixture with $q + 1$ components[2]. Now, this mixture distribution will hardly ever be exactly the same as the factor model's distribution — mixtures of Gaussians aren't Gaussian, the mixture will usually (but not always) be multimodal while the factor distribution is always unimodal — but it will have the same geometry, the same mean and the same covariances, so we will have to look beyond those to tell them apart. Which, frankly, people hardly ever do.

20.1.5   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 $\alpha + c_0 + \sum_j j f_j(x_j) + c_j$ 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

---

[2]See Bartholomew (1987, pp. 36–38). The proof is tedious algebraically.

the convention that $\alpha = \mathbf{E}[Y]$ and $\mathbf{E}\left[f_j(X_j)\right] = 0$ — 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 Bernoulli distribution. If we try this, we'll see that we've gotten a *single* Bernoulli distribution with an intermediate risk of repayment. A mixture of Bernoulli is always just another Bernoulli. More generally, a mixture of discrete distributions over any finite number of categories is just another distribution over those categories[3]

20.1.6 Probabilistic Clustering

Yet another way to view mixture models, which I hinted at when I talked about how they are a way of putting similar data points together into "clusters", where clusters are represented by, precisely, the component distributions. The idea is that all data points of the same type, belonging to the same cluster, are more or less equivalent and all come from the same distribution, and any differences between them are matters of chance. This view *exactly* corresponds to mixture models like Eq. 20.1; the hidden variable Z I introduced above in just the cluster label.

One of the very nice things about probabilistic clustering is that Eq. 20.1 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

---

[3]That is, a mixture of any two $n = 1$ multinomials is another $n = 1$ multinomial. This is not generally true when $n > 1$; for instance, a mixture of a Binom$(2, 0.75)$ and a Binom$(2, 0.25)$ is *not* a Binom$(2, p)$ for any p. (EXERCISE: show this.) However, both of those binomials is a distribution on $\{0, 1, 2\}$, and so is their mixture. This apparently trivial point actually leads into very deep topics, since it turns out that which models can be written as mixtures of others is strongly related to what properties of the data-generating process can actually be learned from data: see Lauritzen (1984). (Thanks to Bob Carpenter for pointing out an error in an earlier draft.)

the model is wrong. We can compare different probabilistic clusterings by how well they predict (say under cross-validation).[4]

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.

20.2   Estimating Parametric Mixture Models

From intro 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

$$\prod_{i=1}^{n} f(x_i; \theta) \tag{20.5}$$

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

$$\ell(\theta) = \sum_{i=1}^{n} \log f(x_i; \theta) \tag{20.6}$$

$$= \sum_{i=1}^{n} \log \sum_{k=1}^{K} \lambda_k f(x_i; \theta_k) \tag{20.7}$$

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

$$\frac{\partial \ell}{\partial \theta_j} = \sum_{i=1}^{n} \frac{1}{\sum_{k=1}^{K} \lambda_k f(x_i; \theta_k)} \lambda_j \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \tag{20.8}$$

$$= \sum_{i=1}^{n} \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i; \theta_k)} \frac{1}{f(x_i; \theta_j)} \frac{\partial f(x_i; \theta_j)}{\partial \theta_j} \tag{20.9}$$

$$= \sum_{i=1}^{n} \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i; \theta_k)} \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \tag{20.10}$$

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

$$\sum_{i=1}^{n} \frac{\partial \log f(x_i; \theta_j)}{\partial \theta_j} \tag{20.11}$$

---

[4]Contrast this with k-means or hierarchical clustering, which you may have seen in other classes: 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!

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

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i; \theta_k)} \tag{20.12}$$

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 $X = x_i$,

$$w_{ij} = \frac{\lambda_j f(x_i; \theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i; \theta_k)} = p(Z = j | X = x_i; \theta) \tag{20.13}$$

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. A crude way of doing this[5] would 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. 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 this "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.

---

[5] Related to what's called "k-means" clustering.

20.2.1   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

$$\ell(\theta) = \log p(d; \theta) = \log \sum_h p(d, h; \theta) \tag{20.14}$$

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.

   We can introduce an arbitrary[6] distribution on h, call it $q(h)$, and we'll

$$\ell(\theta) \quad = \quad \log \sum_h p(d, h; \theta) \tag{20.15}$$

$$= \quad \log \sum_h \frac{q(h)}{q(h)} p(d, h; \theta) \tag{20.16}$$

$$= \quad \log \sum_h q(h) \frac{p(d, h; \theta)}{q(h)} \tag{20.17}$$

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 20.1). Algebraically, this means that

$$w \log t_1 + (1 - w) \log t_2 \le \log w t_1 + (1 - w) t_2 \tag{20.18}$$

for any $0 \le w \le 1$, 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 $\sum_{i=1}^r w_r = 1$,
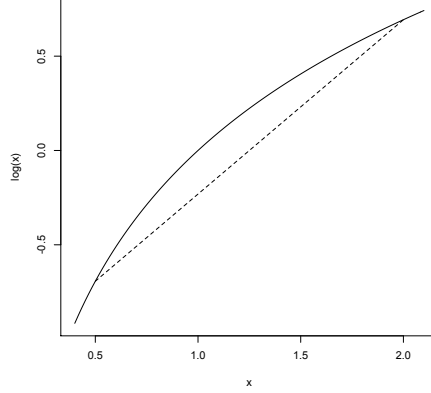
$$\sum_{i=1}^r w_i \log t_i \le \log \sum_{i=1}^r w_i t_i \tag{20.19}$$

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

$$\log \sum_h q(h) \frac{p(d, h; \theta)}{q(h)} \quad \ge \quad \sum_h q(h) \log \frac{p(d, h; \theta)}{q(h)} \tag{20.20}$$

$$\equiv \quad J(q, \theta) \tag{20.21}$$

---

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

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

Figure 20.1: 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 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

$$\frac{p(d,h;\theta)}{q(h)} = \frac{p(d,h;\theta)}{p(h|d;\theta)} = \frac{p(d,h;\theta)}{p(h,d;\theta)/p(d;\theta)} = p(d;\theta) \tag{20.22}$$

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: $q^{(t)} = \text{argmax}_q J(q, \theta^{(t)})$

   (b) M-step: $\theta^{(t+1)} = \text{argmax}_\theta J(q^{(t)}, \theta)$

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,

$$J(q^{(t)}, \theta^{(t)}) \geq J(q^{(t-1)}, \theta^{(t)}) \tag{20.23}$$

and that, after the M step,

$$J(q^{(t)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \tag{20.24}$$

Putting these two inequalities together,

$$J(q^{(t+1)}, \theta^{(t+1)}) \quad \geq \quad J(q^{(t)}, \theta^{(t)}) \tag{20.25}$$

$$\ell(\theta^{(t+1)}) \quad \geq \quad \ell(\theta^{(t)}) \tag{20.26}$$

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.

