Mixture and Hidden Markov Models: a Unified Introduction

Norman Matloff
Department of Computer Science
University of California, Davis

December 7, 2022

The notion of *mixture models* (MMs) is a classical probabilistic concept, arising frequently in applications. The field of *Hidden Markov Models* (HMMs) is also a quite well established probabilistic model, but has received much more attention with the rise of interest in machine learning. HMMs too have very interesting applications, such as in bioinformatics and language processing.

As there is a natural connection of mixture models (MMs) to HMMs, we present both here. We also present examples of using R packages to apply these models to data.

We will cover enough mathematical detail to specify the models and indicate the statistical issues, but subject to the goal of keeping things simple.

1 Overview

Here we will present some motivating examples, and then give a high-level view of the structures and issues in MMs and HMMs. In both models, we have an observed variable Y, and a hidden or *latent* state S.

1.1 Motivating Examples

1.1.1 Example: Box of Batteries

Say we have a large box of batteries. They are known to be of two different types, but the two types are visually indistinguishable. From past experience, suppose we know that a good model for lifetimes of batteries is exponential. We have three unknown parameters: q, the proportion of batteries of the first type; μ_1, the mean lifetime of the first type, and μ_2, the mean for the second type.

We take a random sample of n batteries and measure the lifetimes of each, resulting in our data $Y_1, ..., Y_n$, independent and identically distributed (iid) random variables. Unseen are the types of these batteries, $S_1, ..., S_n$, the *hidden state* of each battery.

Our objective is the estimate q and the means $\mu_i = 1/\lambda_i$, based on the Y_j.

1.1.2 Example: Network Noise

Suppose we have a network line that is known to occasionally be noisy, and that during noisy periods bits will be corrupted in such a way that the probability of a 0, which is 0.5 in the original transmitted data, is 0.20 during noisy periods. Suppose that on average 10% of the bits arrive during noise periods.

We focus on the situation in which the status of the line, working vs. noisy, is unknown. It thus is a *hidden state*. We have data Y_j on the received bits, and want to estimate O_j, the originally sent bits.

1.1.3 Example: Old Faithful Geyser

The data here consist of durations of eruptions of the famous Old Faithful geyser in the US' Yellowstone National Park. The dataset is **faithful**, a built-in dataset in R.

A histogram, obtained via

```
> hist(faithful$eruptions,breaks=18,freq=FALSE)
```

and shown in Figure 1, seems to suggest that the eruption duration distribution is a mixture of two normally distributed random variables. This seems even more plausible if we use R's **density()** function, as in Figure 2.[1]

This has led to many physical theories over the years. Rather elaborate physical models have been developed, such as that in O'Hara and Esawi, "Model for the eruption of the Old Faithful geyser, Yellowstone National Park," *GSA Today*, June 2013. This paper is full of physical detail ("... the dynamics of vapor bubble formation (and collapse) during boiling in the conduit..."), but in simple terms, it posits two processes, which gave rise to long and short durations before an eruption, consistent with the bimodal density form suggested by the above graphs.

[1] A histogram is a probability density estimate (as long as one keeps the total area under the curve to be 1.0, as we have done here by setting **freq** to FALSEi). More advanced density estimators, such as produced by R's **density()** function, produce smoother and potentially more accurate plots. Such methods have parameters analogous to the number of breaks/bins in a histogram; for R's **density()** function, the argument is the bandwidth **bw**, which we have taken to be the default here.

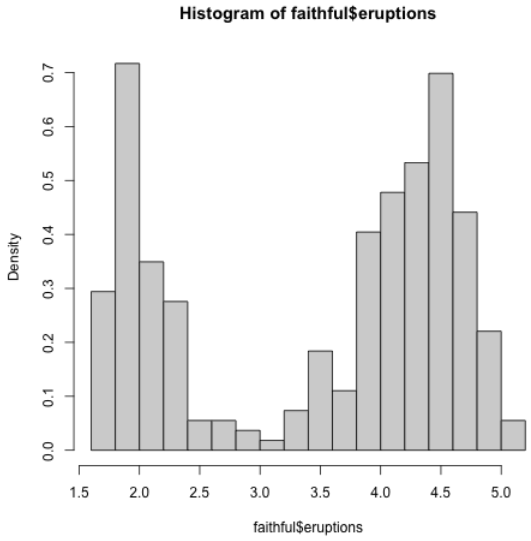

Figure 1: Old Faithful eruption durations

Assuming there really are two types of eruptions, our hidden state S_j for the j^{th} eruption in our dataset is the type of eruption. Y_j is the duration of that eruption. Again, the S_j are unobserved.

