Lecture 20, Mixture Examples and Complements

36-402, Advanced Data Analysis

5 April 2011

Contents

1 Snoqualmie Falls Revisited

1.1 Fitting a Mixture of Gaussians to Real Data

Let's go back to the Snoqualmie Falls data set, last used in lecture 16. 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 1 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?

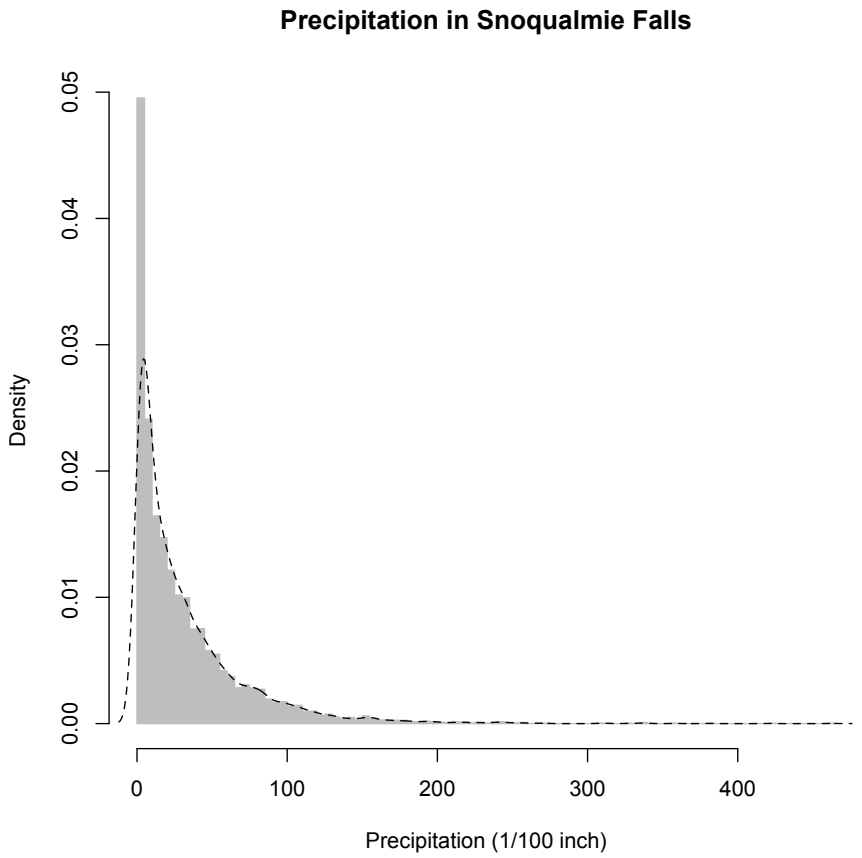
```
> snoq.k2 <- normalmixEM(snoq,k=2,maxit=100,epsilon=0.01)
number of iterations= 59
> summary(snoq.k2)
summary of normalmixEM object:
          comp 1    comp 2
lambda  0.557564  0.442436
mu     10.267390 60.012594
sigma   8.511383 44.998102
loglik at estimate:  -32681.21
```

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.

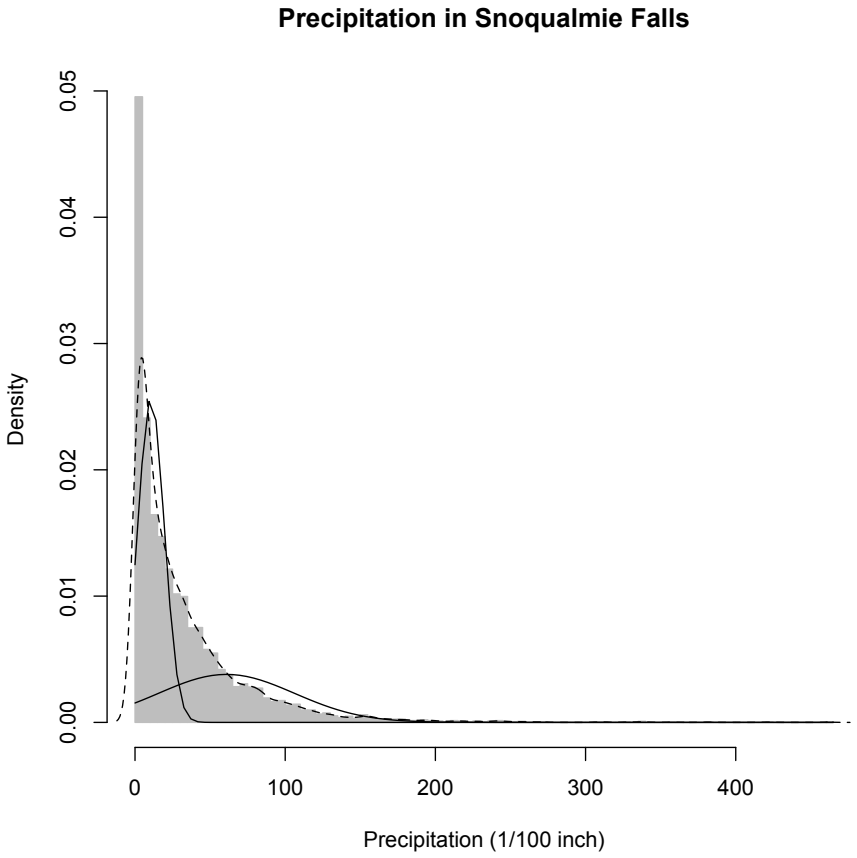
```
plot.normal.components <- function(mixture,component.number,...) {
  curve(mixture$lambda[component.number] *
        dnorm(x,mean=mixture$mu[component.number],
        sd=mixture$sigma[component.number]), add=TRUE, ...)
}
```