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?

$$\sum_h q(h) \log \frac{p(d, h; \theta)}{q(h)} = \sum_h q(h) \log p(d, h; \theta) - \sum_h q(h) \log q(h) \tag{20.27}$$

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!

20.2.2   Further Reading on and Applications of EM

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.

20.2.3 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.

20.3 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 in Chapter 15. (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.

20.4 Computation and Example: Snoqualmie Falls Revisited

20.4.1 Mixture Models in 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.

20.4.2   Fitting a Mixture of Gaussians to Real Data

Let's go back to the Snoqualmie Falls data set, last used in §13.3. There we built a system to forecast whether there would be precipitation on day t, on the basis of how much precipitation there was on day $t − 1$. Let's look at the distribution of the amount of precipitation on the wet days.

```
snoqualmie <- read.csv("snoqualmie.csv",header=FALSE)
snoqualmie.vector <- na.omit(unlist(snoqualmie))
snoq <- snoqualmie.vector[snoqualmie.vector > 0]
```

Figure 20.2 shows a histogram (with a fairly large number of bins), together with a simple kernel density estimate. This suggests that the distribution is rather skewed to the right, which is reinforced by the simple summary statistics

```
> summary(snoq)
   Min. 1st Qu.  Median   Mean 3rd Qu.    Max.
   1.00    6.00   19.00  32.28   44.00  463.00
```

Notice that the mean is larger than the median, and that the distance from the first quartile to the median is much smaller (13/100 of an inch of precipitation) than that from the median to the third quartile (25/100 of an inch). One way this could arise, of course, is if there are multiple types of wet days, each with a different characteristic distribution of precipitation.

We'll look at this by trying to fit Gaussian mixture models with varying numbers of components. We'll start by using a mixture of two Gaussians. We *could* code up the EM algorithm for fitting this mixture model from scratch, but instead we'll use the `mixtools` package.

```
library(mixtools)
snoq.k2 <- normalmixEM(snoq,k=2,maxit=100,epsilon=0.01)
```

The EM algorithm "runs until convergence", i.e., until things change so little that we don't care any more. For the implementation in `mixtools`, this means running until the log-likelihood changes by less than `epsilon`. The default tolerance for convergence is not 10^{-2}, as here, but 10^{-8}, which can take a very long time indeed. The algorithm also stops if we go over a maximum number of iterations, even if it has not converged, which by default is 1000; here I have dialed it down to 100 for safety's sake. What happens?

```
> snoq.k2 <- normalmixEM(snoq,k=2,maxit=100,epsilon=0.01)
number of iterations= 59
> summary(snoq.k2)
summary of normalmixEM object:
          comp 1    comp 2
lambda  0.557564  0.442436
mu     10.267390 60.012594
sigma   8.511383 44.998102
loglik at estimate:  -32681.21
```
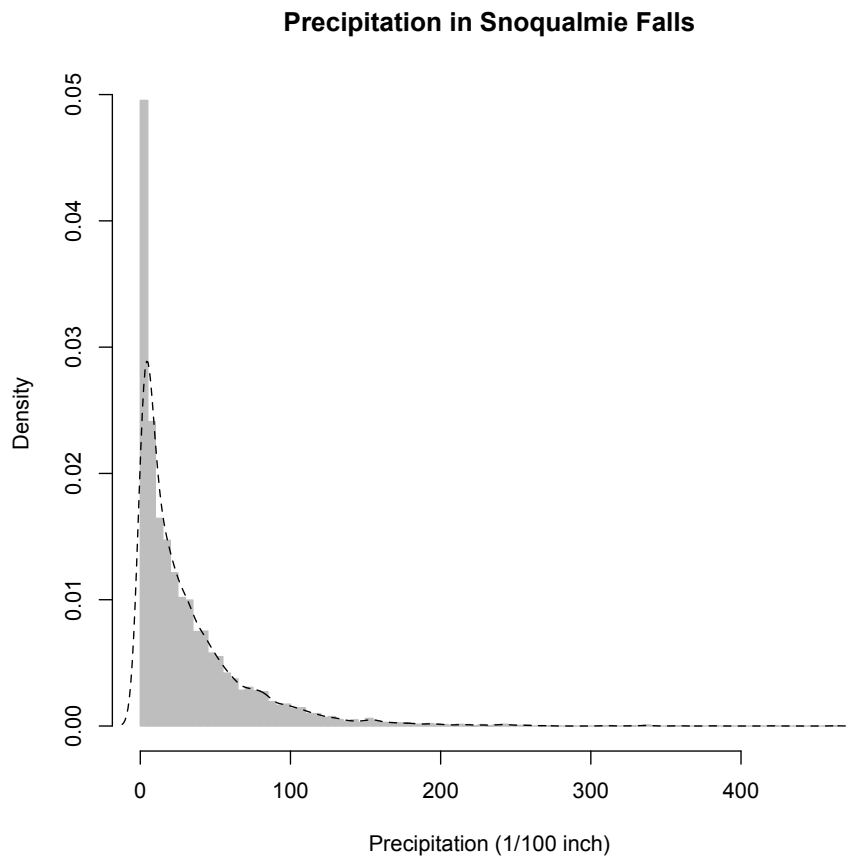
Precipitation in Snoqualmie Falls


```
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
     xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
```

Figure 20.2: Histogram (grey) for precipitation on wet days in Snoqualmie Falls. The dashed line is a kernel density estimate, which is not completely satisfactory. (It gives non-trivial probability to negative precipitation, for instance.)

There are two components, with weights (`lambda`) of about 0.56 and 0.44, two means (`mu`) and two standard deviations (`sigma`). The over-all log-likelihood, obtained after 59 iterations, is -32681.21. (Demanding convergence to $\pm 10^{-8}$ would thus have required the log-likelihood to change by less than one part in a trillion, which is quite excessive when we only have 6920 observations.)

