Hidden Markov Models

July 10, 2024

Before we study Hidden Markov Models (HMM) it is necessary to begin with an introduction to dynamic programming (DP).

1 An Intro to Dynamic Programming

An instructive example for the use of DP deals with finding the optimal alignment between two DNA strands. DNA consists of two stands intertwined like vines bound together by hydrogen bonds. These bonds, called bases, contain information about the organism. There are four types of bases; adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T).
When studying the evolutionary origin and relations of different species it is often helpful to compare the sequences of bases from two different DNA strands. Suppose we wanted to compare two sequences of DNA, x and y, noting how many times $x_i = y_i$, where x_i is the ith element of x. Let $x = \{\text{C A G A T G A G C A}\}$ and $y = \{\text{T C A A T G A A C A}\}$ then their ungapped alignment is shown in figure **??**.

```
C  A  G  A  T  G  A  G  C  A
      |  |  |  |        |  |
T  C  A  A  T  G  A  A  C  A
```

Figure 1: ungapped alignment

As can be seen in Figure **??** this alignment has 6 matches. If we allow gaps, denoted by '−', to be inserted into the sequences then we can create more matches. Figure **??** shows us gapped alignment.

```
−  C  A  G  A  T  G  A  G  C  A
   |  |     |  |  |  |        |  |
T  C  A  −  A  T  G  A  A  C  A
```

Figure 2: gapped alignment

Since adding gaps changes the original sequence we will denote the set of all gapped sequences of a sequence x by $\mathcal{A}_x$. We let $x \in \mathcal{A}_x$. Suppose we are

interested in finding the optimal gapped alignment of two sequences x and y using the scoring function S given by

$$S(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = y_i \\ -1 & \text{if } x_i \neq y_i \\ -2 & \text{if } x_i = - \text{ or } y_i = - \end{cases}$$

Then the optimal alignment is given by solving

$$\begin{aligned} \max_{(\hat{x},\hat{y}) \in \mathcal{A}_x \times \mathcal{A}_y} \quad & \left\{ \sum_{i}^{\text{len}(\hat{x})} S(\hat{x}_i, \hat{y}_i) \right\} \\ \text{s.t.} \quad & \text{len}(\hat{x}) = \text{len}(\hat{y}) \end{aligned}$$

where $\text{len}(x)$ denotes the number of elements in x. Note that we do not require $\text{len}(x) = \text{len}(y)$ but only $\text{len}(\hat{x}) = \text{len}(\hat{y})$. The alignment in Figure **??** is the solution to the system given by (**??**) with a score of 3.
One way to solve this system for any given x and y is to check the score of all possible gapped alignments by considering every $\hat{x}$ and $\hat{y}$ satisfying the constraints in (**??**). This method is obviously computationally inefficient. Dynamic programming allows us to eliminate suboptimal alignments by breaking solving subproblems of the original problem.

1.1 Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm is a DP that solves (**??**). Given two sequences x and y let $\text{len}(x) = n$ and $\text{len}(y) = m$. We will let F be an $(n+1) \times (m+1)$ matrix where F(i,j) represents the score of the optimal alignment for $x_1, x_2, \ldots, x_{i-1}$ and $y_1, y_2, \ldots, y_{j-1}$. Since we have boundary issues with this definition of F we let $F(i, 1) = -2(i-1)$ for $i = 1, \ldots, n+1$ which gives us the score of aligning the $x_1, \ldots, x_{i-1}$ with all gaps. Likewise, $F(1, j) = -2(j-1)$ for $j = 2, \ldots, m+1$. Since we have the first column and row of F we can find the rest of the entries by solving the recursive equation given by

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + S(x_i, y_i) \\ F(i-1, j) - 2 \\ F(i, j-1) - 2 \end{cases} .$$

This recursive equation for $F(i, j)$ helps us see if it is better to align x_{i-1} with y_{i-1}, x_{i-1} with a gap, or y_{i-1} with a gap. Once the entire matrix has been generated we trace back from the entry $F(n+1, m+1)$ to see which decisions were made to find the max of each entry. If $F(n, m) + S(x_{n+1}, y_{m+1})$ was the max then we know that in the optimal alignment x_n was aligned with y_m and we next look at $F(n, m)$. If $F(n, m+1) - 2$ was the max then x_n was aligned with a gap and we look at $F(n, m+1)$ next. If $F(m+1, n) - 2$ was the max then y_m was aligned with a gap and we look at $F(m+1, n)$. By continuing this process and sweeping back to the beginning of the alignment we can find the optimal alignment.

In summary the Needleman-Wunsch algorithm is outlined in the following steps.