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.




```
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 1: 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.)




```
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 2: 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.

1.2 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.[1] The CDF of a two-component mixture is

$$F(x) = \lambda_1 F_1(x) + \lambda_2 F_2(x) \tag{1}$$

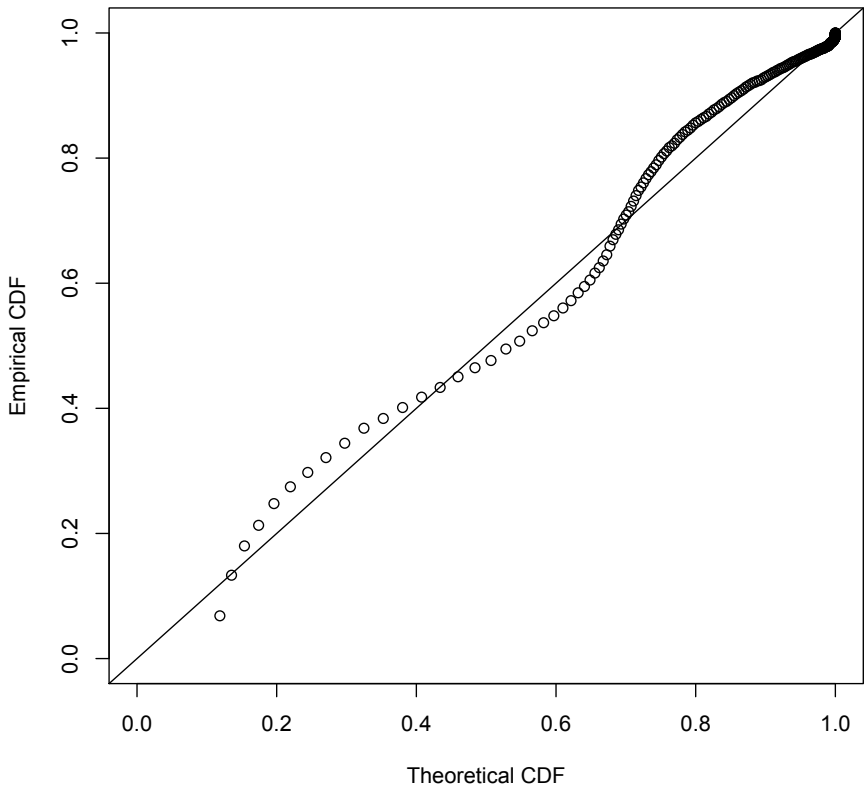
and similarly for more components. A little R experimentation gives a function for computing the CDF of a Gaussian mixture:

```
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 1.2. We do not have the tools to assess whether the *size* of the departure from the main diagonal is significant[2], but the fact that the errors are so very structured is rather suspicious.

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

[2] 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 3: 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 $\leq x$.

1.3 Selecting the Number of Components by Cross-Validation

Since a two-component mixture seems iffy, we could consider using more components. By going to three, four, etc. components, we improve our in-sample likelihood, 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 hypothesis 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 function 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)
}
```