Our objective is to use the Y_j data to estimate q, the proportion of type 1 eruptions, and μ_1, μ_2, σ_1 and σ_2, the means and standard deviations of the assumed normal distributions for duration in the two eruption types.

1.1.4 Example: Financial Time Series

The **sp500** dataset, included with some software we will use below, consists of 772 days of the Standard and Poor US stock market average. In the book associated with the software,[2] the authors postulate 2 hidden states, "bull" and "bear" sentiments among the traders, and fit an HMM accordingly.

1.2 Number of States

In many applications, a major part of the modeling process is deciding on the number of states. In our battery, network and geyser examples above, it is natural to take this number to be 2, but

[2] *Mixture and Hidden Markov Models with R,* by Ingmar Visser and Maarten Speekenbrink, Springer.

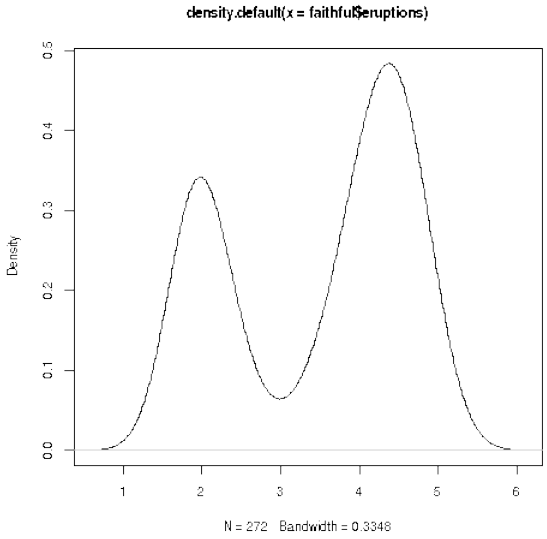

Figure 2: Old Faithful eruption times, smoothed

generally there is no obvious such number. In the financial data example, for instance, why have just 2 levels of investor sentiment? Whyt not 3, or 8, and so on?

In such cases, we have a classic model-fitting choice, a famous principle in model fitting. Here is the issue (iilustrated in a non-MM/HMM context):

> The *Bias-Variance Tradeoff*.
>
> Say we wish to predict human weight from height. We wish to estimate the function $w(t) = E(W|H = t)$, the relation between weight and height in our population. We might try a linear model for $w(t)$, but a quadratic model would give use greater flexibility. A cubic model be even more general, and so on.
>
> From the notion of a Taylor series in calculus[3], one might think that the higher the degree of the fitted polynomial, the better. But that is merely saying that higher-degree models have smaller model bias, and counteracting that is the problem of sampling error. The higher the degree, the more the sampling error (called the *standard error* in statistics). So, we need larger datasets for higher-degree models.
>
> A graph of prediction accuracy versus model complexity, say polynomial degree, is typically U-shaped. If we are on the "wrong" side of the tradeoff, i.e. to the right of the

[3]Or for those with a background in real analysis, the Stone-Weierstrass Theorem.

bottom of the U, the model is said to be *overfit*.[4]

So, the more hidden states in our model, the smaller the model bias but the larger the sampling error; setting the number of hidden states at too large a level will result in overfitting.

For instance, consider financial time series data as in Section 1.1.4. The authors postulate 2 hidden states, "bull" and "bear" sentiments among the traders, and fit an HMM accordingly. With 3 states, 4 states and so on, we might achieve more accuracy for the first few models, but eventually the Bias-Variance Tradeoff will result in overfitting.

And though the choice of 2 states in that example does not seem large relative to 772, we will see below that we are maximizing a certain quantity over an enormous number of choices, again raising the possibility of overfitting.