1. Initialization: Define $(n+1) \times (m+1)$ matrix F.
   (a) $F(i,1) = -2(i-1)$ for $i = 1, \ldots, n+1$
   (b) $F(1,j) = -2(j-1)$ for $j = 2, \ldots, m+1$
2. Recursion: Let $F(i,j) = \max\{F(i-1,j-1) + S(x_i, y_i), F(i-1,j) - 2, F(i,j-1) - 2\}$ for $i = 2, \ldots, n+1$ and $j = 2, \ldots, m+1$.
3. Trace back: Record where you came from starting at $F(n+1, m+1)$.

1.1.1 Needleman-Wunsch Example

For example, let $x = \{\text{T A C A T G C}\}$ and $y = \{\text{T T C A G C}\}$, e.g.,

$$\begin{array}{ccccccc} \text{T} & \text{A} & \text{C} & \text{A} & \text{T} & \text{G} & \text{C} \\ | & & | & | & & | & | \\ \text{T} & \text{T} & \text{C} & \text{A} & - & \text{G} & \text{C} \end{array}$$

We will use the scoring function S given by

$$S(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = y_i \\ -1 & \text{if } x_i \neq y_i \\ -2 & \text{if } x_i = - \text{ or } y_i = - \end{cases}$$

and the Needleman-Wunsch algorithm to find the optimal alignment.
First We will define the following DP matrix:

		t	a	c	a	t	g	c
t								
t								
c								
a								
g								
c								

Then completing the margins using the Needleman-Wunsch scoring:

		t	a	c	a	t	g	c
	0	-2	-4	-6	-8	-10	-12	-14
t	-2							
t	-4							
c	-6							
a	-8							
g	-10							
c	-12							

And finishing the entire matrix:

		t	a	c	a	t	g	c
	0	-2	-4	-6	-8	-10	-12	-14
t	-2	**1**	-1	-3	-5	-7	-9	-11
t	-4	-1	**0**	-2	-4	-4	-6	-8
c	-6	-3	-2	**1**	-1	-3	-3	-5
a	-8	-5	-2	-1	**2**	**0**	-2	-2
g	-10	-7	-4	-3	0	1	**1**	-1
c	-12	-7	-6	-3	-2	1	0	**2**

The optimal alignment is then found by **back-tracing** the source of the optimal subscores (i.e. outlined in black), i.e.,

$$\begin{array}{ccccccc} \mathrm{T} & \mathrm{A} & \mathrm{C} & \mathrm{A} & \mathrm{T} & \mathrm{G} & \mathrm{C} \\ | & & | & | & & | & | \\ \mathrm{T} & \mathrm{T} & \mathrm{C} & \mathrm{A} & - & \mathrm{G} & \mathrm{C} \end{array}$$

1.2 Smith-Waterman Algorithm

So far we have assumed we know which sequences we want to align. Suppose now that we want to find the optimal alignment between subsequences of x and y. The optimal alignment of subsequences of two sequences is called the best local alignment. This algorithm is nearly the same as Needleman-Wunsch with just a couple differences. The first difference is that

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1,j-1) + S(x_i, y_i) \\ F(i-1,j) - 2 \\ F(i,j-1) - 2 \end{cases} .$$

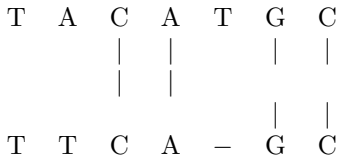
The other difference is that the trace back step starts at the highest value of F. From there you trace back in the DP matrix until the first 0 is found. This will yield the best local alignment of x and y.

1.2.1 Smith-Waterman Example

Using the previous matrix:

		t	a	c	a	t	g	c
	0	0	0	0	0	0	0	0
t	0	1	0	0	0	1	0	0
t	0	1	0	0	0	1	0	0
c	0	0	0	**1**	0	0	0	1
a	0	0	1	0	**2**	0	0	0
g	0	0	0	0	0	1	**1**	0
c	0	0	0	1	0	0	0	**2**

Now **back-tracing from the maximum** (i.e. outlined in black), yields three possible local alignments



2 Hidden Markov Models

A HMM is a statistical model where the system is assumed to be a Markov process but the state is not observed. However, one does observe some kind of outcome that comes from being in a state. We call this outcome an emission. We will let $y_1, y_2, \ldots, y_n$ be a sequence of states which we cannot observe. We let $x_1, x_2, \ldots, x_n$ be the sequence of emissions observed corresponding to $y_1, \ldots, y_n$. The object is to find out which state we are in for each emission.

