CS262 Lecture Notes: Hidden Markov Models

Sarah S

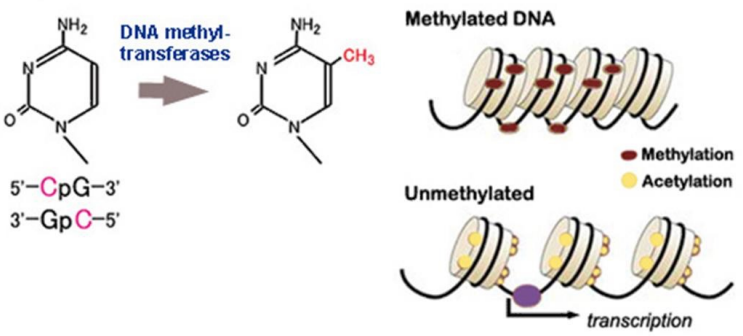
January 21 2016

1 Summary

Last lecture introduced hidden Markov models, and began to discuss some of the algorithms that can be used with HMMs to learn about sequences. In this lecture, we dive more deeply into the capabilities of HMMs, focusing mostly on their use in evaluation. We discuss higher order HMMs, and end with a brief discussion of HMM learning, to be expanded on in the next lecture.

2 CpG Islands

The main example used to illustrate the concepts of HMMs is the idea of "CpG islands." The notation 'CpG' indicates two adjacent nucleotides, a C and a G (these are next to each other, not paired across the strand — the p stands for the phosphate inbetween the cytosine and guanine). CpG islands are interesting because the distribution of adjacent Cs and Gs is different from other nucleotide pairs — they occur much less often than you'd expect in the overall genome ("CG supression").
However, there are places in the genome where they occur much more often. These are CpG islands, and they're interesting because they tend to be methylated (an extra methyl is added to a cytosine base). Methylation closes up the structure of the strand, making it too close together for the transcription enzymes to get to — so the section can't be transcribed. When this happens near a promoter, and entire gene can be permanently silenced. Over 60% of promoters occur in CpG islands.



[1]

Research has linked methylation of CpG groups near the promoters of tumor supression genes to human cancer [2]. So there is a lot of value in being able to find and identify CpG islands in the genome. And we can do that with hidden Markov models.

3 A Review of Decoding

Decoding is one of the three main uses of HMMs. You know the model and the sequence. You are maximizing for the likeliest path to produce a known sequence.
The other two are:
Evaluation: you know the model and the sequence, and are looking for the particular probability of some substring being produced by the model.
Learning: you know the sequence, but the model is unspecified, so you need to figure out the parameters that maximize the probability of getting that sequence.

Formally:

1. Decoding

GIVEN a HMM M, and a sequence x,
FIND the sequence π of states that maximizes P[x, π | M]

2. Evaluation

GIVEN a HMM M, and a sequence x,
FIND Prob[x | M]

3. Learning

GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,

FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]

Consider the question of decoding in the context of CpG islands. Decoding can tell you which areas are islands, and which are not, by determining what path is most likely to have led to the sequence at hand. (For detailed formulas and math, see the notes from last lecture)

4 Evaluation

This determines the likelihood a sequence is generated by a particular model. Or, given the sequence and the model, what is the most likely state at some position? If you remember the casino example (a casino is switching in a loaded die that rolls sixes half the time), evaluation asks "is a particular region loaded or fair?" You pick a particular area, then compare against all possible states to find out what the greatest probability is. (N.B. remember to account for the probability cost of switching between states at the borders of the selected area.) Specifically, Evaluation can compute:
P(x) – Probability of x given the model
$P(x_i \dots x_j)$ – Probability of a substring of x given the model
$P(\pi_i = k|x)$ – "Posterior" probability that the ith state is k, given x

5 Probability Review

$Pr(Y) = \sum_{x \in X} Pr(X, Y)$
Joint probability: $Pr(X, Y) = Pr(X|Y) * Pr(Y)$
Bayes Rule: $P(X|Y) = (P(Y|X)P(X))/P(Y)$

6 The Forward Algorithm

So how do you calculate P(x) given the model? With a "forward probability." The forward probability is the probability that some position x_i is what it is, given everything that came before. So, sum over all possible ways of generating x: $P(x) = \sum_\pi P(x, \pi) =_\pi P(x|\pi)P(\pi)$

Seems reasonable, but this creates an exponential number of paths! Luckily, we don't have to deal with all of them. Really all we care about is generating the first i characters of x, and ending up in state k. So we can simplify to: $f_k(i) = P(x_1 \dots x_i, \pi_i = k)$.

