Lecture 6: Hidden Markov Models Continued

Professor: Serafim Batzoglou
Lecturer: Victoria Popic
Class: Computational Genomics (CS262)
Scribe: John Louie

Due Date: Thursday January 22th 2015

1 Hidden Markov Model Example - Dishonest Casino

1.1 Conditions:

A casino has two die:

- Fair Dice: $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$
- Loaded Dice: $P(1) = P(2) = P(3) = P(4) = P(5) = 1/10$ and $P(6) = 1/2$

And the casino player switches between the fair and loaded die once every 20 rolls. Therefore, there are two states to the model: a Fair and Loaded state

The casino game is as follows:

1. You bet a dollar
2. You roll with a fair die
3. The casino rolls (with or without a loaded die)
4. Highest number wins two dollars

1.2 Definition of a HMM:

- **Alphabet**: $\Sigma = \{b_1, b_2, ..., b_n\}$
- **Set of States**: $Q = \{1, 2, ..., K\}$
- **Transition probabilities** between any two states: a_{ij} = transition probability from state i to state j, such that $i \in Q$ and $j \in Q$. $a_{i1} + ... + a_{iK} = 1$, for all states $i = 1, 2, .., K$. Note that the probabilities of transitioning from position i to any other state must sum to 1.
- **Start probabilities**: Specifically a_{0i}, such that $a_{01} + a_{02} + ... + a_{0K} = 1$, for all states $i = 1, 2, ...K$
- **Emission probabilities** within each state: $e_i(b) = P(x_i = b|\pi_i = K)$, which denotes the probability of observing $x \in \sum$, given that we begin from state $i \in Q.e_i(b_1) + ... + e_i(b_M) = 1$, for all states $i = 1, 2, ...k$.

1.3 HMMs are memory-less

Another important element to HMMs is the fact that they are memory-less, which means that the only thing that affects a future state, x_t, is the current state, π_t. More formally, we can see that

$$P(x_t = b|\text{`}whatever's\ happened\ so\ far') = P(x_t = b|\pi_1, \pi_2, ..., \pi_t, x_1, x_2, ..., x_{t-1}) = P(x_t = b|\pi_t)$$

So in a HMM, what is actually **hidden**?... the state we are currently in. All we can do is observe the sequence of transitions.

1.4 Definition of a Parse and the Parse Likelihood:

For a given sequence $x = x_1 x_2 ... x_n$, we have that a parse of x is a sequence of states $\pi = \pi_1, \pi_2, ..., \pi_n$

To determine the likelihood of our given parse (produced from a given HMM), we can use the following expression

$$P(x, \pi) =$$

$$P(x_1, ..., x_n, \pi_1, ..., \pi_n) =$$

$$P(x_n|\pi_n)P(\pi_n|\pi_{n-1})...P(x_2|\pi_2)P(\pi_2|\pi_1)P(x_1|\pi_1)P(\pi_1) =$$

$$a_{0\pi_1} a_{\pi_1 \pi_2} a_{\pi_{n-1}\pi_n} e_{\pi_1}(x_1)...e_{\pi_n}(x_n)$$

Note: A compact way to write the above expression We can essentially enumerate all parameters a_{ij} and $e_i(b)$ for the expression $a_{0\pi_1} a_{\pi_1 \pi_2} a_{\pi_{n-1}\pi_n} e_{\pi_1}(x_1)...e_{\pi_n}(x_n)$. Then we count in x and π the number of times each parameter $j = 1, 2, ..., n$ occurs:

$$F(j, x, \pi) = number\ of\ times\ parameter\ \theta_j\ occurs\ in\ (x, \pi)$$

Then, we have

$$P(x, \pi) = \Pi_{j=1,...,n} \theta_j^{F(j,x,\pi)} = exp(\sum_{j=1,...,n} log(\theta_j) * F(j, x, \pi))$$

1.5 Generation of a Sequence/Parse (given a HMM):

Given a HMM, we can obtain a sequence of length n as follows:

1. Start at state π_1, according to the probability $a_{0\pi_1}$
2. Emit symbol x_1 with the probability $e_{\pi_1}(x_1)$
3. Continue to state π_2, with the probability $a_{\pi_1 \pi_2}$
4. Repeat from step 2 until n symbols obtained

2 Problems with HMMs:

There are three main questions when working with HMM:

1. **Decoding**: GIVEN a HMM M and a sequence x, FIND $P(x|M)$ (The probability that sequence x was generated by the model M)
2. **Evaluation**: GIVEN a HMM M and a sequence x, FIND the sequence π of states that maximizes $P(x, \pi|M)$
3. **Learning**: GIVEN a HMM M, with unspecified transition/emission probabilities, and a sequence x, FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize $P(x|\theta)$

