COMP 5970/6970-004
Computational Biology: Genomics and Transcriptomics
Lecture notes 9: 2/10/2022

Haynes Heaton

Spring, 2022

Lecture Objectives

- Optimal state sequence vs posterior state probability
- Forward-Backward algorithm
- Implementation

Optimal state sequence vs posterior state probability

In many cases with a hidden markov model, all we want is the optimal state sequence. But maybe you want some estimate of the certainty of that state sequence. Or perhaps you are interested in sampling different state sequences based on their posterior probabilities. Also if we do not know the parameters of an HMM, these can be estimated by treating an HMM as a time series mixture model and estimating the parameters by maximizing the likelihood of the data given those parameters. Remember this is maximum likelihood estimation (MLE) which may be solved with expectation maximization (EM). In order to do this, the expectation step requires the posterior probabilities that the state sequence passes through state s_i at time t for all i and t. Let's make some definitions first.
Definitions

- s_i: State i
- t: Position t in the sequence. Hidden Markov models are often used in time series data, and while we are using them for sequence data, p for position is overloaded with probability
- n: length of the observed data
- D_t: observed data at time t (CG or not CG)
- $s_{i,j}$: Transition probability of going from state i to state j.
- $\pi_i = p(s_i)$ the priors for each state

So the value we want is

$$p(s_i(t)|D) \tag{1}$$

and if we do a Bayesian flip we get

$$p(s_i(t)|D) = \frac{p(D|s_i(t))p(s_i)}{\sum_j(p(D|s_j(t))p(s_j))} \tag{2}$$

But how can we compute the likelihood $p(D|s_i(t))$ given all possible state sequences? Enter the forward-backward algorithm.

0.1 Forward-Backward algorithm

Recall our previous discussion of mixture models. In this simplest case the likelihood of a mixture model may look like $p(D|M_1)p(M_1) + p(D|M_2)p(M_2)$. In a hidden markov model, the states (or models) are hidden, thus we can treat this as a mixture model. The only difference is that there is a time element and transition probabilities. At time point 0, the likelihood of all state sequences from 0..0 is very simply $p(D_0|s_i)$ for each state. For time point 1, we have a mixture of the previous possible states. In a 2 state HMM we have

$$p(D_{0..1}|s_i) = p(D_0|s_0)s_{0,i}p(D_1|s_i) + p(D_0|s_1)s_{1,i}p(D_1|s_i) \tag{3}$$

Just summing over the previous likelihoods times the transition probabilities times the likelihood of the current observed data given the current state. So this is a mixture marginalizing across the previous states. And the next time point, we will do the same thing, but it will be a mixture over two things that are already a mixture over two things etc etc etc. So in this way, we are marginalizing across all possible state sequences up until now. With this in mind, we can generalize this to any time point.

$$p(D_{0..t}|s_i) = p(D_{0..t-1}|s_0)s_{0,i}p(D_t|s_i) + p(D_{0..t-1}|s_1)s_{1,i}p(D_t|s_i) \tag{4}$$

And any number of states

$$p(D_{0..t}|s_i) = \sum_j (p(D_{0..t-1}|s_j)s_{j,i}p(D_t|s_i)) \tag{5}$$

For those who prefer a slightly more visual and computational look at things, recall our optimal path HMM dynamic programming table. It looked like this.

t

s_0	π_0	$max_j(hmm[j][t-1] * s_{j,0})p(D_t|s_0)$	..	..	..	..	..
s_1	π_1	$max_j(hmm[j][t-1] * s_{j,1})p(D_t|s_1)$	..	..	..	..	..

Well, the forward algorithm is very similar and looks like the following.

t

s_0	π_0	$sum_j(forward[j][t-1] * s_{j,0})p(D_t|s_0)$	..	..	..	..	..
s_1	π_1	$sum_j(forward[j][t-1] * s_{j,1})p(D_t|s_1)$	..	..	..	..	..

This is the forward algorithm, getting the likelihood marginalized across all state sequences $0..t$. Okay, but this value isn't the actual likelihood we are wanting. We want $p(D|s_i(t))$ marginalized on all state sequences. Currently we are only marginalizing across state sequences $0..t$. What about the state sequences $t..n$? This is where the backward algorithm comes into play.