1.3 Time (and "Time")

In spite of sharing the property of having hidden states, and the $Y|S$ structure, there is a fundamental difference between the battery example above and the others. In the battery case, the pairs (S_j, Y_j) are iid, while in the network, geyser and financial examples they are sequential in time; there is a time dependency between each pair and the next.

In some cases "time" may correspond to position. say in HMM genomics models, but again, the sequential nature is key. That is what distinguishes HMMs from MMs.

1.4 Conditional Probabilities of Observed Values

Let S denote the hidden state, and let Y denote the corresponding observed value. A major ingredient in the analysis will be expressions of the form

$$P(Y = w|S = v) \tag{1}$$

If Y is continuous rather than discrete, the above expression would be something like

$$f_{Y|S}(w, v) \tag{2}$$

[4]See *https://github.com/matloff/qeML/blob/master/inst/Overfitting.md* for further discussion in the general context.

where $f_{Y|S}$ is the conditional density of Y given S:

$$f_{Y|S}(w,v) = \frac{d}{dw} F_{Y|S}(w,v) = \frac{d}{dw} P(Y \leq w \mid S = v) \tag{3}$$

These conditional distribution quantities are then used to estimate model parameters, as we will see below.

1.5 We Work Only with Sample Data

The word *estimate* is vital above. As with any statistical method, our results are just estimates of population values, and the larger our sample, the more likely our estimate is close to the true value. This also means that the larger our sample, the further to the right at which the bottom of the U curve occurs; in other words, with larger samples, we can afford to fit more complex models.

1.6 Mean and Variance of Random Variables in Latent-State Models

A useful expression for EY follows from the Law of Total Expectation:

$$EY = E[E(Y|S)] \tag{4}$$

Also, we have the Law of Total Variance,

$$Var(Y) = E[Var(Y|S)] + Var[E(Y|S)] \tag{5}$$

2 Mixture Models

Actually, MMs are typically not presented in the conditional distribution form we saw above. Let's see how to reconcile the standard description with what we saw above.

2.1 Definition

Say Y is discrete. Then

$$P(Y = w) = \sum_v P(S = v)\ P(Y = w|S = v) \tag{6}$$

So in terms of cumulative distribution functions (cdfs),[5]

$$F_Y(w) = P(Y \leq w) = \sum_v P(Y \leq w | S = v)\ P(S = v) = \sum_v q_v F_{Y|S}(w, v) \tag{7}$$

where

$$q_v = P(S = v) \tag{8}$$

Speaking just in terms of cdfs, we say that F_Y is a *mixture* of the cdfs $F_{Y|S}$, which simply means that F_Y is a linear combination of the $F_{Y|S}$, where the coefficients are nonnegative numbers whose sum is 1.[6]

We may have mixtures of more than two distributions. Consider random variables $X_1, ..., X_k$, with cdfs F_{X_i} (not necessarily independent),[7] and let $d_i,\ i = 1, ..., k$ be nonnegative numbers whose sum is 1. Define the random variable W to take on the value X_i with probability q_i, $i = 1, ..., k$. Then we say that W is a mixture of the X_i.

Note that

$$F_W(t) = \sum_{i=1}^{k} q_i F_{X_i}(t) \tag{9}$$

Similar relations hold for probability mass functions (X_i discrete) and density functions (X_i continuous).

In MM analysis, the usual quantities of interest are the F_{X_i} and the q_i. In the Old Faithful example, this means using the data to somehow estimate the means and standard deviations of the two normal distributions, and the proportions of eruptions of the two types.

Denote the mean and variance of X_i here by μ_i and σ_i^2 (whether or not the X_i have a normal distribution). Then (4) becomes

$$EY = \sum_{i=1}^{k} q_i \mu_i \tag{10}$$

[5]Standard notation for the cdf of a random variable X is F_X. For the conditional cdf of X given Z, we write $F_{X,Z}$.

[6]In math parlance, we say that F_Y is a *convex* combination of the $F_{(Y|S)}$..

[7]I use "X" for my variable name rather than "Y," to emphasize the mainly implicit nature of states.

