CS262 Lecture 7: HMMs continued

Scribe: Justin Lee

January 27, 2015

1 VITERBI, FORWARD, BACKWARD

Concept check, which is bigger:

- $V_k(i)$ in the DP matrix for Viterbi, or
- $f_k(i)$ in the DP matrix for the Forward algorithm?

$\rightarrow f$ is a sum of all the state sequences from π_i to π_{i-1}, for generating the first letters of x, whereas V_k is just the maximum. So f is exponentially larger.

Let's redefine $f_k(i)$ in such a way that it'll be more convenient for use today.

$f_k(i) = P(x - x_i, \pi_i = k$

$b_k(i) = P(x_{i+1}, x_n, \pi_i = k)$

2 HIGHER ORDER HMM

How do we model "memory" larger than one time point?

Claim: a second order HMM with K states is equivalent to a first order HMM with K^2 states.

Intuitively, look at the current state and the previous state (there are K^2 possible pairs of two states).

Instead of having a transition probability that looks at two states, you keep in memory the previous and current state.

For two states, state H and state T:

- There is an $\alpha_{HT}(prev = H)$
- For four states, state HH, state HT, state TH, state TT:
- Regarding the square, there is no path from state HH to state TT because there must be overlap between the current state in the FROM node and the previous state in the TO node.

Example: calculate the probability of achieving the following sequence of states: HHTTH

for two states: $\alpha_{HT}(p = H) \times \alpha_{TH}(p = H) \times \alpha_{TH}(p = T)$

for four states: $\alpha_{HHT} \times \alpha_{HTT} \times \alpha_{TTH}$

3 MODELING THE DURATION OF STATES

We can still talk about the hidden state signals.

Concept check:

What is the probability that we'll stay in X for this current state?

$\rightarrow P(l_x = 1) = 1 - p$

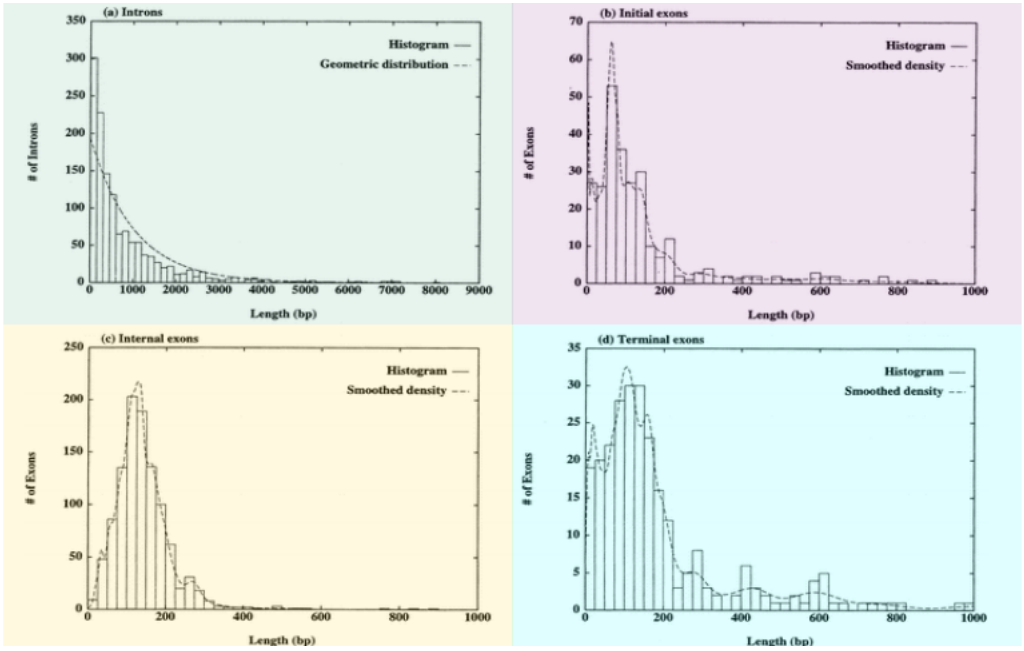
What is the probability that we'll stay in X for two steps, this and the next?

$\rightarrow P(l_x = 2) = p(1 - p)$

The generalization is $\sum_i i \times p^{i-1}(2 - p)$, and this leads to an $E(l_x) = \frac{1}{1-p}$.

Consider a casino player who only wants to stay in a certain state for a 100 iterations, so as to not appear conspicuous. We can train the HMM to allow for these sorts of requirements.

Example: exon lengths in genes

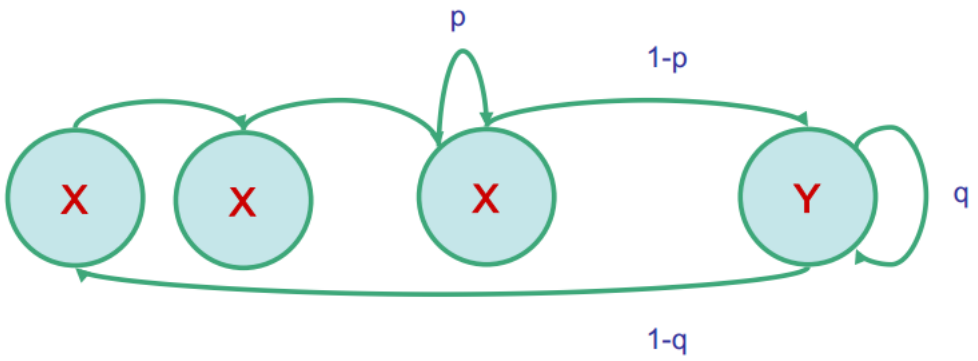


Exons are used to encode protein sequences.