Backward algorithm The backward algorithm will compute the following likelihood $p(D_{n..t}|s_i)$ marginalizing across all state sequences from n backwards to t. We use a new dynamic programming table and start the $n+1$ cell with probability 1. Whereas with the forward algorithm, we started with the priors and the data likelihood, here the nth position will just the be data likelihood. The reason for this is that we will later be combining the forward and backward likelihoods for a final likelihood and we only want to apply the priors once.

$$p(D_{n..t}|s_i) = sum_j(p(D_{n..t+1}|s_j)s_{i,j}p(D_t|s_i)) \tag{6}$$

As you can see, otherwise it is very similar to the forward algorithm, but in reverse, marginalizing over state sequences $t+1..n$ or $n..t+1$ as written because that is the order in which they are calculated. And also note the ordering of indices in the transition probabilities as we are calculating backwards, they should also be reversed though most times these probabilities are symmetrical and this detail wouldn't matter.

t

s_0	..	..	..	..	..	$sum_j(backward[j][t+1] * s_{0,j})p(D_t|s_0)$	1
s_1	..	..	..	..	..	$sum_j(backward[j][t+1] * s_{1,j})p(D_t|s_1)$	1

So now that we have our likelihoods marginalizing on paths $0..t$ and $n..t$, we can just multiply them together to get the likelihood marginalizing across all paths $0..n$.

$$p(D|s_i(t)) = p(D_{0..t}|s_i)p(D_{n..t}|s_i) \tag{7}$$

So this allows us to finish our initial equation

$$p(s_i(t)|D) = \frac{p(D|s_i(t))p(s_i)}{\sum_j(p(D|s_j(t))p(s_j))} \tag{8}$$

as

$$p(s_i(t)|D) = \frac{p(D_{0..t}|s_i(t))p(D_{n..t}|s_i(t))}{\sum_j(p(D_{0..t}|s_j(t))p(D_{n..t}|s_j(t)))} \tag{9}$$

Note that I have left off the priors here as they are baked in to the forward algorithm already. This may seem complicated, but its really just normalizing the state probabilities for a given time point to sum to 1. We could probably make this look simpler if we give a notation for the forward and backward algorithms.

- $F_{s_i}(t) = p(D_{0..t}|s_i)$
- $B_{s_i}(t) = p(D_{n..t}|s_i)$

Now we have

$$p(s_i(t)|D) = \frac{F_{s_i}(t)B_{s_i}(t)}{\sum_j(F_{s_j}(t)B_{s_j}(t))} \tag{10}$$

Which is the posterior probability of being in state i at time t across all possible state sequences weighted on their likelihood. Okay, let's implement it on our CpG island example.

```
import numpy as np
import pandas as pd
import scipy
from plotnine import *

## /Library/Frameworks/R.framework/Versions/4.1/Resources/library/reticulate/python/rpytools/loader.py:3
```

Let's load some sequence from the human genome containing a CpG island.

```
cpgtest = ""
with open("cpgtest.txt") as infile:
    cpgtest = infile.readline()
len(cpgtest)

## 3749
```

