**Hidden Markov Models: Extensions**

In my previous project, I developed the reasoning behind Hidden Markov Models (HMM) and developed a way of determining the most likely sequence of states that resulted in an observed sequence of emissions using the Viterbi algorithm. This project will explore extensions to the Hidden Markov Model, such as the Forwards-Backwards algorithm and the Baum-Welch algorithm.

Consider a discrete time, discrete state-space Hidden Markov chain with:

: the state at time *t* (unknown in the HMM) of *n* possible states,   
 : the emission at time *t* of *m* possible emissions,

P(0) : initial probabilities of being in the states. ***P***  : a *nxn* transition matrix of transitions between states, giving P( for the ith row and jth column.   
 ***E*** : a *nxm* matrix containing P( in the ith row, jth column, the emission probabilities. Note that the row sums must sum to 1.

Given a set of observed emissions, , the Viterbi algorithm gives us the most likely sequence of states, that resulted in the observed emissions, given ***P*** and ***E.***

Although this decoding algorithm is very useful, there may be other aspects of the HMM we are interested in. One such problem is, given a set of observed emissions , what is the probability of the observed emissions? This problem can be solved using the law of total probability, i.e.:

where refers to any vector of state sequences. This can be computationally intensive – for many steps, this will involve summing over an exponentially large number of possible state sequences. Thus, an algorithmic approach will be useful to make this result practical to obtain for large *T.* This is a similar problem to the Viterbi algorithm, but not exactly the same. This can be solved using the *forward algorithm*, as described below. Note that we can also write the above as:

Let’s call =

It follows that our desired probability of the sequence is equal to the sum over all states of the above quantity.

Then, using properties of joint probability:

But is conditionally independent of everything but and conditionally independent on everything but . So we can write this simply as:

It follows, then, that we can compute each recursively. This will eliminate the need to look at the probability of all states. This has a computation time of which is far faster than doing it for each possible sequence, which is. After computing this for each possible state at time T, we sum the probabilities to get the probability of the observed sequence. Note that , where is to be considered the initial probabilities of entering the chain, predefined by the problem.

Consider the following example of a fair and biased coin, where:

Initial probabilities are equal, .5 for fair and .5 for biased

***E =***

We observe the sequence HTHHTTHH and want to determine the probability of this sequence occurring given our HMM. First,

So it follows that the probability of a head being observed, alone, is the sum of these partial probabilities, or .385. To determine the probability of the full sequence, we continue this recursively, considering the results of the function at the last time step.

We can also simplify this using matrix multiplication. If we consider as a *nx1* vector pertaining to each of the *n* states, we can calculate the following:

x **P**

Where j refers to a row vector of the jth column pertaining to the observed value at time *t*. This can help to simplify the calculations. Here, x refers to component-wise multiplication. I have included a user-defined function to determine this probability recursively for the problem at hand:

forwardalgor <- function(obs, trans,emissions,init){  
 alphas <- list(1,init\*emissions[,obs[1]])  
   
 for(i in 2:(length(obs) + 1)){  
 alphasum <- as.numeric(trans%\*%alphas[[i]])  
 alphas[[i + 1]] <- emissions[,obs[i-1]]\*alphasum  
 }  
 print("Sequences:")  
 print(alphas)  
 print("Final Probability")  
 print(sum(alphas[[length(obs)+1]]))  
 return(sum(alphas[[length(obs)+1]]))  
}  
  
forwardalgor(c(1,2,1,1,2,2,1,1),rbind(c(.9,.1),c(.95,.05)),rbind(c(.5,.5),c(.25,.75)),c(.5,.5))

## [1] "Sequences:"  
## [[1]]  
## [1] 1  
##   
## [[2]]  
## [1] 0.250 0.125  
##   
## [[3]]  
## [1] 0.1187500 0.0609375  
##   
## [[4]]  
## [1] 0.05648438 0.08689453  
##   
## [[5]]  
## [1] 0.02976270 0.01450122  
##   
## [[6]]  
## [1] 0.014118274 0.007249905  
##   
## [[7]]  
## [1] 0.006715719 0.010331142  
##   
## [[8]]  
## [1] 0.003538630 0.005172367  
##   
## [[9]]  
## [1] 0.0018510021 0.0009050793  
##   
## [[10]]  
## [1] 0.0008782049 0.0004509265  
##   
## [1] "Final Probability"  
## [1] 0.002756081

This gives us our result – there is about a .2% chance of this observation being output by this particular HMM. This can be useful for numerous applications: consider if one was testing against *n* possible HMMs. We could then find which HMM is the more likely fit by considering

. Bayes’ Rule tells us that:

If we consider our *n* models as equally likely, we can then determine which model is most probable given the observations simply by considering the likelihood in the fashion calculated above. That is, . This relatively simple result and simple algorithm gives us a method of comparing and optimizing our fit. Also, since the probabilities here can become quite small, the fact that a logarithmic function is monotonically increasing means we can also compare the logarithms, in order to avoid machine zeros.