What about (5)? In the present context,

$$E(Y|S) = \mu_S \tag{11}$$

Note that this is a random variable! It takes on the values $\mu_1, ..., \mu_k$ with probabilities $1_1, ..., q_k$, so its variance is

$$\sum_{i=1}^{k} q_i(\mu_i - \hat{\mu})^2 \tag{12}$$

We then have

$$Var(Y) = \sum_{i=1}^{k} q_i\sigma_i^2 + \sum_{i=1}^{k} q_i(\mu_i - \hat{\mu})^2 \tag{13}$$

where

$$\hat{\mu} = EY \tag{14}$$

2.2 The EM Algorithm

Probably the most common approach to estimating such quantities is the *EM algorithm*. The details can become complex, but let's at least look at an overview here.

Say the distribution of some probabilistic quantity depends on a vector of parameters ω. In many cases, finding a way to estimate ω is very difficult, but the estimation become mathematically tractable if we split ω into two subsets, θ and γ.

In our case here, we choose θ to be the parameters that describe distribution of $Y|S$, and choose γ to be the parameters describing the distribution of S, that is, consisting of the mixing proportions q_i. (We only need $q_i, q_2, ..., q_{k-1}$, since the proportions must sum to 1.)

In the battery example, θ is the vector (λ_1, λ_2), and γ is the proportion q of type 1 batteries. In the geyser example, θ is consists of the two means μ_i and two standard deviations σ_i of the two normal distributions, and γ is the proportion q of the the type 1 eruptions.

The algorithm works like this: We set initial guesses, θ_0 and γ_0 for the two parameter sets, then update alternately, first finding a new guess for θ based on our latest guess for γ, then vice versa. Of course, we also make use of our dataset at every step. So, the core of the algorithm is to iterate the following for $i = 1, 2, 3, ...$ until convergence:

1. Form a new guess for θ, denoted θ_{i+1}, based on γ_i.

2. Form a new guess for γ, denoted γ_{i+1}, based on θ_{i+1}.

How does this work in the battery example?

Step 1. The 'M' in "EM" stands for "maximization," alluding to the famous statistical estimation tool, Maximum Likelihood Estimation (MLE), familiar to many readers of this tutorial. The intuitive view is that we find the value of θ that "would have made our data most likely to occur," i.e. the value that maximizes

$$\Pi_{i=1}^{n} f_Y(Y_i;\theta) \tag{15}$$

(One typically maximizes the log of the above quantity, as sums are easier to deal with than products.)

Note that the likelihood is calculated using the *marginal* (i.e. unconditional) density of Y, which is

$$f_Y(t) = q\lambda_1 e^{-\lambda_1 t} + (1-q)\lambda_2 e^{-\lambda_2 t} \tag{16}$$

Remember, in every Step 1, q is considered known. We maximize with respect to the λ_j, not with respect to both the λ_j and q.

Step 2. Since $k = 2$, (10) becomes

$$EY = q/\lambda_1 + (1-q)/\lambda_2 \tag{17}$$

Remember, in every Step 2, the θ_j are considered known, in this case the λ_i. And, we can estimate EY in the left side (17) by the mean Y value in our dataset. So, we then simply solve for q to obtain the latest iterate for q.[8]

The geyser example is similar, except that f_Y is assumed to be a mixture of normals:

$$f_Y(t) = q\left[\frac{1}{\sqrt{2\pi}\sigma_1}\exp\left(\frac{t-\mu_1}{\sigma_1}\right)^2\right] + (1-q)\left[\frac{1}{\sqrt{2\pi}\sigma_2}\exp\left(\frac{t-\mu_2}{\sigma_2}\right)^2\right] \tag{18}$$

[8]For readers who know the Method of Moments estimation tool, the EM algorithm can be modified so that we use that tool in the Step 2s, the 'E' ("Expectation") steps. For cases with general k, which have more than one q_i, we need to estimate $k-1$ moments.

2.3 The mixtools Package

This is a large package with many functions for analysis of MMs. The EM algorithm is used extensively. Here we will illustate the function **normalmixEM()**, which as the name implies, fits an MM of normal distributions. Again, for the sake of simplicity, we will cover only a few of the many features of this function.