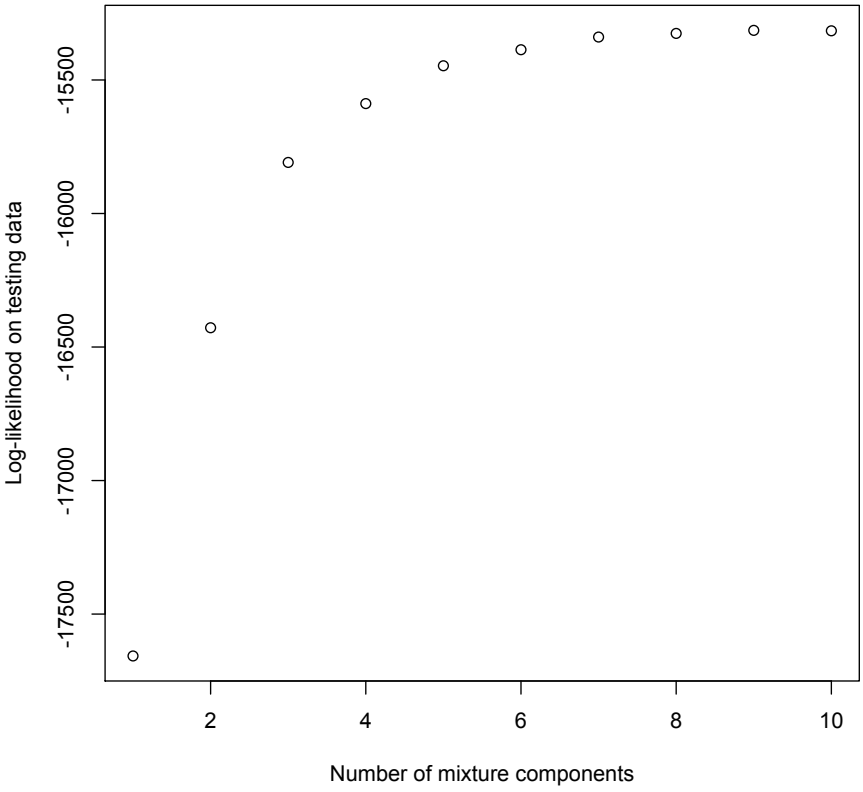
When you run this, you will probably see a lot of warning messages saying "One of the variances is going to zero; trying new starting values." The issue is that we can give any one value of x arbitrarily high likelihood by centering a Gaussian there and letting its variance shrink towards zero. This is however generally considered unhelpful — it leads towards the pathologies that keep us from doing pure maximum likelihood estimation in non-parametric problems (lecture 6) — 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[3]:

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

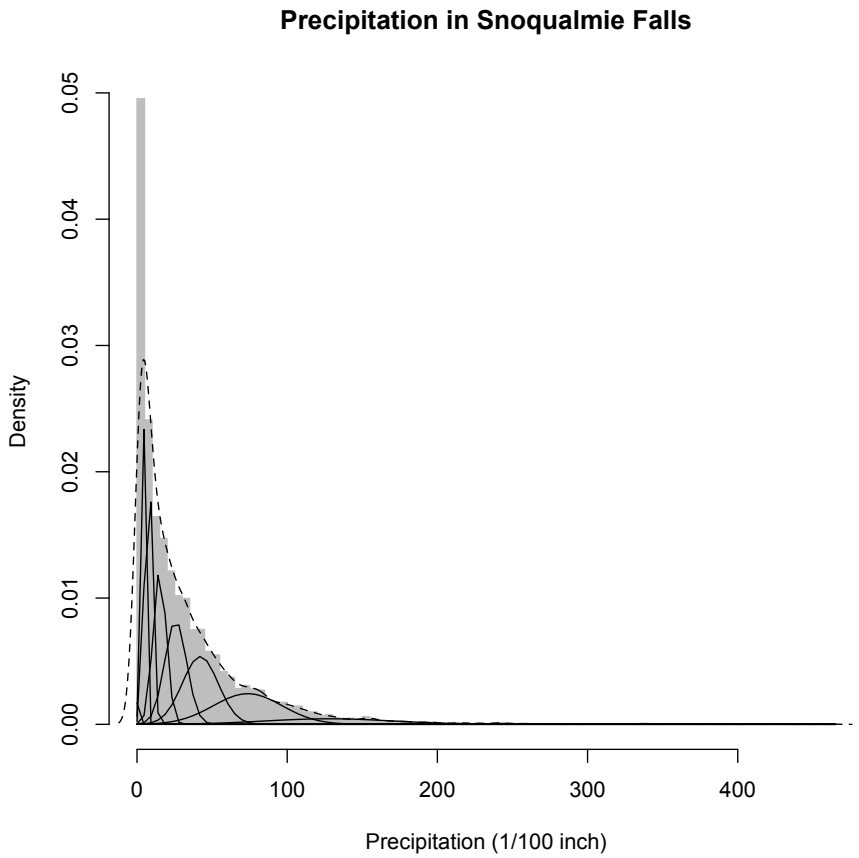
(See also Figure 4). This favors nine components to the mixture. It looks like Figure 5. The calibration is now nearly perfect, at least on the training data (Figure 1.3).

[3]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.