   We can plot this along with the histogram of the data and the non-parametric density estimate. I'll write a little function for it.

```
plot.normal.components <- function(mixture,component.number,...) {
  curve(mixture$lambda[component.number] *
        dnorm(x,mean=mixture$mu[component.number],
        sd=mixture$sigma[component.number]), add=TRUE, ...)
}
```

This adds the density of a given component to the current plot, but scaled by the share it has in the mixture, so that it is visually comparable to the over-all density.
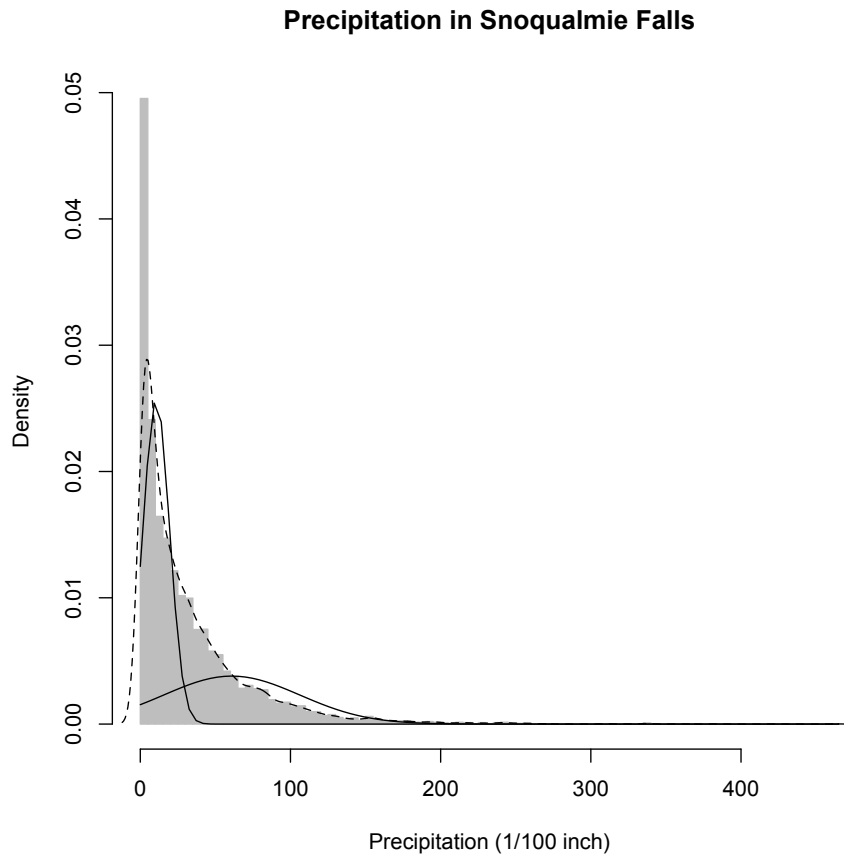
Precipitation in Snoqualmie Falls


```
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
     xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
sapply(1:2,plot.normal.components,mixture=snoq.k2)
```

Figure 20.3: As in the previous figure, plus the components of a mixture of two Gaussians, fitted to the data by the EM algorithm (dashed lines). These are scaled by the mixing weights of the components.

20.4.3   Calibration-checking for the Mixture

Examining the two-component mixture, it does not look altogether satisfactory — it seems to consistently give too much probability to days with about 1 inch of precipitation. Let's think about how we could check things like this.

When we looked at logistic regression, we saw how to check probability forecasts by checking calibration — events predicted to happen with probability p should in fact happen with frequency $\approx p$. Here we don't have a binary event, but we do have lots of probabilities. In particular, we have a cumulative distribution function $F(x)$, which tells us the probability that the precipitation is $\leq x$ on any given day. When x is continuous and has a continuous distribution, $F(x)$ should be uniformly distributed.[7] The CDF of a two-component mixture is

$$F(x) = \lambda_1 F_1(x) + \lambda_2 F_2(x) \tag{20.28}$$

and similarly for more components. A little R experimentation gives a function for computing the CDF of a Gaussian mixture:

```
pnormmix <- function(x,mixture) {
  lambda <- mixture$lambda
  k <- length(lambda)
  pnorm.from.mix <- function(x,component) {
    lambda[component]*pnorm(x,mean=mixture$mu[component],
                            sd=mixture$sigma[component])
  }
  pnorms <- sapply(1:k,pnorm.from.mix,x=x)
  return(rowSums(pnorms))
}
```

and so produce a plot like Figure 20.4.3. We do not have the tools to assess whether the *size* of the departure from the main diagonal is significant[8], but the fact that the errors are so very structured is rather suspicious.

---

[7]We saw this principle when we looked at generating random variables in Chapter 16.

[8]Though we could: the most straight-forward thing to do would be to simulate from the mixture, and repeat this with simulation output.
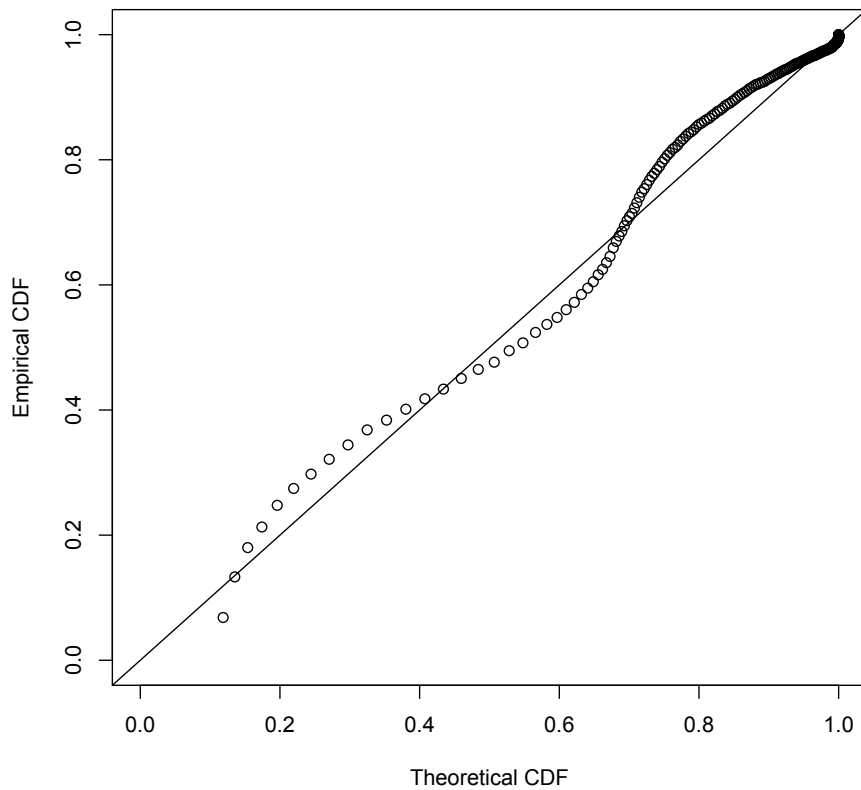
```
distinct.snoq <- sort(unique(snoq))
tcdfs <- pnormmix(distinct.snoq,mixture=snoq.k2)
ecdfs <- ecdf(snoq)(distinct.snoq)
plot(tcdfs,ecdfs,xlab="Theoretical CDF",ylab="Empirical CDF",xlim=c(0,1),
     ylim=c(0,1))
abline(0,1)
```

Figure 20.4: Calibration plot for the two-component Gaussian mixture. For each distinct value of precipitation x, we plot the fraction of days predicted by the mixture model to have $\leq x$ precipitation on the horizontal axis, versus the actual fraction of days $\leq x$.