The algorithm is iterative, and thus requires initial guesses for the means/standard deviations of the two normal distributions, and the proportions of the two eruption types. I took the former (arguments **mu** and **sigma**) from the appearance of the histogram, and used equal weights for the latter (argument **lambda**[9]).

```
> mixout <- normalmixEM(faithful$eruptions,
   lambda=0.5,mu=c(55,80),sigma=10,k=2)
number of iterations= 7
> str(mixout)
List of 9
 $ x         : num [1:272] 3.6 1.8 3.33 2.28 4.53 ...
 $ lambda    : num [1:2] 0.36 0.64
 $ mu        : num [1:2] 2.05 4.3
 $ sigma     : num [1:2] 0.364 0.364

 ...
```

The **lambda** component of the return value indicates that about 36% of the eruptions are of type 1. The estimated mean eruption durations for the two eruption types are 2.05 and 4.3. (My initial guess for the standard deviations, 1.0, was about 3 times too high.)

2.4 Vector-Valued X

The most common mixture modeling is in *cluster analysis*, often referred to as *unsupervised learning*. We have multivariate data, say in a marketing application, and wish to find meaningful subgroups, say different types of customers. Again, we don't know what types are there, if there are any in some sense, but if we can find some, this may be very useful.

The **faithful** data is bivariate, with columns for both eruption duration and waiting time between eruptions. A plot, seen in Figure 3, does seem to show two groups. Of course, if there really are two groups, we can't tell for sure here which point belongs to which group, but again, in say, a marketing context, we just want to identify rough groups.

[9] Not to be confused with the λ_i in the battery example, just a coincidence in naming!

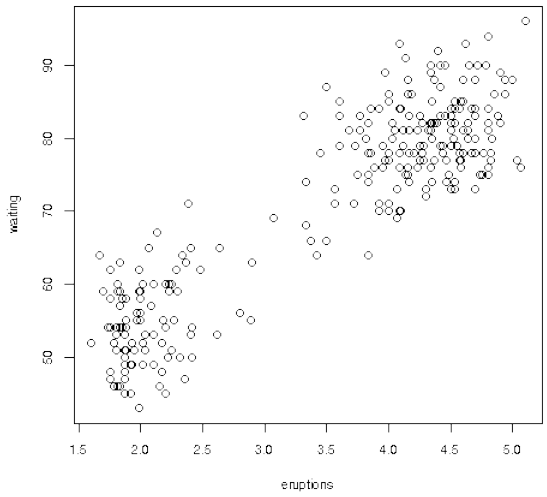

Figure 3: Old Faithful bivariate plot

In the univariate case, we assumed normal distributions for the components. The **mixtools** function **mvnormalmixEM()** fits a multivariate normal model. I tried running it completely on the basis of the argument defaults:

```
> mvnout <- mvnormalmixEM(fMaithful)
number of iterations= 12
> str(mvnout)
List of 9
 $ x         : num [1:272, 1:2] 3.6 1.8 3.33 2.28 4.53 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:272] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "eruptions" "waiting"
 $ lambda    : num [1:2] 0.356 0.644
 $ mu        :List of 2
  ..$ : num [1:2] 2.04 54.48
  ..$ : num [1:2] 4.29 79.97
 $ sigma     :List of 2
  ..$ : num [1:2, 1:2] 0.0692 0.4352 0.4352 33.6973
  ..$ : num [1:2, 1:2] 0.17 0.941 0.941 36.046
...
```

Since our data is bivariate, the estimated mean for each cluster is a vector of length 2, with a 2 $\times$

2 covariance matrix. The estimates are displayed in the above output. The **lambda** output again shows the estimated mixing proportions, similar to the ones we found in the univariate case.

One of the key issues is the number of groups to postulate, again deterimined informally, and again something we should do with an eye toward avoiding overfitting.

Cluster analysis is a vast topic, a major field in machine learning/data science. Estimation via the EM algorithm is only one of many methods to choose from. See *https://cran.r-project.org/web/views/Cluster.html* for an extensive choice of R libraries for clustering.

2.5 Overdispersion Models

Recall the following about the Poisson distribution family:

(a) This family is often used to model counts.