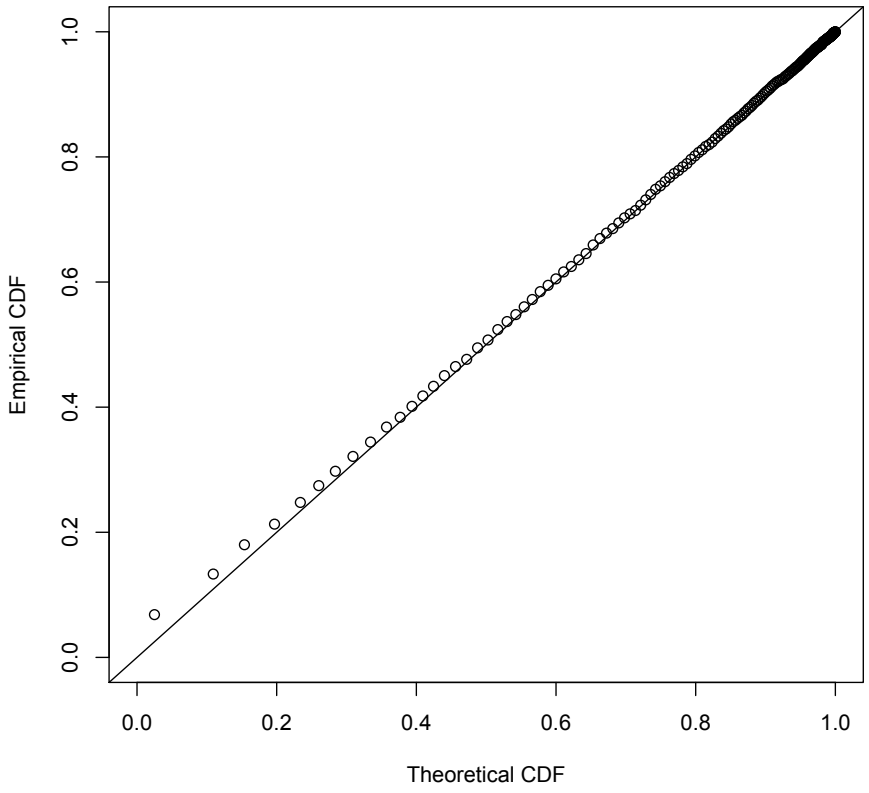

```
plot(x=1:10, y=loglikes,xlab="Number of mixture components",
     ylab="Log-likelihood on testing data")
```

Figure 4: 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.




```
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 5: As in Figure 2, 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 6: Calibration plot for the nine-component Gaussian mixture.

1.4 Interpreting the Mixture Components, or Not

The components of the mixture are far from arbitrary. It appears from Figure 5 that as the mean increases, so does the variance. This impression is confirmed from Figure 7. 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 8). It requires 26 parameters[4], 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.

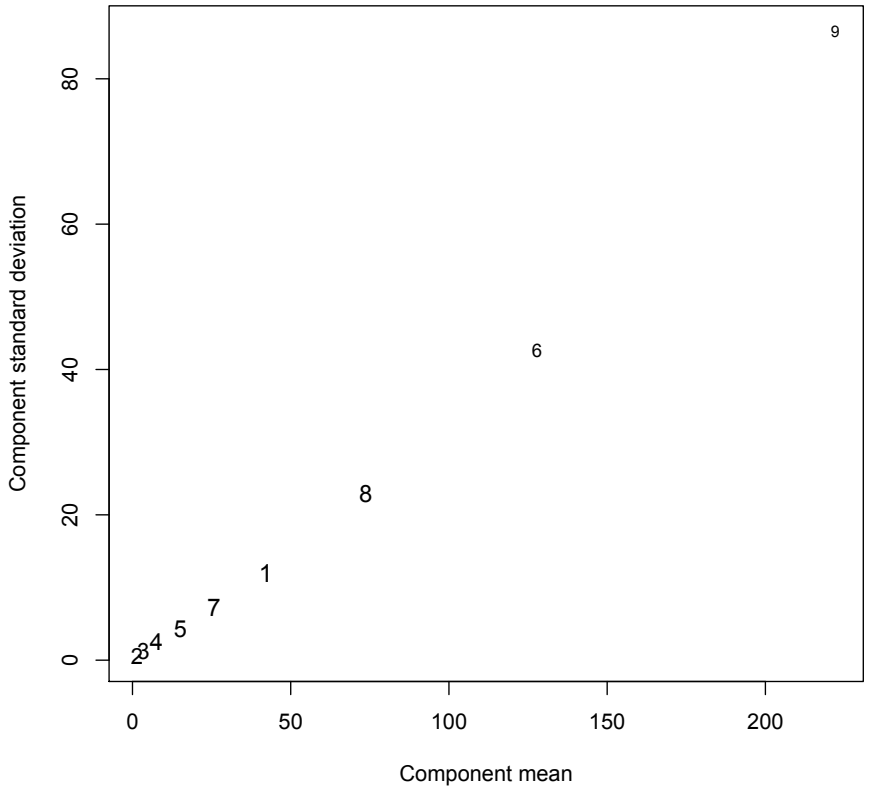
On the substantive side, there are various things we could do to check the idea that wet days really do divide into nine types. These are going to be informed by our background knowledge about the weather. One of the things we know, for example, is that weather patterns more or less repeat in an annual 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)
```