20.4.4   Selecting the Number of Components by Cross-Validation

Since a two-component mixture seems iffy, we could consider using more compo-
nents. By going to three, four, etc. components, we improve our in-sample like-
lihood, but of course expose ourselves to the danger of over-fitting. Some sort of
model selection is called for. We could do cross-validation, or we could do hypothe-
sis testing. Let's try cross-validation first.

   We can already do fitting, but we need to calculate the log-likelihood on the held-
out data. As usual, let's write a function; in fact, let's write two.

```
dnormalmix <- function(x,mixture,log=FALSE) {
  lambda <- mixture$lambda
  k <- length(lambda)
  # Calculate share of likelihood for all data for one component
  like.component <- function(x,component) {
    lambda[component]*dnorm(x,mean=mixture$mu[component],
                              sd=mixture$sigma[component])
  }
  # Create array with likelihood shares from all components over all data
  likes <- sapply(1:k,like.component,x=x)
  # Add up contributions from components
  d <- rowSums(likes)
  if (log) {
    d <- log(d)
  }
  return(d)
}


loglike.normalmix <- function(x,mixture) {
  loglike <- dnormalmix(x,mixture,log=TRUE)
  return(sum(loglike))
}
```

To check that we haven't made a big mistake in the coding:

```
> loglike.normalmix(snoq,mixture=snoq.k2)
[1] -32681.2
```

which matches the log-likelihood reported by `summary(snoq.k2)`. But our func-
tion can be used on different data!

   We *could* do five-fold or ten-fold CV, but just to illustrate the approach we'll do
simple data-set splitting, where a randomly-selected half of the data is used to fit the
model, and half to test.

```
n <- length(snoq)
data.points <- 1:n
data.points <- sample(data.points) # Permute randomly
train <- data.points[1:floor(n/2)] # First random half is training
```

```
test <- data.points[-(1:floor(n/2))] # 2nd random half is testing
candidate.component.numbers <- 2:10
loglikes <- vector(length=1+length(candidate.component.numbers))
# k=1 needs special handling
mu<-mean(snoq[train]) # MLE of mean
sigma <- sd(snoq[train])*sqrt((n-1)/n) # MLE of standard deviation
loglikes[1] <- sum(dnorm(snoq[test],mu,sigma,log=TRUE))
for (k in candidate.component.numbers) {
  mixture <- normalmixEM(snoq[train],k=k,maxit=400,epsilon=1e-2)
  loglikes[k] <- loglike.normalmix(snoq[test],mixture=mixture)
}
```

When you run this, you will probably see a lot of warning messages saying "One of the variances is going to zero; trying new starting values." The issue is that we can give any one value of *x* arbitrarily high likelihood by centering a Gaussian there and letting its variance shrink towards zero. This is however generally considered unhelpful — it leads towards the pathologies that keep us from doing pure maximum likelihood estimation in non-parametric problems (Chapter 15) — so when that happens the code recognizes it and starts over.

If we look at the log-likelihoods, we see that there is a dramatic improvement with the first few components, and then things slow down a lot[9]:

```
> loglikes
 [1] -17656.86 -16427.83 -15808.77 -15588.44 -15446.77 -15386.74
 [7] -15339.25 -15325.63 -15314.22 -15315.88
```

(See also Figure 20.5). This favors nine components to the mixture. It looks like Figure 20.6. The calibration is now nearly perfect, at least on the training data (Figure 20.4.4).

---

[9]Notice that the numbers here are about half of the log-likelihood we calculated for the two-component mixture on the complete data. This is as it should be, because log-likelihood is proportional to the number of observations. (Why?) It's more like the *sum* of squared errors than the *mean* squared error. If we want something which is directly comparable across data sets of different size, we should use the log-likelihood per observation.
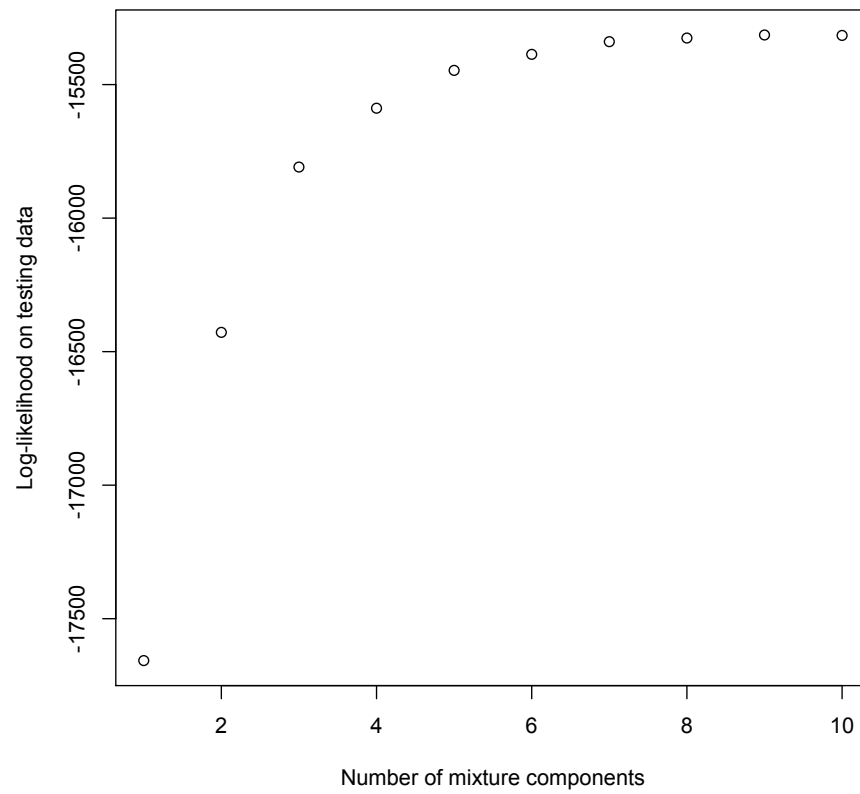
```
plot(x=1:10, y=loglikes,xlab="Number of mixture components",
     ylab="Log-likelihood on testing data")
```

Figure 20.5: Log-likelihoods of different sizes of mixture models, fit to a random half
of the data for training, and evaluated on the other half of the data for testing.