2.1 Decoding: Finding the most likely parse of a sequence

Given the sequence $x = x_1 x_2 ... x_n$, our objective is to find the most likely series of states $\pi^* = \pi_1, \pi_2, ...\pi_n$:

$$\pi^* = argmax_\pi P(\pi|x) = argmax_\pi \frac{P(\pi, x)}{P(x)} = argmax_\pi P(\pi, x) =$$

Two possible approaches: Naive Technique and a **Dynamic Programming**

- Naive Approach: use an exhaustive search; however, we'd have to consider k^n possibilities, for some constant k. This could take a very long time and quickly becomes an infeasible solution to maximizing π^*.
- Dynamic Programming Approach: this approach can be accomplished with the Viterbi Algorithm! (see below)

2.1.1 Viterbi Algorithm:

Let us consider the following inductive assumption:

$$V_k(i) = max_{\pi_1...\pi_{i-1}} P(x_1, ..., x_{i-1}, \pi_1, ..., \pi_{i-1}, x_i, \pi_i = k) =$$
$$Probability\ of\ most\ likely\ sequence\ of\ states\ ending\ at\ state\ \pi_i = k$$

Let us now consider the quantity $V_l(i+1)$. By definition,

$$\begin{aligned}
V_l(i+1) &= max_{\pi_1...\pi_i} P(x_1, ..., x_i, \pi_1, ..., \pi_i, x_{i+1}, \pi_{i+1} = l) \\
&= max_{\pi_1...\pi_i} P(x_{i+1}, \pi_{i+1} = l | x_1, ..., x_i, \pi_1, ..., \pi_i) P(x_1, ..., x_i, \pi_1, ..., \pi_i) \\
&= max_{\pi_1...\pi_i} P(x_{i+1}, \pi_{i+1} = l | \pi_i) P(x_1, ..., x_i, \pi_1, ..., \pi_i) \\
&= max_k [P(x_{i+1}, \pi_{i+1} = l | \pi_i = k)] max_{\pi_1...\pi_{i-1}} [P(x_1, ..., x_{i-1}, \pi_1, ..., \pi_{i-1}, x_i, \pi_i = k)] \\
&= max_k [P(x_{i+1}, \pi_{i+1} = l) P(\pi_{i+1} = l | \pi_i = k) V_k(i)] \\
&= e_l(x_{i+1}) max_k [a_{kl}]
\end{aligned}$$

The full definition of The Viterbi Algorithm is as follows:

For in put sequence $x = x_1...x_n$:

- **Initialization**: $V_0(0) = 1$ and $V_k(0) = 0$, for all $k > 0$ and where 0 is the imaginary first position.
- **Iteration**: $V_j(i) = e_j(xi) max_k a_{k,j} V_k(i-1)$ and $Ptr_j = argmax_k a_{k,j} V_k(i-1)$
- **Termination**: $P(x, \pi^*) = max_k V_k(N)$
- **Traceback**: $\pi_n^* = argmax_k V_k(N)$ and $\pi_{i-1}^* = Ptr_{\pi i}(i)$

Space-time Complexity Analysis:

- Space: $O(KN)$, because we are storing a $K * N$ sized matrix
- Time: $O(K^2N)$, because we need to do K work for each cell and $K * KN = K^2N$

Note: Practical detail: In practice, we are often dealing with small numbers when multiplying. Therefore, *underflow* may occur. To avoid this, we take the logarithm of all the values.

2.2 Evaluation: Finding the likelihood a sequence is generated by the model

In this particular problem, we are attempting to find:

- $P(x)$: Probability of x given the model.
- $P(x_i...x_j)$: Probability of a substring of x, given the model.
- $P(\pi_i = k|x)$: ?Posterior? probability that the ith state is k, given x. The process of posterior decoding will be explained in more depth with the following sections.

2.2.1 The Forward Algorithm:

Since we want to calculate $P(x)$ (the probability of getting x, given the HMM M), we can obtain $P(x)$ by summing over all possible ways of generating x:

$$P(x) = \sum_\pi P(x, \pi) = \sum_\pi P(x|\pi) P(\pi)$$

However, to avoid computing an exponential number of paths π, we want to instead define a *forward* probability (which essentially means to generate the i first characters of x and end up in state k):

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

We can further define $f_k(i)$ as a recursive formula as follows:

$$\begin{aligned}
f_k(i) &= P(x_1...x_i, \pi_i = k) \\
&= \textstyle\sum_{\pi_1...\pi_{i-1}} P(x_1...x_{i-1}, \pi_1, ..., \pi_{i-1}, \pi_i = k) e_k(x_i) \\
&= \textstyle\sum_l \sum_{\pi_1...\pi_{i-2}} P(x_1...x_{i-1}, \pi_1, ..., \pi_{i-2}, \pi_{i-1} = l) a_{lk} e_k(x_i) \\
&= \textstyle\sum_l P(x_1...x_{i-1}, \pi_{i-1} = l) a_{lk} e_k(x_i) \\
&= a_{lk} \textstyle\sum_l f_l(i-1) e_k(x_i)
\end{aligned}$$