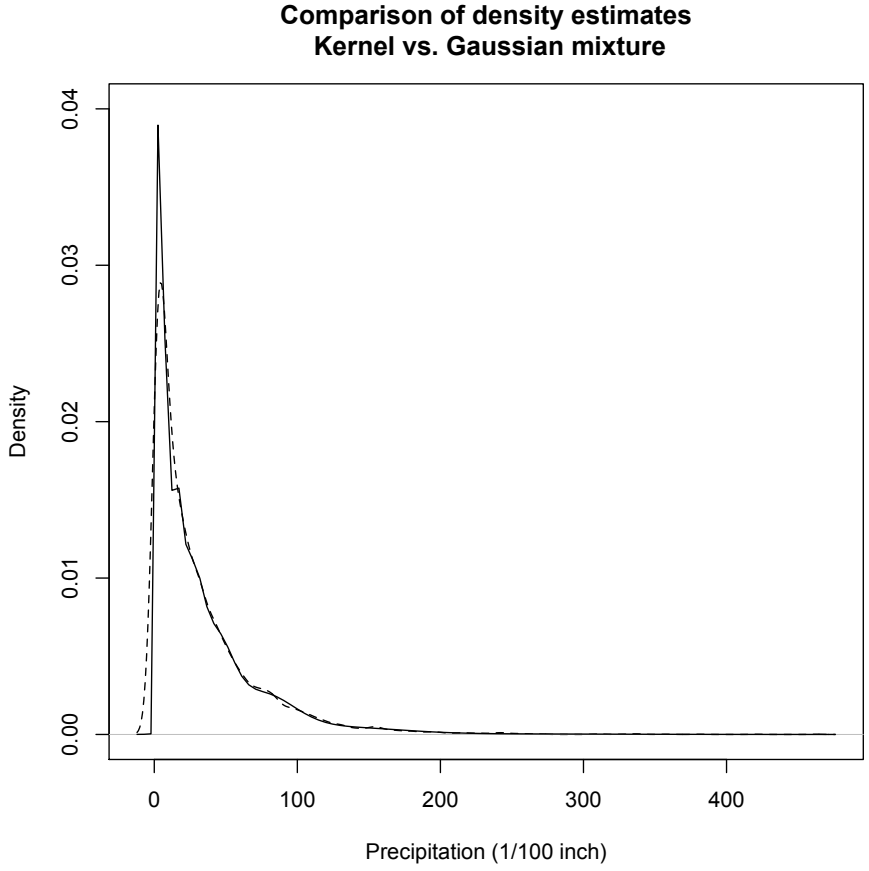
We can't just plot this and hope to see any useful patterns, because we want to see stuff recurring every year, and we've stripped out the dry days, the division

[4] 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 7: 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.




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

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.

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

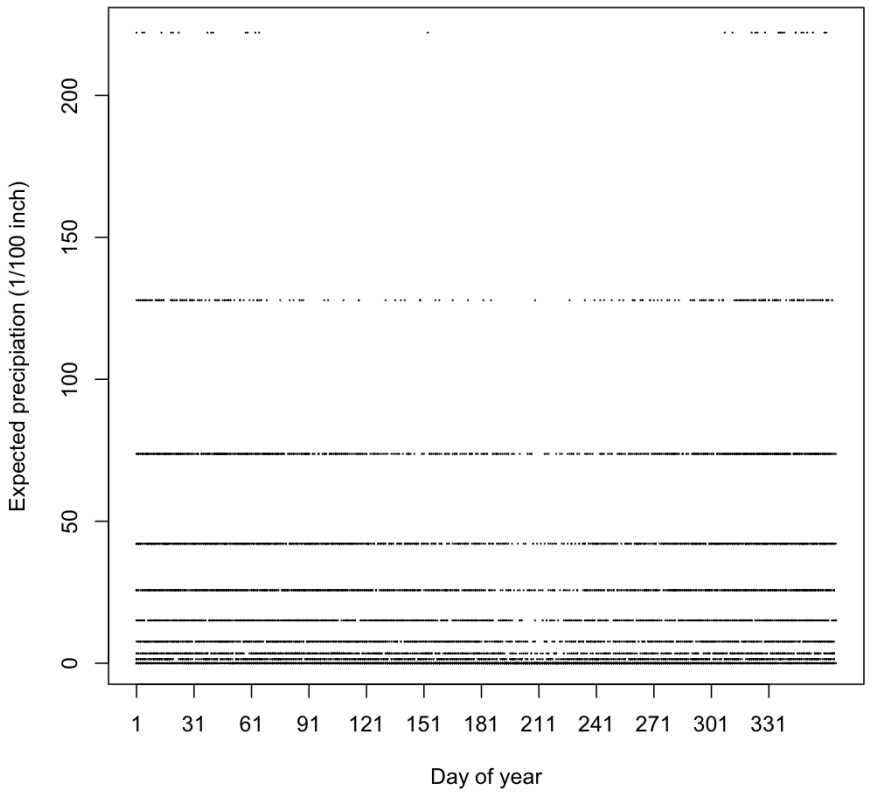
(Note that `wet.days` is a 36×366 logical array.) Now, it's somewhat inconvenient that the index numbers of the components do not perfectly correspond to the mean amount of precipitation — class 9 really is more similar to class 6 than to class 8. (See Figure 7.) Let's try replacing the numerical labels 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 9).