Precipitation in Snoqualmie Falls
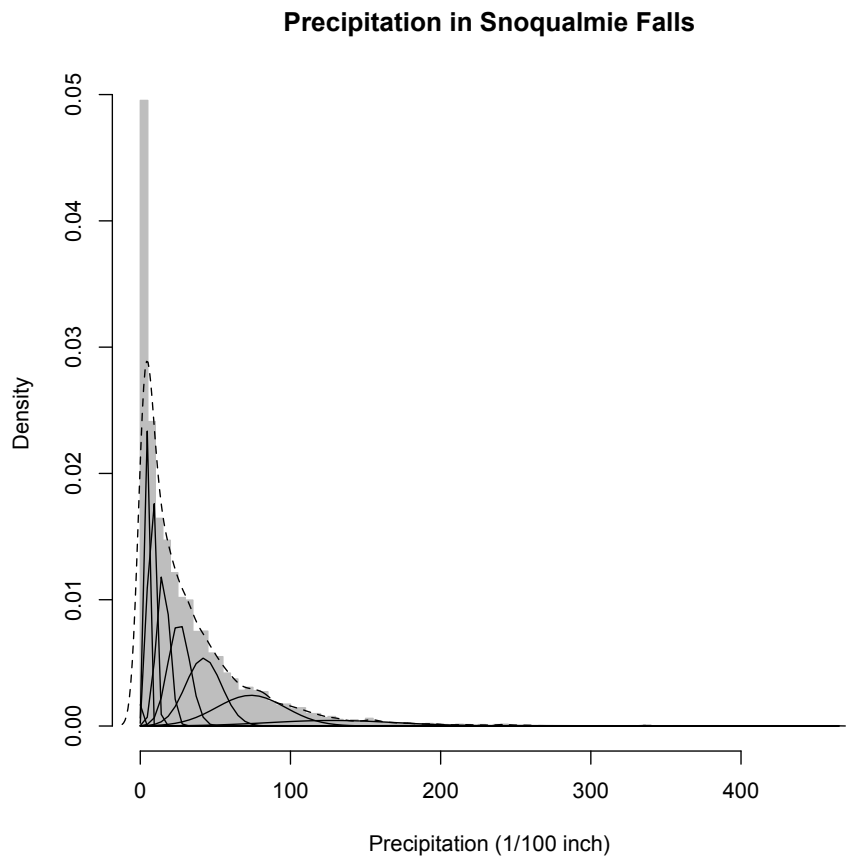

```
snoq.k9 <- normalmixEM(snoq,k=9,maxit=400,epsilon=1e-2)
plot(hist(snoq,breaks=101),col="grey",border="grey",freq=FALSE,
     xlab="Precipitation (1/100 inch)",main="Precipitation in Snoqualmie Falls")
lines(density(snoq),lty=2)
sapply(1:9,plot.normal.components,mixture=snoq.k9)
```

Figure 20.6: As in Figure 20.3, but using the nine-component Gaussian mixture.
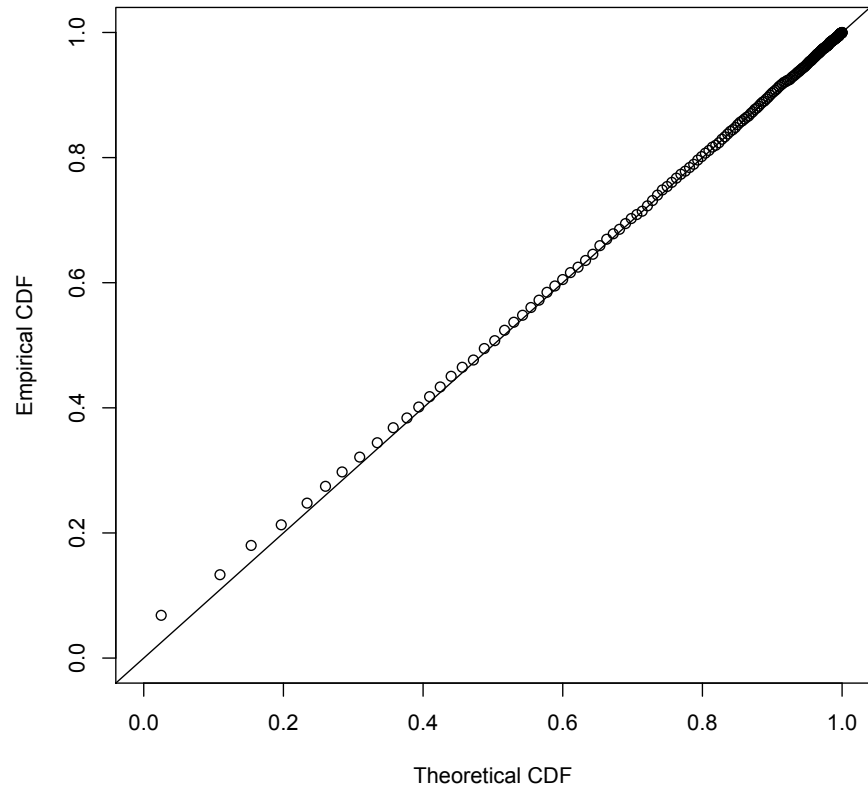
```
distinct.snoq <- sort(unique(snoq))
tcdfs <- pnormmix(distinct.snoq,mixture=snoq.k9)
ecdfs <- ecdf(snoq)(distinct.snoq)
plot(tcdfs,ecdfs,xlab="Theoretical CDF",ylab="Empirical CDF",xlim=c(0,1),
     ylim=c(0,1))
abline(0,1)
```

Figure 20.7: Calibration plot for the nine-component Gaussian mixture.

20.4.5 Interpreting the Mixture Components, or Not

The components of the mixture are far from arbitrary. It appears from Figure 20.6 that as the mean increases, so does the variance. This impression is confirmed from Figure 20.8. Now it *could* be that there really are nine types of rainy days in Snoqualmie Falls which just so happen to have this pattern of distributions, but this seems a bit suspicious — as though the mixture is trying to use Gaussians systematically to approximate a fundamentally different distribution, rather than get at something which really is composed of nine distinct Gaussians. This judgment relies on our scientific understanding of the weather, which makes us surprised by seeing a pattern like this in the parameters. (Calling this "scientific knowledge" is a bit excessive, but you get the idea.) Of course we are sometimes wrong about things like this, so it is certainly not conclusive. Maybe there really *are* nine types of days, each with a Gaussian distribution, and some subtle meteorological reason why their means and variances should be linked like this. For that matter, maybe our understanding of meteorology is wrong.

There are two directions to take this: the purely statistical one, and the substantive one.