This leads to $f_k(i) = e_k(x_i) \sum_l f_l(i–1)a_{lk}$

Derivation:

$f_k(i) = P(x_1 ... x_i, \pi_i = k)$

use probability rules to obtain:

$= \sum_{\pi_1 \ldots \pi_{i-1}} P(x_1 ... x_{i-1}, \pi_1 .... \pi_{i-1}, \pi_i = k) e_k(x_i)$

$e_k(x_i)$ is the emission probability, i.e. the probability that state k will emit the value x_i

$= \sum_l \sum_{\pi_1 \ldots \pi_{i-2}} P(x_1 ... x_{i-1}, \pi_1 .... \pi_{i-2}, \pi_{i-1} = l) a_{lk} e_k(x_i)$

a_{lk} is the transition probability, that between states π_{i-1} and π_i we will transition from state l to state k

Simplify:
$= \sum_l P(x_1 ... x_{i-1}, \pi_{i-1} = l) a_{lk} e_k(x_i)$

And now using the original definition:
$= e_k(x_i) \sum_l f_l(i-1) a_{lk}$

Now that we know how to get a forward probability, how do we figure out the maximum probability? Dynamic programming!
Initialization: $f_0(0) = 1$ and $f_k(0) = 0$, for all k ¿ 0
Iteration: $f_k(i) = e_k(x_i) \sum_l f_l(i–1) a_l k$
Termination: $P(x) = \sum_k f_k(N)$

7 Backwards Algorithm

Now, what if we want some $P(\pi_i = k|x)$, the probability distribution that the ith position is k, given x? We need a helper to figure out the probability at our point of interest. We already have the forward probability — now we need the backwards probability.
Fun fact! P(x) can be defined with the backwards probability as well as the forward —- they are identical in their structure of computational steps and recursion.

7.1 Derivation of backwards probability

Define the backward probability: Starting from ith state = k, generate rest of x
$b_k(i) = P(x_i + 1 \ldots x_N | \pi_i = k)$

Expand with probability rules:
$= \sum + \pi_{i+1} \ldots \pi_N P(x_{i+1}, x_{i+2}, ... x_N, \pi_{i+1} ... \pi_N | \pi_i = k)$

Extract the state of π_{i+1}:
$= \sum_l \sum \pi_i ... \pi_N P(x_{i+1} \ldots x_N, \pi_{i+1} = l, \pi_{x+2}, \ldots, \pi_N | \pi_i = k)$

e is the emission probability; a is the transition probability:
$= \sum_l e_l(x_{i+1}) a_{kl} \sum \pi_i ... \pi_N P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N | \pi_{i+1} = l)$

Final result:
$= \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1)$

And again, dynamic programming saves the day, as we find the best probability by computing $b_k(i)$ for all k, i

Initialization: $b_k(N) = 1$, for all k

Iteration: $b_k(i) = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1)$

Termination: $P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$

7.2 Computational Complexity for forward and backward

Not great. $O(K^2 N)$ time, and O(KN) space.

There's also a danger of underflows, since all this multiplying decimals leads to really small numbers. Remember that when using Viterbi, you can use the sum of logs to avoid underflow? Well, when computing Forward or Backward probabilities, you can rescale at each few positions by multiplying by a constant. That keeps the numbers big enough for the computer to handle.

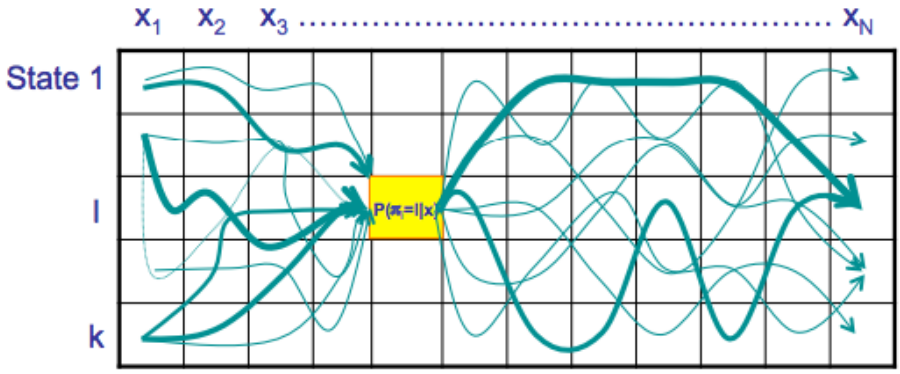
8 Posterior Decoding:

Posterior probability asks: what is the most likely state at position i of sequence x? Or, what is the probability that the ith state is k, given x?