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.[5]

[5] 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 9: 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.

1.5 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,

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

where L_k is the likelihood under the k-component mixture model, and $dim(k)$ is the number of parameters in that model. (See the appendix to Lecture 2.) 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 however simply 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[6], the `mixtools` package contains a function to do this for us, called `boot.comp`, for "bootstrap comparison". Let's try it out[7].

```
# See footnote regarding this next command
source("http://www.stat.cmu.edu/~cshalizi/402/lectures/20-mixture-examples/bootcomp.R")
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 10). 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.

[6] Exercise: Try it!

[7] 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.

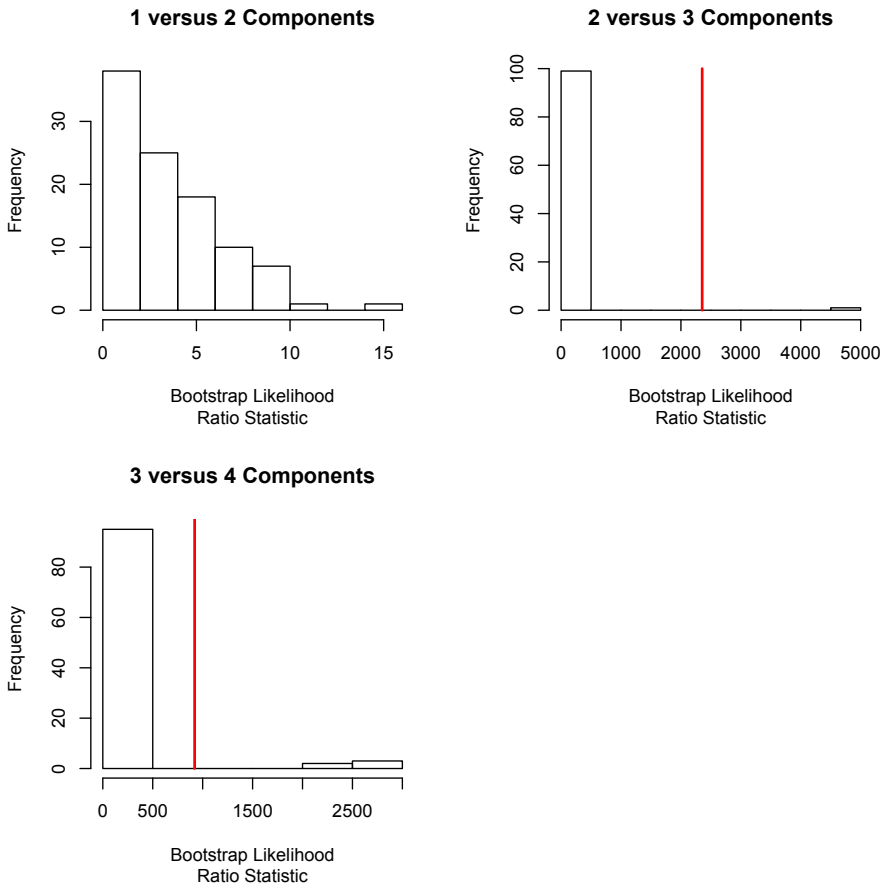


Figure 10: Histograms produced by `boot.comp()`. The vertical red lines mark the observed difference in log-likelihoods.

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 4, 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 9 is no more encouraging for the reality of the latent classes.

2 Multivariate Gaussians

Most of this section repeats the appendix to Lecture 4.

The multivariate Gaussian is just the generalization of the ordinary Gaussian to vectors. Scalar Gaussians are parameterized by a mean μ and a variance σ^2, which we symbolize by writing $X \sim \mathcal{N}(\mu, \sigma^2)$. Multivariate Gaussians, likewise, are parameterized by a mean vector $\vec{\mu}$, and a variance-covariance matrix $\mathbf{\Sigma}$, written $\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \mathbf{\Sigma})$. The components of $\vec{\mu}$ are the means of the different components of $\vec{X}$. The i, j^{th} component of $\mathbf{\Sigma}$ is the covariance between X^i and X^j (so the diagonal of $\mathbf{\Sigma}$ gives the component variances).