(b) For any Poisson distribution, the variance equals the mean.

In some applications in which we are modeling count data, condition (b) is too constraining. We want a "Poisson-ish" distribution in which the variance is greater than the mean, called an *overdispersion* model.

One may then try to fit a mixture of several Poisson distributions, instead of a single one. This does induce overdispersion, as we will now see.

The states here will be totally fictitious, just a vehicle to achieve an overdispersed model. Say the distribution of Y given $S = i$ is Poisson with parameter $\lambda_i, i = 1, 2, ..., k$. Then Y has a mixture distribution. Our goal here will be to show that Y is indeed overdispersed, i.e. has a large variance than mean.

By the Law of Total Variance (10)

$$\begin{aligned} Var(Y) &= E[Var(Y|S)] + Var[E(Y|S)] \quad (19) \\ &= E(\lambda_S) + Var(\lambda_S) \quad (20) \end{aligned}$$

Note again that in the above, the expression λ_S is a random variable, since its subscript S is random. The random variable λ_S takes on the values $\lambda_1, ..., \lambda_k$ with probabilities $q_1, ..., q_k$.

Note too that due to the fact that $Y|S$ has a Poisson distribution (it was constructed as such), its mean is λ_S. So (20) becomes

$$
\begin{aligned}
Var(Y) &= E(\lambda_S) + Var(\lambda_S) && (21)\\
&= E[E(Y|S)] + Var(\lambda_S) && (22)\\
&= EY + Var(\lambda_S) && (23)\\
& && (24)
\end{aligned}
$$

Thus

$$Var(Y) = EY + Var(\lambda_S) \geq EY \tag{25}$$

That last inequality comes from the fact that variances are nonnegative. In fact, as long as the λ_i are not all identical, their variance will be strictly positive. In other words,

$$Var(Y) > EY \tag{26}$$

so yes indeed, the given mixture model has the overdispersion property, as desired.

So, if one has count data in which the variance is greater than the mean, one might try using this model. Overdispersion is also of interest in some applications where Y is a continuous random variable.

In mixing the Poissons, there is no need to restrict to discrete S. In fact, it is not hard to derive the fact that if X has a gamma distribution with parameters r and p/(1-p) for some $0 < p < 1$, and Y given X has a Poisson distribution with mean X, then the resulting Y neatly turns out to have a negative binomial distribution.[10] In other words, the negative binomial family also has the overdispersion property.

3 Hidden Markov Models

As in MMs, we have an observable variable Y, and a state S, but now the state evolves in time. Thus the Y_j are no longer iid. Instead, the time pattern is assumed to be *Markovian*, or "memoryless," a property we assume about the states S_j:[11]

$$P(S_{k+1} = v_{k+1} \mid S_1 = v_1, S_2 = v_2, ...S_k = v_k) = P(S_{k+1} = v_{k+1} \mid S_k = v_k) \tag{27}$$

[10] Recall that this distribution family arises as the number of trials, e.g. number of coin flips, needed to accumulate m succeses, e.g. m heads.

[11] Technically, the Markovian nature of the S_j does not imply the same for the Y_j. For that, we need to assume, say, that conditionally on the S_j, the random variables $Y_1, Y_2, ...$ are independent.

In English,

> The probability of a future event, given the present and the past, depends only on the present.

Again the states S_i are unobserved, i.e. "hidden." Note that they may be real, as in our noisy network example, or just postulated, as in the geyser example. In our stock market example, for instance, one might postulate "bull" and "bear" moods among the traders.

3.1 The EM Algorithm

The situation here is largely analogous to that of MMs:

- We again need a model for the distribution of $Y|S$. In the geyser example, for instance, that could be Gaussian with the μ_i and σ_i as parameters.
- The analog of the parameter q is now the *transition matrix* of S, whose row i, column j element is $P(S_{m+1} = j \mid S_m = i)$, which as noted does not depend on m.

