COMP 5970/6970-004
Computational Biology: Genomics and Transcriptomics
Lecture notes 11: 2/17/2022

Haynes Heaton

Spring, 2022

Lecture Objectives

- Baum Welch algorithm

Hidden Markov model as time series mixture model

First, let's revisit our definitions so that there is no confusion on the notation. **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.
- p_i: probability of seeing CG from state i
- $F_{i,t}$: forward algorithm calculation of $p(D_{0..t}|s_i)$ the likelihood of the data if in state i at time t given all possible state sequences $0..t$.
- $B_{i,t}$: backward algorithm calculation of $p(D_{n..t}|s_i)$ the likelihood of the data if in state i at time t given all possible state sequences from $t..n$.

Recall our friend the mixture model in which each data point could be produced from multiple underlying distributions. In the simplest case, this would be two distributions and the marginal likelihood of the data would be the following.

$$\mathcal{L} = p(D|M_1)p(M_1) + p(D|M_2)p(M_2) \tag{1}$$

Now let us look at the update rule for the forward algorithm for a 2 state HMM.

$$p(D_{0..t}|s_i) = p(D_{0..t-1}|s_1)s_{1,i}p(D_t|s_i) + p(D_{0..t-1}|s_2)s_{2,i}p(D_t|s_i) \tag{2}$$

The likelihood of the data given being in state i at time t is a mixture of being in either state at the previous time point, making the state transition, and observing the data at time t. So an HMM is essentially a time series mixture model.

In lecture 5 and 10, we showed that we could estimate the parameters of the underlying distribution of a mixture model by starting with some random initial parameters and changing those parameters to maximize the likelihood of the data. This is **maximum likelihood estimation** and we accomplished this with the **expectation maximization** algorithm. In this case, our hidden Markov model has transition probability parameters $s_{i,j}$ and the parameters of the underlying distribution, which for the CpG island example are *Bernoulli* distributions (coin flip distribution) with a probability p.

$$\underset{p_1,p_2,s_{i,j}}{\operatorname{argmax}}(\mathcal{L}) \tag{3}$$

Recall the steps involved in the previous applications we have gone over to the expectation maximization.

1. Initialize the distribution parameters we want to solve randomly within the range that is reasonable for them. So here that may be p_1 which will represent the CpG island state to something higher than p_2 but of course in $(0..1)$.
2. Find the posterior probability that each data point was generated from each distribution. In our hidden Markov model, we also have a time component. So here, we want the posterior probability of being in state i at time t. This is given to us by the forward-backward algorithm
3. Update the parameters in some fashion that guarantees that the total likelihood increases. This uses the posterior probabilities of each data point to contribute to new parameter estimations in a weighted fashion. They contribute to each distribution's parameters according to the posterior probabilities from step 2.
4. Repeat steps 2-3 until the parameters don't change significantly (this is called convergence)

1 Expectation

For the case of the hidden Markov model, the expectation step is the forward-backward algorithm. To review, the posterior probability of being in state i at time t is

$$p(s_i(t)|D) = \frac{p(D|s_i(t))p(s_i)}{\sum_j(p(D|s_j(t))p(s_j))} \tag{4}$$

But we must have a way of calculating the likelihood of the data given being in state i at time t given all possible state paths. We do this with the forward and backward algorithms. First the forward computes the likelihood of being in state i at time t given all state sequences from $0..t$. For the first step we have

$$p(D_{0..1}|s_i) = p(D_0|s_0)s_{0,i}p(D_1|s_i) + p(D_0|s_1)s_{1,i}p(D_1|s_i) \tag{5}$$

And for each subsequent step we have

$$p(D_{0..t}|s_i) = p(D_{0..t-1}|s_0)s_{0,i}p(D_t|s_i) + p(D_{0..t-1}|s_1)s_{1,i}p(D_t|s_i) \tag{6}$$

Then the backward algorithm calculates the likelihood of being in state i at time t given all possible state sequences $n..t$.

$$p(D_{n..t}|s_i) = sum_j(p(D_{n..t+1}|s_j)s_{i,j}p(D_t|s_i)) \tag{7}$$

And finally, to get the total likelihood taking into account all state sequences $0..t..n$, we multiply the values at time t for the forward likelihood by the values at time t for the backward likelihood. And to get the posterior probability of being in state i at time t, we normalize this across all states for each time t.

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

2 Maximization

So now comes the tricky part. First let's start with the easier one, updating the probabilities. Basically we just want the probability of seeing a CG sequence in the data at time t weighted on the posterior probability of being in each state at time t and then let all times t contribute to this.

$$\hat{p_i} = \frac{\sum_t p(s_i(t)|D) \text{ if } D_t = \text{ CG else } 0}{\sum_t p(s_i(t)|D)} \tag{9}$$

Now the update for the transitions is more complicated. Let's first build the probability of transitioning from state i at time t to state j at time $t+1$.

$$p(s_i(t), s_j(t+1)|D) = \frac{F_{i,t} s_{i,j} B_{j,t+1}}{\sum_k \sum_m F_{k,t} s_{k,m} B_{m,t+1}} \tag{10}$$

Now that we have the probability of transitioning from state i to j at each time point, we can make an update for the transition probabilities by making a weighted average across all time points based on the posterior probability of being in state i at time t.

$$\hat{s_{i,j}} = \frac{\sum_t p(s_i(t), s_j(t+1)|D)}{\sum_t p(s_i(t)|D)} \tag{11}$$