Just as the probability density of scalar Gaussian is

$$p(x) = \left(2\pi\sigma^2\right)^{-1/2} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{3}$$

the probability density of the multivariate Gaussian is

$$p(\vec{x}) = (2\pi \det \mathbf{\Sigma})^{-d/2} \exp\left\{-\frac{1}{2}(\vec{x}-\vec{\mu}) \cdot \mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right\} \tag{4}$$

Finally, remember that the parameters of a Gaussian change along with linear transformations

$$X \sim \mathcal{N}(\mu, \sigma^2) \Leftrightarrow aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2) \tag{5}$$

and we can use this to "standardize" any Gaussian to having mean 0 and variance 1 (by looking at $\frac{X-\mu}{\sigma}$). Likewise, if

$$\vec{X} \sim \mathcal{MVN}(\vec{\mu}, \mathbf{\Sigma}) \tag{6}$$

then

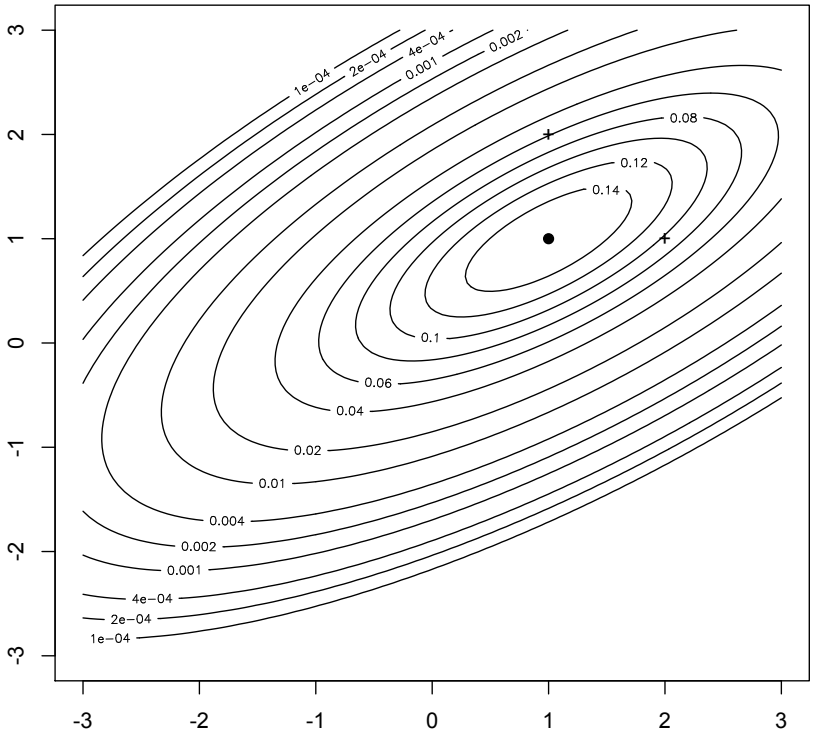
$$\mathbf{a}\vec{X} + \vec{b} \sim \mathcal{MVN}(\mathbf{a}\vec{\mu} + \vec{b}, \mathbf{a\Sigma a}^T) \tag{7}$$

In fact, the analogy between the ordinary and the multivariate Gaussian is so complete that it is very common to not really distinguish the two, and write $\mathcal{N}$ for both.

The multivariate Gaussian density is most easily visualized when $d = 2$, as in Figure 11. The probability contours are ellipses. The density changes comparatively slowly along the major axis, and quickly along the minor axis. The two points marked + in the figure have equal geometric distance from $\vec{\mu}$, but the one to its right lies on a higher probability contour than the one above it, because of the directions of their displacements from the mean.

In fact, we can use some facts from linear algebra to understand the general pattern here, for arbitrary multivariate Gaussians in an arbitrary number of dimensions. The covariance matrix $\mathbf{\Sigma}$ is symmetric and positive-definite, so we know from matrix algebra that it can be written in terms of its eigenvalues and eigenvectors:

$$\mathbf{\Sigma} = \mathbf{v}^T\mathbf{dv} \tag{8}$$