Since we've shown how to compute $f_k(i)$ and $b_k(i)$, we can calculate $P(\pi_i = k|x) = (f_k(i) b_k(i))/P(x)$

Let's introduce a bit of new notation:
$\hat{\pi_i} = argmax_k P(\pi_i = k|x)$

Posterior decoding produces a curve of likelihood of state for each position:



But some of these paths may be invalid, made up of states that cannot occur in that sequence. The probability of such a path will be zero. How does this happen? You might end up with the most likely state at x_i not having a transition to get there from x_{i-1}!

Step back for a moment: what are the differences between Viterbi and posterior decoding? In Viterbi you are finding the entire path; in posterior you are finding one state.

Another way to see it: the probability of entering a state is the probability of all the paths that go through that state added up.

9 Comparison of what we've seen

VITERBI	FORWARD	BACKWARD
Initialization:	**Initialization:**	**Initialization:**
$V_0(0) = 1$	$f_0(0) = 1$	$b_k(N) = 1$, for all k
$V_k(0) = 0$, for all $k > 0$	$f_k(0) = 0$, for all $k > 0$	
Iteration:	**Iteration:**	**Iteration:**
$V_l(i) = e_l(x_i)\ \max_k V_k(i\text{-}1)\ a_{kl}$	$f_l(i) = e_l(x_i)\ \Sigma_k f_k(i\text{-}1)\ a_{kl}$	$b_l(i) = \Sigma_k e_l(x_i\text{+}1)\ a_{kl}\ b_k(i\text{+}1)$
Termination:	**Termination:**	**Termination:**
$P(x, \pi^*) = \max_k V_k(N)$	$P(x) = \Sigma_k f_k(N)$	$P(x) = \Sigma_k a_{0k}\ e_k(x_1)\ b_k(1)$

Note that it only requires small tweaks to adapt one algorithm to the other.

10 Variants of HMMs

10.1 Higher order HMMs

The models we've been considering have a "memory" of one position. But sometimes we might be interested in information about more than one time point.

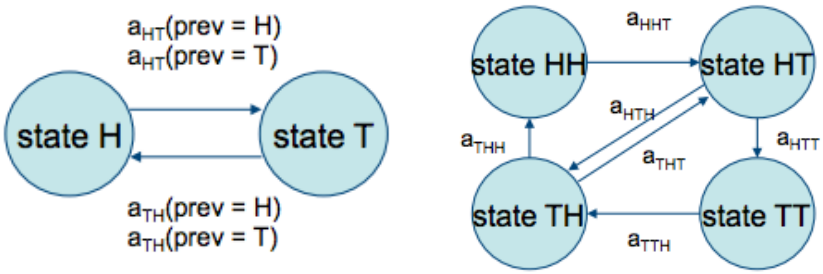
In math:

$$P(\pi_{i+1} = l | \pi_i = k) a_{kl}$$

versus

$$P(\pi_{i+1} = l | \pi_i = k, \pi_{i-1} = j) a_{jkl}$$

Take a look at these example models for flipping a coin, one is a single state model, the other a double state model.



In a double state model, some of the transitions can't happen — but that depends on how you choose to define transitions. For example, if we think of coin flips, heads as H and tails as T, HT to HT isn't possible if the second letter of the first double state is the first letter of the second state, because T is not H. But HT to TH makes sense, because the T overlaps.

What happens if we try Viterbi on a double state model?

The dimensions of the matrix increase: a k state, n-length parse leads to k^2 states, and a $k^2 n$ computation. This shows why you should try and avoid too much past dependence: if you have to model every possible state in the world, it takes too much time and space!

10.2 Modeling the Duration of States

Let's go back to our CpG example. At this point, we can ask WHERE are the CpG islands, can we also ask HOW LONG are the CpG islands?

That answer can tell us about accumulation of mutations over generations, or susceptibility to mutation in an individual.

To do that, we need the probability you will stay in or switch out of a state.