One difference, though, is that now we also need to estimate the state sequence $S_1, ..., S_n$ itself. This of course is of interest, as it is needed for predicting new states $S_{n+1}, S_{n+2}, ...$ and thus predicting $Y_{n+1}, Y_{n+2}, ...$ But also, the estimated $S_1, ..., S_n$ are needed as intermediate results in estimating θ and γ, as follows. The analogs of Steps 1 and 2 in Section 2.2 are:

Step 1: Use MLE to find estimates of the θ vector, as before. But now the maximization is much more complex, as it takes into account all possible the state sequences $S_1, ..., S_n$. In the geyser example, for instance, we find the sequence *and* the θ value that "makes our Y data most likely."

Step 2: As mentioned, in this step we estimate the distribution of S, in the form of the transition matrix. This can be done directly, since we have an estimated state sequence. Our estimate for the matrix entry in row 2, column 5, for instance, will be the proportion of indices i for which $S_{i+} = 5$, among those for which $S_i = 2$.

One major problem is that the number of possible state sequences can be enormous, s^n for a system with s states. But there are recursive algorithms that have been developed to better organize the computation, called the *forward* and *backward* algorithms, and to more efficiently perform the maximization, the *Viterbi* algorithm. Interested readers will find many detailed presentations of these algorithm on the Web.

3.2 The hmmr Package

There are various R packages for fitting HMMs. The one we present here is **hmmr**.

```
> z <- hmmr::hmm(faithful$eruptions,2)
> summary(z)
Initial state probabilities model
pr1 pr2
  1   0

Transition matrix
        toS1  toS2
fromS1 0.479 0.521
fromS2 0.938 0.062

Response parameters
Resp 1 : gaussian
    Re1.(Intercept) Re1.sd
St1           4.289  0.413
St2           2.036  0.263

# z is an S4 class, one of whose components, posterior, is a data frame
> z@posterior$state
  [1] 1 2 1 2 1 2 1 1 2 1 2 1 1 2 1 2 2 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 2 2
 [38] 1 2 1 1 2 1 2 1 1 1 2 1 2 1 1 2 1 2 1 1 2 1 1 2 1 2 1 2 1 1 1 2 1 1 2 1 1
 [75] 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 2 1 2 1 1 1 2 1 2 1 2 1 1 2 1 2 1 1 1
[112] 2 1 1 2 1 2 1 2 1 2 1 1 2 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 1 2 1 1 1 2 1 2
[149] 1 2 1 1 2 1 1 1 1 1 2 1 2 1 2 1 1 1 2 1 2 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2
[186] 1 1 2 1 2 1 2 1 1 1 1 1 1 2 1 2 1 1 2 1 2 1 1 2 1 2 1 2 1 1 1 2 1 2 1 2 1
[223] 2 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 1 2 1 2 1 2 1 1 2 1 2 1 2 1 1 1 1 1 1 1 2
[260] 1 1 1 2 1 2 2 1 1 2 1 2 1
```

Our call to **hmm()** specifies our data Y_i, and requests a 2-state model. We have taken the default value, NULL, for the **family** argument, which specifies the distribution of $Y|S$. The value NULL is taken by **hmm()** to mean the Gaussian family.

We see that under this model, if the current state is, say 2, i.e. there was just a type 2 eruption, then almost certainly the next eruption will be of type 1. On the other hand, after a type 1 eruption, there are approximately equal chances that the next eruption will be of type 1 or 2.

The mean of $Y|S = 1$ is about 4.3, while the corresponding value for state 1 is abou 2.0. These are

close to what we obtained above with the mixture model. The estimated proportions of the two types, 0.36 and 0.64, are also similar to the earlier result.

The **hmmr** package makes heavy use of an earlier package by the same authors, **depmixS4**. The "S4" part of that latter name alludes to the fact the main function of of the package, **depmix()** returns objects of R class S4. One of the components in this object type, **posterior**, contains information about the final estimated state sequence. The software has found, for instance, that the most likely scenario was that $S_1 = 1, S_2 = 2, S_3 = 1, ...$[12]

[12]Readers who know Bayesian statistics should not interpret this wording to me that this is "Bayesian" analysis in the sense of subjective prior distributions, which is not the case. Of course, since we are working with various conditional and unconditional distribution, Bayes' *Rule* of probability *is* used, but not in the subjective sense.