```
library(mvtnorm)
x.points <- seq(-3,3,length.out=100)
y.points <- x.points
z <- matrix(0,nrow=100,ncol=100)
mu <- c(1,1)
sigma <- matrix(c(2,1,1,1),nrow=2)
for (i in 1:100) {
  for (j in 1:100) {
     z[i,j] <- dmvnorm(c(x.points[i],y.points[j]),mean=mu,sigma=sigma)
  }
}
contour(x.points,y.points,z)
```

Figure 11: Probability density contours for a two-dimensional multivariate Gaussian, with mean $\vec{\mu} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ (solid dot), and variance matrix $\mathbf{\Sigma} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$. Using `expand.grid`, as in Lecture 6, would be more elegant coding than this double `for` loop.

where $\mathbf{d}$ is the diagonal matrix of the eigenvalues of $\mathbf{\Sigma}$, and $\mathbf{v}$ is the matrix whose columns are the eigenvectors of $\mathbf{\Sigma}$. (Conventionally, we put the eigenvalues in $\mathbf{d}$ in order of decreasing size, and the eigenvectors in $\mathbf{v}$ likewise, but it doesn't matter so long as we're consistent about the ordering.) Because the eigenvectors are all of length 1, and they are all perpendicular to each other, it is easy to check that $\mathbf{v}^T\mathbf{v} = \mathbf{I}$, so $\mathbf{v}^{-1} = \mathbf{v}$ and $\mathbf{v}$ is an orthogonal matrix. What actually shows up in the equation for the multivariate Gaussian density is Σ^{-1}, which is

$$(\mathbf{v}^T\mathbf{d}\mathbf{v})^{-1} = \mathbf{v}^{-1}\mathbf{d}^{-1}\left(\mathbf{v}^T\right)^{-1} = \mathbf{v}^T\mathbf{d}^{-1}\mathbf{v} \tag{9}$$

Geometrically, orthogonal matrices represent rotations. Multiplying by $\mathbf{v}$ rotates the coordinate axes so that they are parallel to the eigenvectors of $\mathbf{\Sigma}$. Probabilistically, this tells us that the axes of the probability-contour ellipse are parallel to those eigenvectors. The radii of the probability-contour ellipses are proportional to the square roots of the eigenvalues. To see *that*, look carefully at the math. Fix a level for the probability density whose contour we want, say f_0. Then we have

$$\begin{aligned}
f_0 &= (2\pi \det \mathbf{\Sigma})^{-d/2} \exp\left\{-\frac{1}{2}(\vec{x}-\vec{\mu})\cdot\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right\} && (10)\\
c &= (\vec{x}-\vec{\mu})\cdot\mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu}) && (11)\\
&= (\vec{x}-\vec{\mu})^T\mathbf{v}^T\mathbf{d}^{-1}\mathbf{v}(\vec{x}-\vec{\mu}) && (12)\\
&= (\vec{x}-\vec{\mu})^T\mathbf{v}^T\mathbf{d}^{-1/2}\mathbf{d}^{-1/2}\mathbf{v}(\vec{x}-\vec{\mu}) && (13)\\
&= \left(\mathbf{d}^{-1/2}\mathbf{v}(\vec{x}-\vec{\mu})\right)^T\left(\mathbf{d}^{-1/2}\mathbf{v}(\vec{x}-\vec{\mu})\right) && (14)\\
&= \left\|\mathbf{d}^{-1/2}\mathbf{v}(\vec{x}-\vec{\mu})\right\|^2 && (15)
\end{aligned}$$

where c combines f_0 and all the other constant factors, and $\mathbf{d}^{-1/2}$ is the diagonal matrix whose entries are one over the square roots of the eigenvalues of $\mathbf{\Sigma}$. The $\mathbf{v}(\vec{x}-\vec{\mu})$ term takes the displacement of $\vec{x}$ from the mean, $\vec{\mu}$, and replaces the components of that vector with its projection on to the eigenvectors. Multiplying by $\mathbf{d}^{-1/2}$ then scales those projections, and so the radii have to be proportional to the square roots of the eigenvalues.[8]

In terms of inference, the multivariate Gaussian again basically works the same way as the univariate Gaussian. If we use maximum likelihood, the estimated mean vector is just the sample mean vector, and the estimated covariance matrix is just the sample covariance matrix.

Computationally, it is not hard to write functions to calculate the multivariate Gaussian density, or to generate multivariate Gaussian random vectors. Unfortunately, no one seems to have thought to put a standard set of such functions in the basic set of R packages, so you have to use a different library.

[8] If you find all this manipulation of eigenvectors and eigenvalues of the covariance matrix very reminiscent of principal components analysis, you're right; this was one of the ways in which PCA was originally discovered. But PCA does not require any distributional assumptions.

`mvtnorm` contains functions for calculating the density, cumulative distribution and quantiles of the multivariate Gaussian, and for generating random vectors[9] The package `mixtools`, which we are using for mixture models, includes functions for the multivariate Gaussian density and for random-vector generation.

3 Exercises

Not to be handed in.

1. Write a function to calculate the density of a multivariate Gaussian with a given mean vector and covariance matrix. Check it against an existing function from one of the packages mentioned above.
2. Write a function to generate multivariate Gaussian random vectors, using `rnorm`.
3. Write a function to simulate from a Gaussian mixture model.
4. 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?

References

Arnold, Barry C. (1983). *Pareto Distributions*. Fairland, Maryland: International Cooperative Publishing House.

Maguire, B. A., E. S. Pearson and A. H. A. Wynn (1952). "The time intervals between industrial accidents." *Biometrika*, **39**: 168–180. URL `http://www.jstor.org/pss/2332475`.

Shalizi, Cosma Rohilla (2007). "Maximum Likelihood Estimation and Model Testing for q-Exponential Distributions." *Physical Review E*, **submitted**. URL `http://arxiv.org/abs/math.ST/0701854`.

[9] It also has such functions for multivariate t distributions, which are to multivariate Gaussians exactly as ordinary t distributions are to univariate Gaussians.