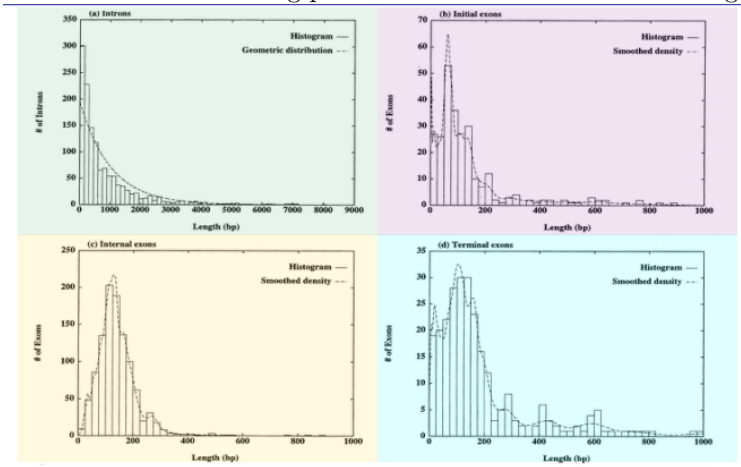
The canonical example is, of course, flipping a coin. The question: How many times do I have to flip coin before seeing a tails? (i.e. how long do I stay in a heads-state before transitioning to a tails-state?)

P(num of heads before first tail)
$= P(ktimes)$
$= p^{k-1} * (1-p)$
(little p is probability of flipping heads)

Expectation[k] = 1/(1-p)
Note that if p is large, 1-p is small, so E[k] is large.

This results in a geometric distribution.

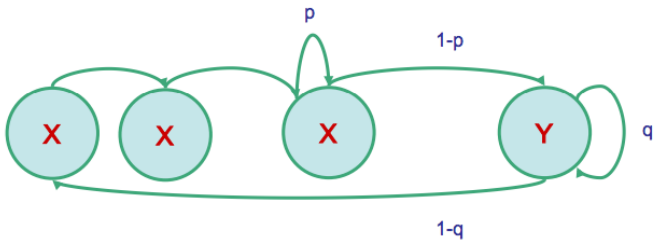
But there's an interesting problem here. Take a look at these graphs:



These graphs show the exon lengths in genes. Only the upper left graph (which actually shows introns) has a nice geometric distribution. In the rest, there is a peak, and we can't just ignore the smaller values to the left. We need a model that can account for those exons so they don't get lost. A geometric sequence will decrease right from the start, so it doesn't match what we see in real life.

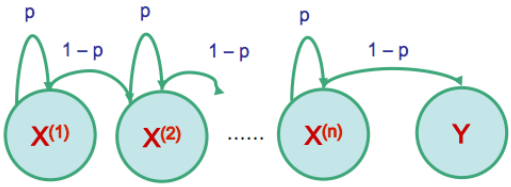
So what can we do?

One option is to chain several states together.



In this case, we end up with $l_x = C+$ geometric with mean 1/(1-p).
This means the distribution is still very inflexible, though it will catch a different set of possible graphs.

Another option is to use a negative binomial distribution. It turns out that this is a very good model for gene distribution.

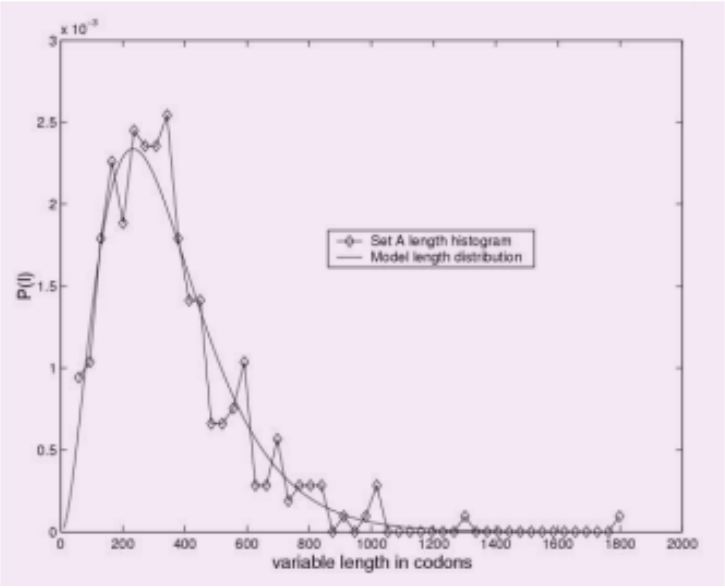


Duration in X: m turns, where

- During first m – 1 turns, exactly n – 1 arrows to next state are followed
- During m^{th} turn, an arrow to next state is followed

$$P(l_X = m) = \begin{bmatrix} m-1 \\ n-1 \end{bmatrix} (1-p)^{n-1+1} p^{(m-1)-(n-1)} = \begin{bmatrix} m-1 \\ n-1 \end{bmatrix} (1-p)^n p^{m-n}$$