On the purely statistical side, if all we care about is being able to describe the distribution of the data and to predict future precipitation, then it doesn't really matter whether the nine-component Gaussian mixture is true in any ultimate sense. Cross-validation picked nine components not because there really are nine types of days, but because a nine-component model had the best trade-off between approximation bias and estimation variance. The selected mixture gives a pretty good account of itself, nearly the same as the kernel density estimate (Figure 20.9). It requires 26 parameters[10], which may seem like a lot, but the kernel density estimate requires keeping around all 6920 data points plus a bandwidth. On sheer economy, the mixture then has a lot to recommend it.

On the substantive side, there are various things we could do to check the idea that wet days really do divide into nine types. These are going to be informed by our background knowledge about the weather. One of the things we know, for example, is that weather patterns more or less repeat in an annual cycle, and that different types of weather are more common in some parts of the year than in others. If, for example, we consistently find type 6 days in August, that suggests that is at least compatible with these being real, meteorological patterns, and not just approximation artifacts.

Let's try to look into this visually. `snoq.k9$posterior` is a 6920 × 9 array which gives the probability for each day to belong to each class. I'll boil this down to assigning each day to its most probable class:

```
day.classes <- apply(snoq.k9$posterior,1,which.max)
```

We can't just plot this and hope to see any useful patterns, because we want to see stuff recurring every year, and we've stripped out the dry days, the division into years, the padding to handle leap-days, etc. Fortunately, `snoqualmie` has all that, so we'll make a copy of that and edit `day.classes` into it.

---

[10]A mean and a standard deviation for each of nine components (=18 parameters), plus mixing weights (nine of them, but they have to add up to one).
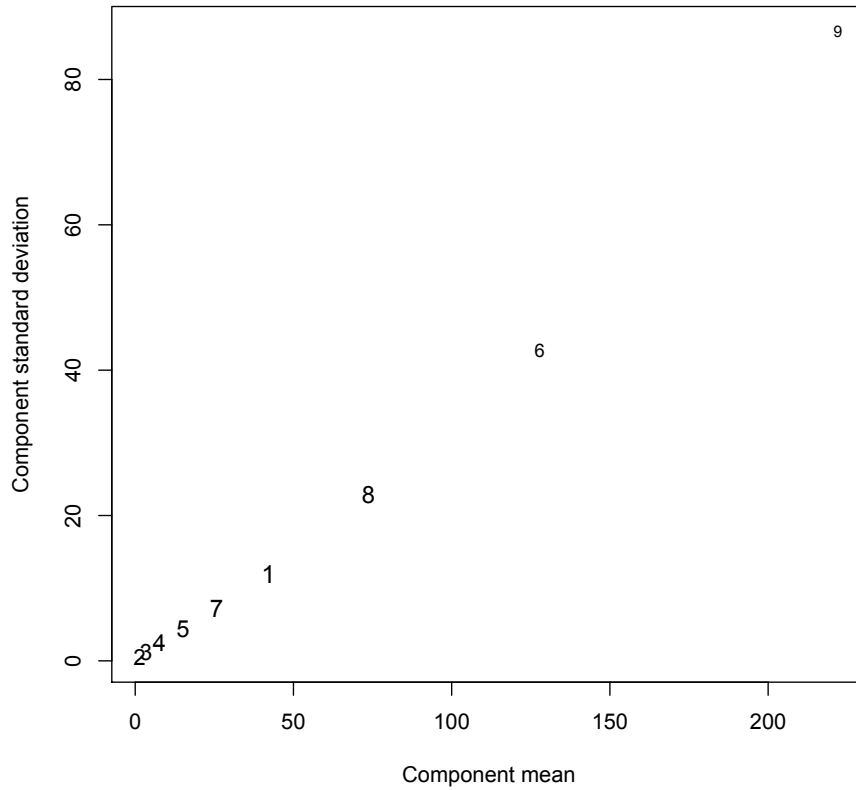
```
plot(0,xlim=range(snoq.k9$mu),ylim=range(snoq.k9$sigma),type="n",
     xlab="Component mean", ylab="Component standard deviation")
points(x=snoq.k9$mu,y=snoq.k9$sigma,pch=as.character(1:9),
       cex=sqrt(0.5+5*snoq.k9$lambda))
```

Figure 20.8: Characteristics of the components of the 9-mode Gaussian mixture. The horizontal axis gives the component mean, the vertical axis its standard deviation. The area of the number representing each component is proportional to the component's mixing weight.

Comparison of density estimates
Kernel vs. Gaussian mixture
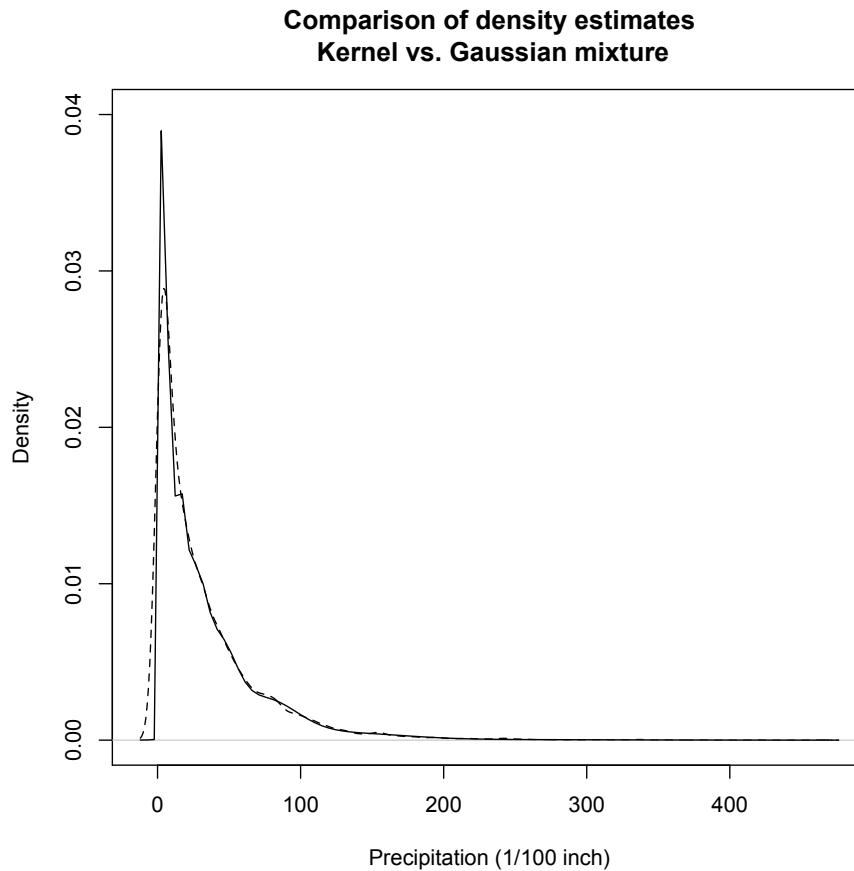

```
plot(density(snoq),lty=2,ylim=c(0,0.04),
    main=paste("Comparison of density estimates\n",
              "Kernel vs. Gaussian mixture"),
    xlab="Precipitation (1/100 inch)")
curve(dnormalmix(x,snoq.k9),add=TRUE)
```