The closely related, but not exactly equivalent, Forwards-Backwards algorithm attempts to find probability of a particular state of a *particular* time point given all of the observations, i.e. :

Notice that we can separate this into two parts,

This is due to the fact that is conditionally independent of given and that is a multiplicative constant which does not change with a selection of . This means we can normalize this probability at each point to ensure that . Notice that as described earlier.

Notice that is a similar proposition, but backwards. That is, we can call =  
 =

This is obtained using the chain rule and the conditional independencies of the HMM. Notice now that and we can now see the recursive nature of this function. That is,

We take for all states, as this can be considered the exit probability.

Intuitively, this means that we are considering, for some intermediate time step, the probabilities of getting there from zero and the probabilities of “back tracking” from T. This gives us the total probability of an intermediate state based on our observations.

The implementation is as follows:

Compute the forward probabilities (by the previously stated algorithm) until time step *k.* For each step, normalize the results so that the probabilities sum to 1.Then, compute the backward probabilities from T to k. Normalize these at each step as well. Then, multiply the two results and normalize to get the probabilities of being in each state after *k* time steps.

To calculate this, we may use the following formula:

x

Notice that this is similar to the formula from before for the forwards portion, but the multiplications are “flipped.” Thus we can implement the Forward-Backwards algorithm as follows:

bkforwardalgor <- function(obs, k, trans,emissions,init){  
 alphas <- list(1,init\*emissions[,obs[1]]/sum(init\*emissions[,obs[1]]))  
 betas <- list(rep(1,nrow(trans)))  
   
 for(i in 3:(k+1)){  
 alphasum <- as.numeric(t(trans)%\*%alphas[[i - 1]])  
 alphas[[i]] <- emissions[,obs[i-1]]\*alphasum  
 alphas[[i]] <- alphas[[i]]/sum(alphas[[i]])  
 }  
 for(i in 2:(length(obs) - k + 1)){  
 betasum <- emissions[,obs[length(obs)-i+2]]\*betas[[i-1]]  
 betas[[i]] <-as.numeric(trans%\*%betasum)  
 betas[[i]] <- betas[[i]]/sum(betas[[i]])  
 }  
 pointalph <- alphas[[(k+1)]]  
 pointbet <- betas[[(length(obs) - k + 1)]]  
   
 return(list(forward = alphas, backward = betas, probstates = pointalph\*pointbet/sum(pointalph\*pointbet)))  
}  
  
  
  
bkforwardalgor(c(1,2,1,1,2,2,1,1),3,rbind(c(.9,.1),c(.95,.05)),rbind(c(.5,.5),c(.25,.75)),c(.5,.5))

## $forward  
## $forward[[1]]  
## [1] 1  
##   
## $forward[[2]]  
## [1] 0.6666667 0.3333333  
##   
## $forward[[3]]  
## [1] 0.88 0.12  
##   
## $forward[[4]]  
## [1] 0.95068206 0.04931794  
##   
##   
## $backward  
## $backward[[1]]  
## [1] 1 1  
##   
## $backward[[2]]  
## [1] 0.4935065 0.5064935  
##   
## $backward[[3]]  
## [1] 0.4936839 0.5063161  
##   
## $backward[[4]]  
## [1] 0.5064686 0.4935314  
##   
## $backward[[5]]  
## [1] 0.5055779 0.4944221  
##   
## $backward[[6]]  
## [1] 0.4933575 0.5066425  
##   
##   
## $probstates  
## [1] 0.94942121 0.05057879

So, at the third time step, we are close to 95% certain that we were in the first state (Heads), given all of the observations.

This all leads us to the Baum-Welch algorithm, which is a way of estimating parameters of a HMM given a sequence of emissions. Let where is the parameters of the distribution (the transition matrix, the emission matrix and the initial probabilities.) Note that we have already calculated this in the previous exercise using the forwards backwards algorithm.

Further, let **.**

This represents the probability of entering state *i* from the left, going from state *i* to *j,* exiting state j from the right, and emitting the emission at the t+1 time step from state j. All of this culminates in the joint probability of two consecutive states.

Then, using an initial set of parameters we can calculate these values and update our parameters to find a local maxima. Our updates are as follows:

Initial probabilities = . This is the expected frequency spent in state *i* at time 1.