We can see the result here, in a graph of prokaryotic gene length. Take a look at EasyGene, a prokaryotic gene-finder to try it yourself. (http://www.cbs.dtu.dk/services/EasyGene/)

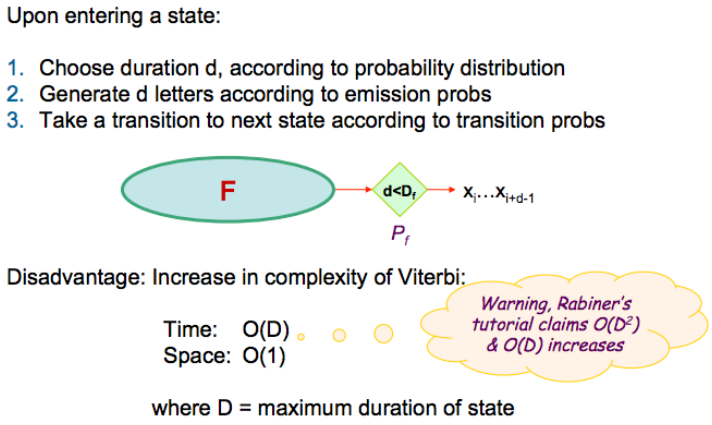


[3]

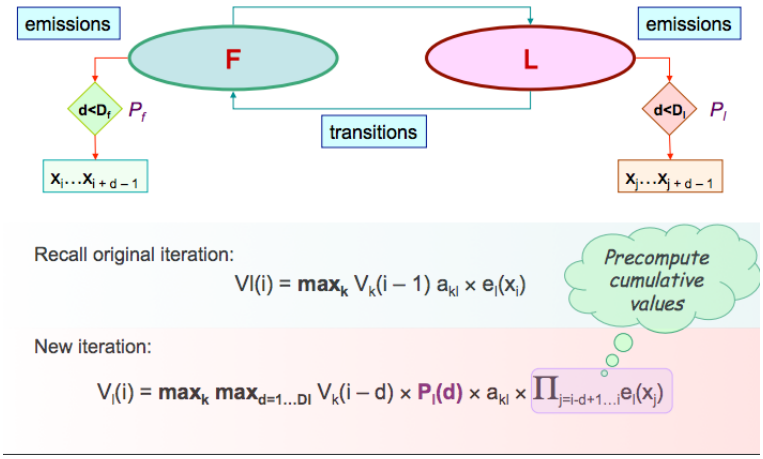
Our last potential solution is duration modeling.

Duration modeling seems like a very elegant solution. It is very flexible, and a simpler model than some of our other options.

Rather than add a bunch more states, simply choose a duration d according to a probability distribution, then generate letters d times while staying in the same state. The letters generated are controlled by the emission probabilities. Once the d letters have been generated, choose the next state to transition to, according to the transition probabilities.



However, the complexity gets much worse.



So choose your method carefully, with an eye to what distributions most accurately model the data.

11 A brief intro to Learning

Often we don't know transition probabilities for a model, and we have to figure them out ourselves.

Questions that can arise in that situation include:
What is the distribution of a fair vs loaded die?
What does a CpG island look like?

To answer these, we'll recursively update paramaters to maximize P, using expectation maximization.

The biggest problem is overfitting to small data. Imagine if in your data some output doesn't show up – it will have a probability of zero in the calculated transition probabilities, if we're simply counting how often we see an output in our data. But that might not be true!

Psuedocounts can compensate for overfitting. To guess a pseudo count, you can use your prior beliefs or knowledge about a situation, or simply use very small probabilities to stand in for ones you don't know anything about.

For a full treatment of this topic, see the next lecture.

---

[1] http://helicase.pbworks.com/w/page/17605615/DNA%20Methylation

[2] See this 1983 Nature article for interesting early work in this area: "Hypomethylation distinguishes genes of some human cancers from their normal counterparts" – http://www.ncbi.nlm.nih.gov/pubmed/6185846

[3] Larsen TS, Krogh A