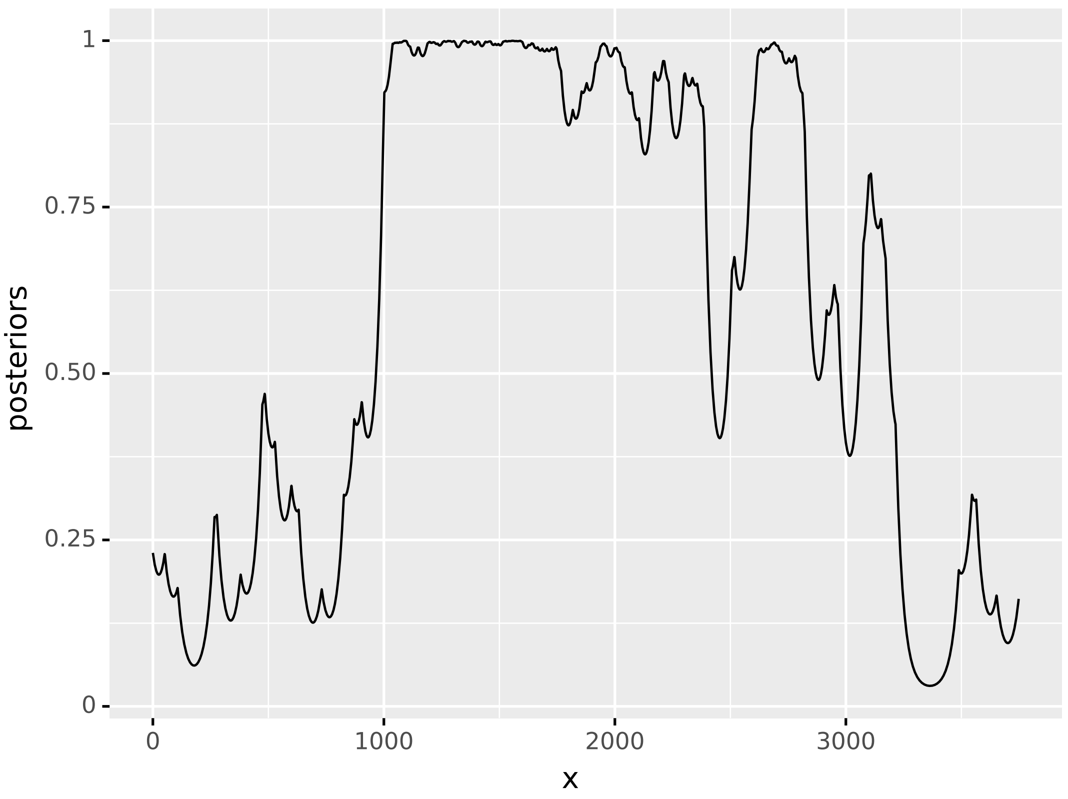
And write a function to find the optimal state sequence. Keep in mind that we need to compute this in log space because of numerical stability of many probabilities multiplied together.

```
def forward_backward(observed, priors, transitions, probabilities):
    # priors is 2 values, one for each state
    # transitions s_{i,j} is 2x2 for from state x to state
    # probabilities 2x2 with state x [cg_prob, 1-cg prob]
    forward = np.zeros((2,len(observed)))
    forward[0][0] = np.log(priors[0]) # compute in log space for numerical stability
    forward[1][0] = np.log(priors[1])

    log_transitions = np.zeros((2,2))
```

```
log_probabilities = np.zeros((2,2)) # convert inputs to log space
for i in range(2):
    for j in range(2):
        log_transitions[i][j] = np.log(transitions[i][j])
        log_probabilities[i][j] = np.log(probabilities[i][j])

# forward algorithm
for t in range(1, len(observed)):
    CG = 1 # to index into probabilities, 1th entry is for no CG
    if observed[t-1:t+1] == "CG":
        CG = 0 # 0th entry is for CG

    # loop over current state
    for current_state in range(2):
        # build log probs array for coming from each previous state with list comprehension.
        # in log space we add instead of multiply
        log_probs = [forward[previous_state][t-1] +
                      log_transitions[current_state][previous_state] +
                      log_probabilities[current_state][CG]
                     for previous_state in range(2)]
        # because we want to add probabilities here
        # but we are in log space, we use a numerically stable
        # function logsumexp
        forward[current_state][t] = scipy.special.logsumexp(log_probs)

# backward algorithm
backward = np.zeros((2, len(observed)))
backward[0][len(observed)-1] = np.log(priors[0])
backward[1][len(observed)-1] = np.log(priors[1])
for t in reversed(range(len(observed)-1)):
    CG = 1
    if observed[t:t+2] == "CG":
        CG = 0
    for current_state in range(2):
        log_probs = [backward[previous_state][t+1] +
                     log_transitions[previous_state][current_state] +
                     log_probabilities[current_state][CG] for previous_state in range(2)]
        backward[current_state][t] = scipy.special.logsumexp(log_probs)

# normalization
posterior = np.zeros((2, len(observed)))
x = []
posteriors = []
for t in range(len(observed)):
    # build an array of F*B for all states at time t
    denoms = [forward[state_j][t] + backward[state_j][t] for state_j in range(2)]
    # sum the probabilities, but in log space logsumexp
    denom = scipy.special.logsumexp(denoms)

    for state in range(2):
        # posterior is F_s_i(t)B_s_i(t)/sum_j(F_s_j(t)B_s_j(t))
        # but in log space we add and subtract.
        posterior[state][t] = np.exp(forward[state][t] + backward[state][t] - denom)
        x.append(t)
        posteriors.append(posterior[0][t]) #return posteriors for 0th state for visualization
    return(x, posteriors)
```

```
priors = [0.5, 0.5]
transitions = [[0.995,0.005],[0.005,0.995]]
probabilities = [[0.04,0.96],[0.01,0.99]] # 0th state is cp_island, 1th state is background genome

(x, posteriors) = forward_backward(cpgtest, priors, transitions, probabilities)
```

```
df = pd.DataFrame({'x':x,'posteriors':posteriors})
ggplot(df)+geom_line(aes(x='x', y='posteriors'))

## <ggplot: (8760425418277)>
```



So here we have the posterior probability of being in the CpG island state across this data.