Figure 20.9: Dashed line: kernel density estimate. Solid line: the nine-Gaussian mixture. Notice that the mixture, unlike the KDE, gives negligible probability to negative precipitation.

```
snoqualmie.classes <- snoqualmie
wet.days <- (snoqualmie > 0) & !(is.na(snoqualmie))
snoqualmie.classes[wet.days] <- day.classes
```

(Note that `wet.days` is a 36 × 366 logical array.) Now, it's somewhat inconvenient that the index numbers of the components do not perfectly correspond to the mean amount of precipitation — class 9 really is more similar to class 6 than to class 8. (See Figure 20.8.) Let's try replacing the numerical labels in `snoqualmie.classes` by those means.

```
snoqualmie.classes[wet.days] <- snoq.k9$mu[day.classes]
```

This leaves alone dry days (still zero) and NA days (still NA). Now we can plot (Figure 20.10).

The result is discouraging if we want to read any deeper meaning into the classes. The class with the heaviest amounts of precipitation is most common in the winter, but so is the classes with the second-heaviest amount of precipitation, the etc. It looks like the weather changes smoothly, rather than really having discrete classes. In this case, the mixture model seems to be merely a predictive device, and not a revelation of hidden structure.[11]

---

[11]A a distribution called a "type II generalized Pareto", where $p(x) \propto (1 + x/\sigma)^{-\theta-1}$, provides a decent fit here. (See Shalizi 2007; Arnold 1983 on this distribution and its estimation.) With only two parameters, rather than 26, its log-likelihood is only 1% higher than that of the nine-component mixture, and it is almost but not quite as calibrated. One origin of the type II Pareto is as a mixture of exponentials (Maguire *et al.*, 1952). If $X|Z \sim \text{Exp}(\sigma/Z)$, and Z itself has a Gamma distribution, $Z \sim \Gamma(\theta, 1)$, then the unconditional distribution of X is type II Pareto with scale σ and shape θ. We might therefore investigate fitting a finite mixture of exponentials, rather than of Gaussians, for the Snoqualmie Falls data. We might of course still end up concluding that there is a continuum of different sorts of days, rather than a finite set of discrete types.
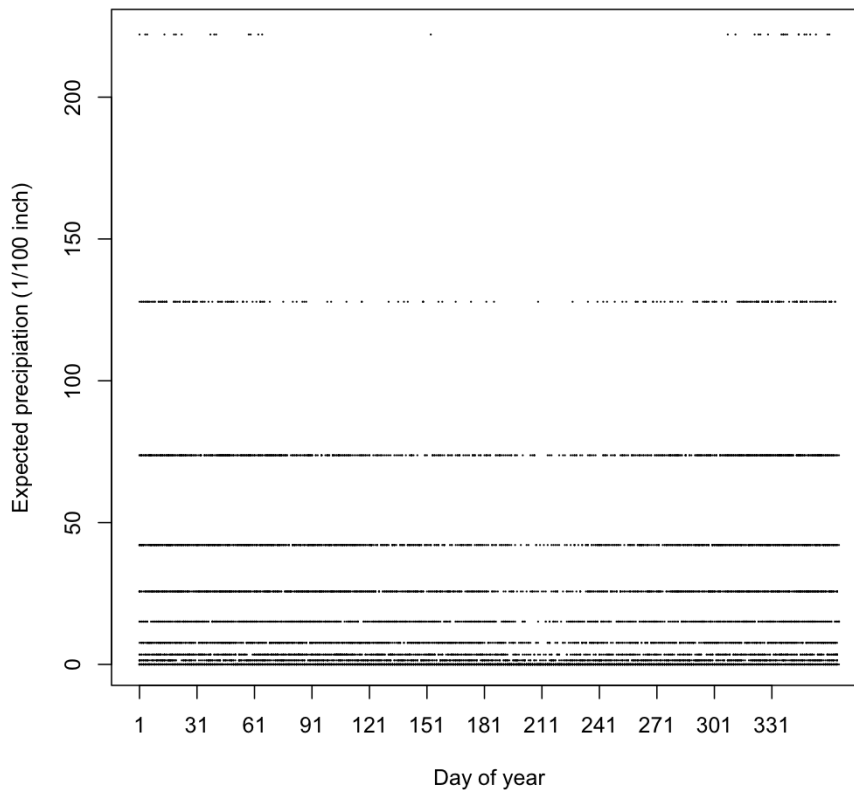
```
plot(0,xlim=c(1,366),ylim=range(snoq.k9$mu),type="n",xaxt="n",
    xlab="Day of year",ylab="Expected precipiation (1/100 inch)")
axis(1,at=1+(0:11)*30)
for (year in 1:nrow(snoqualmie.classes)) {
  points(1:366,snoqualmie.classes[year,],pch=16,cex=0.2)
}
```

Figure 20.10: Plot of days classified according to the nine-component mixture. Horizontal axis: day of the year, numbered from 1 to 366 (to handle leap-years). Vertical axis: expected amount of precipitation on that day, according to the most probable class for the day.

20.4.6   Hypothesis Testing for Mixture-Model Selection

An alternative to using cross-validation to select the number of mixtures is to use hypothesis testing. The k-component Gaussian mixture model is nested within the $(k+1)$-component model, so the latter must have a strictly higher likelihood on the training data. If the data really comes from a k-component mixture (the null hypothesis), then this extra increment of likelihood will follow one distribution, but if the data come from a larger model (the alternative), the distribution will be different, and stochastically larger.

Based on general likelihood theory, we might expect that the null distribution is, for large sample sizes,

$$2(\log L_{k+1} - \log L_k) \sim \chi^2_{dim(k+1)-dim(k)} \tag{20.29}$$

where L_k is the likelihood under the k-component mixture model, and $dim(k)$ is the number of parameters in that model. (See Appendix B.) There are however several reasons to distrust such an approximation, including the fact that we are approximating the likelihood through the EM algorithm. We can instead just find the null distribution by simulating from the smaller model, which is to say we can do a parametric bootstrap.