Exons are biased to look like a beta distribution with some kind of mean.

3.1 SOLUTION 1: CHAIN SEVERAL STATES

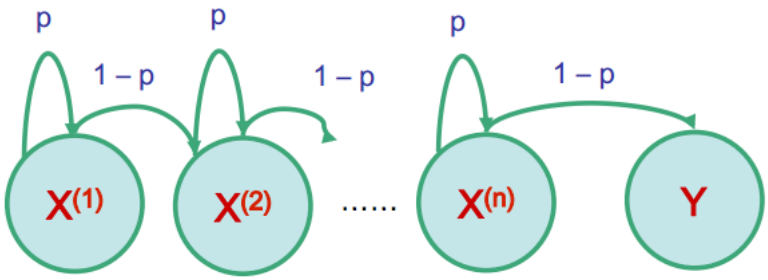
We will copy a state X a number of times. Among the first few copies, we'll have p probability transition.



This is too naive because it's still very inflexible. $I_X = C +$ geometric with mean $\frac{1}{1-p}$.

3.2 SOLUTION 2: NEGATIVE BINOMAIL DISTRIBUTION

Between each successive copy of X, we have transition probability of $1-p$, or probability of staying in x of probability p.



Question: what is the duration in X for precisely m steps?

To get to the last Y state, we would need to reach the last X state (X^n) in $m-1$turns:

$$\binom{m-1}{n-1}(1-p)^{n-1} \times (1-p)p^{m-n} = \binom{m-1}{n-1}(1-p)^{n-1} \times (1-p)^n p^{m-n}$$

The complexity of this algorithm is going to affect the running time linearly in n, as in $O(n)$

3.3 SOLUTION 3: DURATION MODELING

Upon entering a state, we can determine the precise duration of the stay in the state.

- We enter a state, and determine a duration $d < D_f$
- Then we emit d letters according to emission probabilities

Disadvantage: increase the time and complexity of Viterbi.

Recall that the original iteration:

$Vl(i) = max_k V_k(i = 1)a_{kl} \times e_l(x_i)$

New iteration:

$V_l(i) = max_k max_{d=1..Dl} V_k(i-d) \times P_l(d) \times a_{kl} \times \prod_{j-i-d+1...i} e_l(x_j)$

Running time of this is$O(nk^2d)$, which can get very expensive, so duration modeling is not used very often in practice. But it's useful to know that we have access to this kind of flexibility.

4 LEARNING, TWO SCENARIOS

1. Estimation when the "right answer" is known

   - genomic region $x = x_1, ..., x_{1,000,000}$, where we have good experimental annotations of the CpG islands
   - we observe a casino player rolling dice 10k times, and we know that he changes dice between fair and loaded

   Question: how would you determine how fair the die is?
   At any given point, I could ask "what is my confidence that I'm at a particular state at any point?"

2. Estimation when the "right answer" is unknown

   - the porcupine genome: we don't know how frequent the CpG islands are, nor the composition
   - casino player rolling dice 10k times, but we don't see when he changes dice between fair and loaded

   We should update the parameters θ of the model to maximize $P(x|\theta)$.

4.1 WHEN THE STATES ARE KNOWN

Given $x = x_1...x_n$ for which the true$\pi = \pi_1, ..., \pi_N$ is known, define

A_kl = num times $k \to l$ transition occurs in π
$E_k(b)$ = num times state k in π emits b in x

We can show that the MLE parametersθ (to maximize $P(x|\theta)$ are:

$a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}}$ and $e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}$

Imagine there are only 100 draws, some were loaded and some were air. It might be that the loaded state never emitted a 1. We use **pseudocounts** to avoid overfitting to the data.

Add r_{kl} and $r_k(b)$, which represent our prior belief to $A_{kl}$and $E_k(b)$, respectively.

Larger pseudocounts represent a strong prior belief.

4.2 WHEN THE STATES ARE HIDDEN

We are given $x = x_1...x_n$, but we don't know the underlying states. That is, we don't know the true $A_{kl}, E_k(b)$. We want to estimate the HMM parameters: the transition and emission probabilities for the HMM.

Idea is to estimate our "best guess" on what A_{kl},$E_k(b)$ are, based on prior knowledge. Alternatively, we start with random values. We update our parameters with our best guess, and then use our updated parameters to make an updated guess. We repeat this until we see $P(x|\theta)$ converge.

5 THE BAUM-WELCH ALGORITHM

Initialization: pick the best guess for model parameters (or arbitrary)

Iteration:

- forward
- backward
- calculate $A_{kl}, E_k(b)$ given the current θ
- calculate new model parameters θ_{NEW} and then a new $a_{kl}, e_k(b)$
- calculate new log likelihood $P(x|\theta_{NEW})$

This is guaranteed to be higher by expectation-maximization. Iterate until $P(x|\theta)$ does not change much.

Time complexity can be quite high because it is the number of iterations$\times O(K^2N)$, which is much higher than that of Viterbi.

Additionally, you'll need many random resets since Baum-Welch is not guaranteed to find the global optimal solution, but only the local optimal solution.

6 ALTERNATIVE TO BAUM-WELCH: VITERBI TRAINING

Initialization: start with an arbitrary set of parameters θ.

Iteration:

- Perform Viterbi, to find $\pi*$
- Calculate $A_{kl}, E_k(b)$ according to $\pi*$ +pseudocounts
- Calculate the new parameters until convergence

Unlike Baum-Welch, we use both θ and $\pi*$ to estimate the number of transitions and emissions $A_{kl}, E_k(b)$, which we then use to find the new set of parameters.

Viterbi training is not usually recommended in general because the performance is worst than Baum-Welch in general. However, if you're planning to use your model for Viterbi, it may be faster.