The new transition matrix, **P’**, will have entries:

This is the expected number of transitions from state *i* to *j* compared to the expected total number of transitions away from state *i* (including itself).

The new emission matrix, **E’**, will have entries:

Where is an indicator function which is 1 if the emission was the jth emission and zero otherwise. This calculates the average amount of time an emission *j* was emitted in state *i* divided by the average amount of time it was in state *i* in general.

We then iterate this process until we are within a threshold to determine the optimal parameters for the model given only an observed sequence.

While I have not implemented this algorithm directly here, I have used the *HMM* package in R to determine this. Here is the output:

library(HMM)  
chk <- initHMM(States = c("F","B"),Symbols = c("H","T"),startProbs = c(.5,.5),transProbs = rbind(c(.9,.1),c(.95,.05)), emissionProbs = rbind(c(.5,.5),c(.25,.75)))  
baumWelch(chk,c("H","T","H","H","T","T","H","H"))

## $hmm  
## $hmm$States  
## [1] "F" "B"  
##   
## $hmm$Symbols  
## [1] "H" "T"  
##   
## $hmm$startProbs  
## F B   
## 0.5 0.5   
##   
## $hmm$transProbs  
## to  
## from F B  
## F 0.8356626 1.643374e-01  
## B 1.0000000 4.663436e-19  
##   
## $hmm$emissionProbs  
## symbols  
## states H T  
## F 0.530374 4.696260e-01  
## B 1.000000 8.442496e-29  
##   
##   
## $difference  
## [1] 5.089028e-01 2.405566e-01 1.753672e-01 1.257996e-01 7.898661e-02  
## [6] 4.304625e-02 2.202420e-02 1.204572e-02 7.970542e-03 6.489383e-03  
## [11] 5.969237e-03 5.734349e-03 5.568848e-03 5.414358e-03 5.255341e-03  
## [16] 5.087867e-03 4.911547e-03 4.727255e-03 4.536362e-03 4.340440e-03  
## [21] 4.141117e-03 3.939997e-03 3.738615e-03 3.538400e-03 3.340654e-03  
## [26] 3.146539e-03 2.957068e-03 2.773104e-03 2.595359e-03 2.424400e-03  
## [31] 2.260659e-03 2.104445e-03 1.955948e-03 1.815259e-03 1.682381e-03  
## [36] 1.557239e-03 1.439694e-03 1.329553e-03 1.226584e-03 1.130517e-03  
## [41] 1.041062e-03 9.579095e-04 8.807409e-04 8.092317e-04 7.430572e-04  
## [46] 6.818960e-04 6.254334e-04 5.733633e-04 5.253903e-04 4.812312e-04  
## [51] 4.406157e-04 4.032872e-04 3.690029e-04 3.375342e-04 3.086663e-04  
## [56] 2.821979e-04 2.579412e-04 2.357210e-04 2.153745e-04 1.967503e-04  
## [61] 1.797084e-04 1.641190e-04 1.498622e-04 1.368276e-04 1.249129e-04  
## [66] 1.140244e-04 1.040755e-04 9.498672e-05 8.668506e-05 7.910346e-05  
## [71] 7.218037e-05 6.585937e-05 6.008875e-05 5.482110e-05 5.001304e-05  
## [76] 4.562483e-05 4.162013e-05 3.796567e-05 3.463103e-05 3.158841e-05  
## [81] 2.881238e-05 2.627969e-05 2.396914e-05 2.186131e-05 1.993849e-05  
## [86] 1.818450e-05 1.658456e-05 1.512520e-05 1.379408e-05 1.257997e-05  
## [91] 1.147261e-05 1.046263e-05 9.541475e-06 8.701358e-06 7.935157e-06  
## [96] 7.236378e-06 6.599096e-06 6.017905e-06 5.487874e-06 5.004503e-06

Here, we see that the algorithm suggests changing the transition probabilities to almost never transition when in the Biased state, and to transition more frequently when in the fair state. It also suggests that the emission probabilities be heads nearly in all cases when it is biased. This is troublesome, since we knew (by our definition) that the emissions would be fair for the fair coin and biased otherwise. Still, this generalizes the problem and allows us to attempt to fit optimal HMMs to a given set of emissions.

Using the Viterbi algorithm is known as “decoding” – we are attempting to determine the most likely state sequence given a sequence of observed emissions. Determining the most likely state at a time step is known as “filtering” (when the step is the final step, *T)* or “smoothing” (for some intermediate step, *k*). This is what the forward-backward algorithm was used for. Attempting to determine the best fitting parameters is known as “training”. This is the motivation of the Baum-Welch algorithm. Together, these three algorithms provide the basis for many applications of the HMM model.