While it is not too hard to program this by hand (Exercise 4), the `mixtools` package contains a function to do this for us, called `boot.comp`, for "bootstrap comparison". Let's try it out[12].

```
# See footnote regarding this next command
source("http://www.stat.cmu.edu/~cshalizi/402/lectures/20-mixture-examples/
snoq.boot <- boot.comp(snoq,max.comp=10,mix.type="normalmix",
                      maxit=400,epsilon=1e-2)
```

This tells `boot.comp()` to consider mixtures of up to 10 components (just as we did with cross-validation), increasing the size of the mixture it uses when the difference between k and $k+1$ is significant. (The default is "significant at the 5% level", as assessed by 100 bootstrap replicates, but that's controllable.) The command also tells it what kind of mixture to use, and passes along control settings to the EM algorithm which does the fitting. Each individual fit is fairly time-consuming, and we are requiring 100 at each value of k. This took about five minutes to run on my laptop.

This selected three components (rather than nine), and accompanied this decision with a rather nice trio of histograms explaining why (Figure 20.11). Remember that `boot.comp()` stops expanding the model when there's even a 5% chance of that the apparent improvement could be due to mere over-fitting. This is actually pretty conservative, and so ends up with rather fewer components than cross-validation.

---

[12] As of this writing (5 April 2011), there is a subtle, only-sporadically-appearing bug in the version of this function which is part of the released package. The `bootcomp.R` file on the class website contains a fix, kindly provided by Dr. Derek Young, and should be sourced after loading the package, as in the code example following. Dr. Young informs me that the fix will be incorporated in the next release of the `mixtools` package, scheduled for later this month.
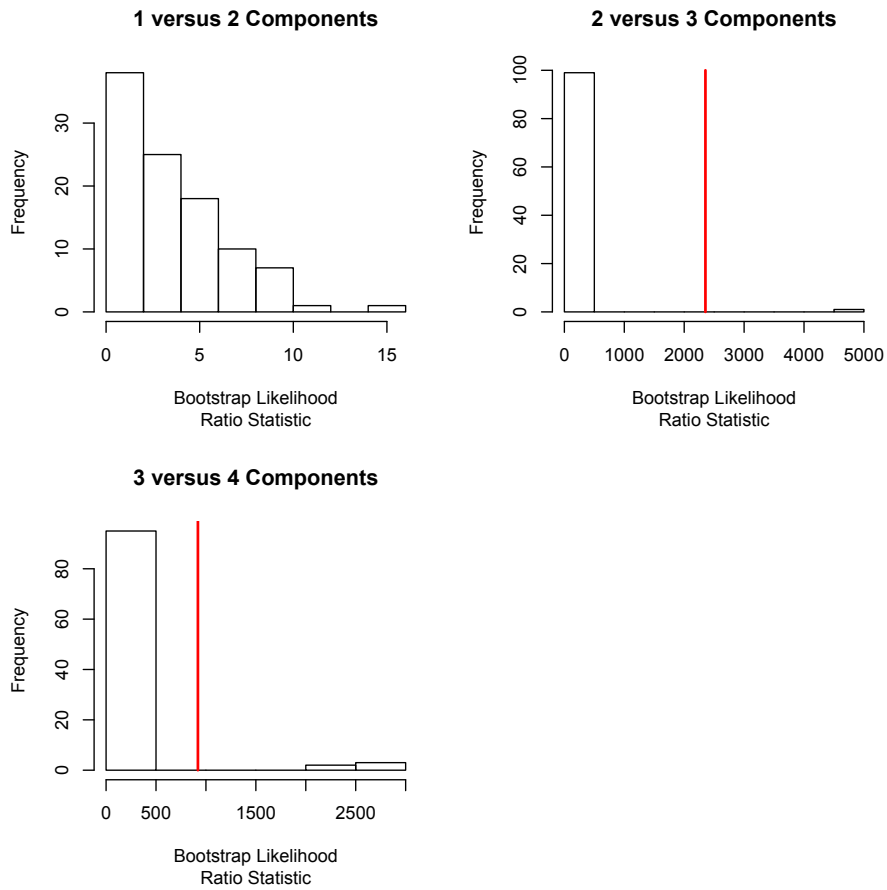
Figure 20.11: Histograms produced by `boot.comp()`. The vertical red lines mark the observed difference in log-likelihoods.

Let's explore the output of `boot.comp()`, conveniently stored in the object `snoq.boot`.

```
> str(snoq.boot)
List of 3
 $ p.values   : num [1:3] 0 0.01 0.05
 $ log.lik    :List of 3
  ..$ : num [1:100] 5.889 1.682 9.174 0.934 4.682 ...
  ..$ : num [1:100] 2.434 0.813 3.745 6.043 1.208 ...
  ..$ : num [1:100] 0.693 1.418 2.372 1.668 4.084 ...
 $ obs.log.lik: num [1:3] 5096 2354 920
```

This tells us that `snoq.boot` is a list with three elements, called `p.values`, `log.lik` and `obs.log.lik`, and tells us a bit about each of them. `p.values` contains the p-values for testing H_1 (one component) against H_2 (two components), testing H_2 against H_3, and H_3 against H_4. Since we set a threshold p-value of 0.05, it stopped at the last test, accepting H_3. (Under these circumstances, if the difference between $k = 3$ and $k = 4$ was really important to us, it would probably be wise to increase the number of bootstrap replicates, to get more accurate p-values.) `log.lik` is itself a list containing the bootstrapped log-likelihood ratios for the three hypothesis tests; `obs.log.lik` is the vector of corresponding observed values of the test statistic.

Looking back to Figure 20.5, there is indeed a dramatic improvement in the generalization ability of the model going from one component to two, and from two to three, and diminishing returns to complexity thereafter. Stopping at $k = 3$ produces pretty reasonable results, though repeating the exercise of Figure 20.10 is no more encouraging for the reality of the latent classes.

20.5 Exercises

To think through, not to hand in.

1. Write a function to simulate from a Gaussian mixture model. Check that it works by comparing a density estimated on its output to the theoretical density.

2. Work through the E- step and M- step for a mixture of two Poisson distributions.

3. Code up the EM algorithm for a mixture of K Gaussians. 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?

4. Write a function to find the distribution of the log-likelihood ratio for testing the hypothesis that the mixture has k Gaussian components against the alternative that it has $k + 1$, by simulating from the k-component model. Compare the output to the `boot.comp` function in `mixtools`.

5. Write a function to fit a mixture of exponential distributions using the EM algorithm. Does it do any better at discovering sensible structure in the Snoqualmie Falls data?

6. Explain how to use relative distribution plots to check calibration, along the lines of Figure 20.4.3.