We can also utilize a dynamic programming matrix to compute $f_k(i)$. The full definition of The Forward Algorithm is as follows:

- **Initialization**: $f_0(0) = 1$ and $f_k(0) = 0$, for all k ¿ 0
- **Iteration**: $f_k(i) = e_k(x_i) \sum_l f_l(i-1) a_{lk}$
- **Termination**: $P(x) = \sum_k f_k(N)$

2.2.2 The Backward Algorithm:

Since we want to compute $P(\pi_i = k|x)$, which essentially represents the probability distribution on the ith position, given the sequence x, we can start by expanding and simplifying the expression, as follows:

$$\begin{aligned} P(\pi_i = k, x) &= \\ &= P(x_1...x_i, \pi_i = k, x_{i+1}...x_N) \\ &= P(x_1...x_i, \pi_i = k) P(x_{i+1}...x_N | x_1...x_i, \pi_i = k) \\ &= P(x_1...x_i, \pi_i = k) P(x_{i+1}...x_N | \pi_i = k) \end{aligned}$$

Note that the first term in our result, $P(x_1...x_i, \pi_i = k)$, is actually the forward probability $f_k(i)$. In addition, we have that $P(x_{i+1}...x_N|\pi_i = k)$ is the backwards probability $b_k(i)$. Therefore, we can compute derive a recursive expression for $b_k(i)$ as we did for the forward probability above:

$$\begin{aligned} b_k(i) &= P(x_{i+1}, x_{i+2}, ...x_N, \pi_i = k) \\ &= \textstyle\sum_{\pi_{i+1}...\pi_N} P(x_{i+1}, x_{i+2}, ...x_N, \pi_{i+1}, ..., \pi_N | \pi_i = k) \\ &= \textstyle\sum_l \sum_{\pi_{i+1}...\pi_N} P(x_{i+1}, x_{i+2}, ...x_N, \pi_{i+1} = l, \pi_{i+2}, ..., \pi_N | \pi_i = k) \\ &= \textstyle\sum_l a_{lk} e_k(x_{i+1}) \sum_{\pi_{i+1}...\pi_N} P(x_{i+2}, ..., x_N, \pi_{i+2}, ..., \pi_N | \pi_{i+1} = l) \\ &= \textstyle\sum_l a_{lk} b_l(i+1) e_k(x_{i+1}) \end{aligned}$$

We can also utilize a dynamic programming matrix to compute $b_k(i)$. The full definition of The Backward Algorithm is as follows:

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

2.2.3 Computational Complexity for Both The Forward and Backward Algorithms:

Our analysis of the algorithms' complexity is very similar to that of the Viterbi Algorithm:

- Space: $O(KN)$, because we are storing a $K * N$ sized matrix
- Time: $O(K^2N)$, because we need to do K work for each cell and $K * KN = K^2N$

Note: Practical detail: In practice, we are often dealing with small numbers when multiplying. Therefore, *underflow* may occur. To avoid this, rescaling at periodic positions by multiplying by a constant.

2.2.4 Posterior Decoding:

In the posterior decoding, we are interested in computing $P(\pi_i = k|x)$ (the probability of being in state k at position i of the sequence x). Now that we have expressions for both the forward and backward probabilities, we can now calculate

$$P(\pi_i = k|x) = \frac{P(\pi_i = k, x)}{P(x)} = \frac{P(x_1...x_i, \pi_i = k) P(x_{i+1}...x_N|\pi_i = k)}{P(x)} = \frac{f_k(i) b_k(i)}{P(x)}$$

We can now define $\hat{\pi}_i = argmax_k P(\pi_i = k|x)$, which represents the most likely state at position i of sequence x. Thus with the posterior decoding we can compute the most likely state at each position. This in general is more helpful than the Viterbi path π^*. Note also that the posterior decoding at position i, $\hat{\pi}_i$, may not coincide with π_i^*. Furthermore, the posterior decoding may give an invalid sequence of states, unlike the Viterbi decoding, since there may be zero probability from transition from $\hat{\pi}_i$ to $\hat{\pi}_{i+1}$, for $i = 1, ..., n-1$. The Viterbi algorithm gives the most likely valid sequence of states that generated the sequence x, while the posterior decoding gives the most likely state at each position, and the resulting path may not be a valid sequence of states due to zero transition probability between states of two consecutive positions.