So how does this translate into code?

```
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.

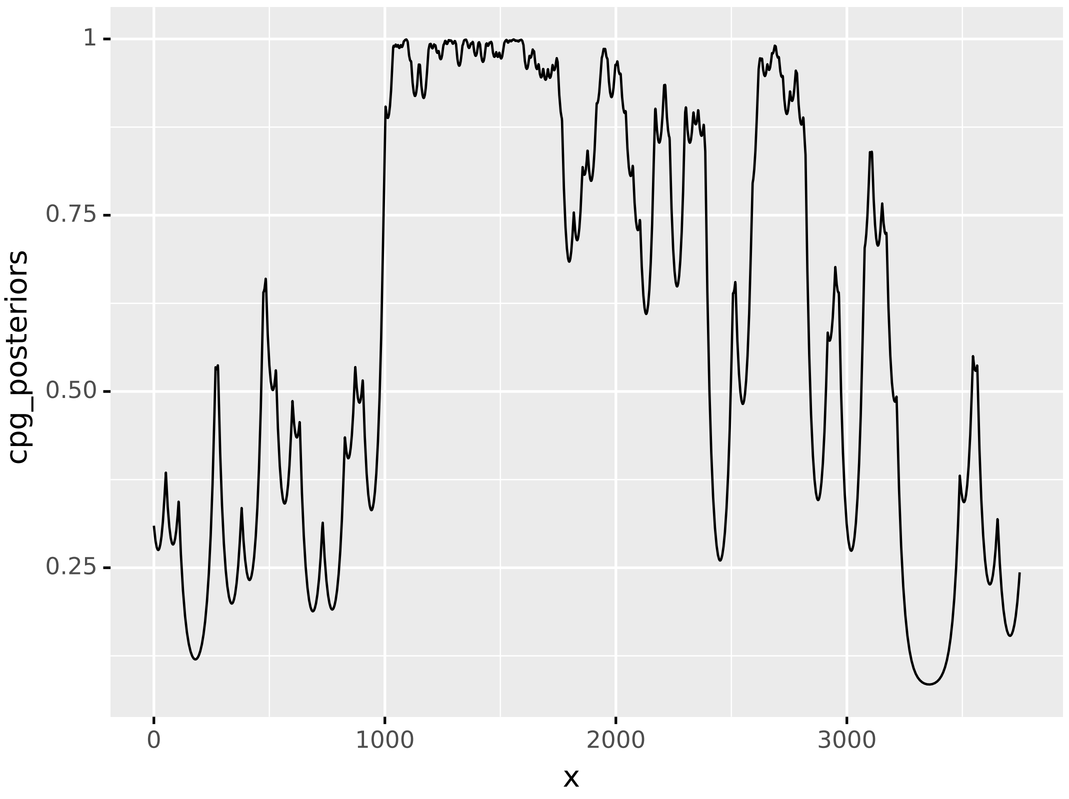
```
cpgtest = ""
with open("cpgtest.txt") as infile:
    cpgtest = infile.readline()
len(cpgtest)

## 3749
```

And the expectation step is stimply the forward-backward algorithm the same as in lecture 9.

```
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 = [] # for visualization
    posteriors = [] # for visualization
    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(x, posteriors, forward, backward, posterior)
```

And we can visualize these posteriors

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

(x, cpg_posteriors, forward, backward, posterior) = forward_backward(cpgtest, priors, transitions, proba
df = pd.DataFrame({'x':x,'cpg_posteriors':cpg_posteriors})
ggplot(df)+geom_line(aes(x='x', y='cpg_posteriors'))

## <ggplot: (8770814263372)>
```



Now for the maximization step.

```
def maximization(observed, posteriors, forward, backward, transitions):
    probability_numerators = np.zeros(2)
    probability_denominators = np.zeros(2)
    transitions_numerators = np.zeros((2,2))
    transitions_denominators = np.zeros((2,2))
    log_transitions = np.zeros((2,2))
    for i in range(2):
        for j in range(2):
            log_transitions[i][j] = np.log(transitions[i][j])
    for t in range(1, len(observed)):
        cg_indicator = 0
        if observed[t-1:t+1] == "CG":
            cg_indicator = 1
        transitions_denom_tmp = []
        transitions_numerator_tmp = np.zeros((2,2))
        for current_state in range(2):
            probability_numerators[current_state] += cg_indicator * posteriors[current_state][t]
            probability_denominators[current_state] += posteriors[current_state][t]
            for previous_state in range(2):
                transitions_numerator_tmp[previous_state][current_state] = (
                    forward[previous_state][t-1] +
                    log_transitions[previous_state][current_state] +
                    backward[current_state][t])
                transitions_denom_tmp.append(transitions_numerator_tmp[previous_state][current_state])
        transition_denom = scipy.special.logsumexp(transitions_denom_tmp)
        for current_state in range(2):
            for previous_state in range(2):
                log_transition = (transitions_numerator_tmp[previous_state][current_state] -
                                  transition_denom)
                transitions_numerators[previous_state][current_state] += np.exp(log_transition)
                transitions_denominators[previous_state][current_state] += posteriors[current_state][t]

    new_transitions = transitions_numerators / transitions_denominators
    new_probabilities = probability_numerators / probability_denominators
    return(new_probabilities, new_transitions)
```

And we can run it and view the output.

```
(x, cpg_posteriors, forward, backward, posterior) = (
    forward_backward(cpgtest, priors, transitions, probabilities))
(probabilities, transitions) = maximization(cpgtest, posterior, forward, backward, transitions)
print(probabilities)

## [0.05818046 0.0115036 ]

print(transitions)

## [[0.9922961  0.01037422]
##  [0.00706861 0.99055452]]
```

So you can see the data support a higher CG rate in the CpG islands and a more harsh transition probability to alternate states. This may go further with additional iterations.