2.1 Example: Occasionally Dishonest Casino

In this example, we suppose that a casino usually uses a fair die but occasionally uses a loaded die. As is expected, the rolls of a fair die have the usual probabilities. The probabilities for the rolls of the loaded die are $P(x_i = j \mid y_i = L) = 1/10$ for $j = 1, \ldots, 5$ and $P(x_i = 6 \mid y_i = L) = 1/2$. The emissions for this example are a sequence of die rolls. The hidden process is the die being used. Though we do not observe which die is being used we assume we know the transition probabilities of the states. The transition probabilities are $P(y_{i+1} = F \mid y_i = F) = .95$, $P(y_{i+1} = L \mid y_i = F) = .05$, $P(y_{i+1} = F \mid y_i = L) = .1$, and $P(y_{i+1} = L \mid y_i = L) = .9$. These transition probabilities can be represented in the Transition matrix

$$T = \begin{pmatrix} .95 & .05 \\ .1 & .9 \end{pmatrix}.$$

We will assume with probability 1 that we know the casino is using the fair die before the first roll.

2.2 Viterbi Algorithm

The Viterbi algorithm provides one way to solve this problem. It is a dynamic programming algorithm that considers all possible routes to a state but only keeps the most likely one. We will present the general algorithm for a finite number of states s. The assumptions are that we know the transition probabilities of the state as well as the emission probabilities. Let n be the number of emissions. Let $T_{k,i} = P(y_{i+1} = i \mid y_i = k)$ and $e_i(x_j) = P(x_j \mid y_j = i)$. Let V be a $s \times (n+1)$ matrix which will be our dynamic programming matrix. The algorithm is as follows:

1. Initialization: Define V as a $s \times (n+1)$ matrix.

(a) $V(1,0) = 1$;

(b) $V(i,0) = 0$ for $i = 2, \ldots, s$

2. Recursion: for $j = 1, \ldots, n$ and $i = 1, \ldots, s$

   (a) $V(i,j) = e_i(x_j) \max_k \{V(k,,j-1)T_{k,i}\}$

   (b) Each time before iterating from j to $j+1$ you must scale the jth column of V so that the column sums to 1.

   (c) Trace back: record $\text{argmax}_k \{V(k,j-1)T_{k,i}\}$

3. Termination: Start at $\text{argmax}_k \{V(k,n)\}$ and sweep back.

There are a few things to note about this algorithm. In step 1 we assumed that we know with probability 1 what the initial state is. However, this algorithm allows any initial distribution. In step 3 we find which state we most likely terminate in. From there we sweep back by using the information from the trace back step to decide which state we most likely came from.

2.3 Forward-Backward Algorithm

The Viterbi algorithm has advantages and disadvantages. The advantage is that it is fast relative to the forward-backward algorithm. The disadvantage is that it doesn't yield actual probabilities but just tells you which state you are most likely in. Before we derive the forward-backward algorithm we review some useful concepts from probability theory.
Recall the definition of conditional probability says that for two events A and B

$$P(A \cap B) = P(A \mid B)P(B).$$

This can be extended to the intersection of three events A, B, and C by

$$P(A \cap B \cap C) = P(A)P(B \mid A)P(C \mid B \cap A).$$

Recall also that for a discrete joint density function $f(x,y)$ the marginal density function is defined as

$$f(x) = \sum_y f(x,y)dy.$$

Now we will derive the forward-backward algorithm. Let $\bar{x}$ be a vector containing all the emissions. We wish to find $P(y_i = k \mid \bar{x})$. Using the definition of conditional probability we have

$$\begin{aligned} P(y_i = k \mid \bar{x}) &= \frac{P(y_i = k, \bar{x})}{P(\bar{x})} \\ &= \frac{P(y_i = k, x_1, \ldots, x_i)P(x_{i+1}, \ldots, x_n \mid y_i = k, x_1, \ldots, x_i)}{P(\bar{x})} \end{aligned}$$

Note that $P(\bar{x})$ is a normalizing constant that you can use to scale the probabilities so that $\sum_k P(y_i = k \mid \bar{x}) = 1$. We let $f_k(i) = P(y_i - k, x_1, \ldots, x_i)$ and

$b_k(i) = P(x_{i+1}, \ldots, x_n \mid y_i = k, x_1, \ldots, x_i)$. We call $f_k(i)$ the forward equation and $b_k(i)$ the backward equation. This gives us

$$P(y_i = k \mid \bar{x}) = \frac{f_k(i) b_k(i)}{P(\bar{x})}.$$

2.3.1 Forward Algorithm

We will now derive an expression for $f_k(i)$. Using the definition of a marginal density function and conditional probability for multiple events we have that

$$\begin{aligned} f_k(i) &= P(y_i = k, x_1, \ldots, x_i) \\ &= \sum_{j=1}^{s} P(y_i = k, x_1, \ldots, x_i, y_{i-1} = j) \\ &= \sum_{j=1}^{s} P(x_i \mid y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) P(y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) \end{aligned}$$

Since we are assuming the underlying process $y_1, \ldots, y_n$ is a markov process we can say that

$$P(x_i \mid y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) = P(x_i \mid y_i = k).$$

This gives us

$$\begin{aligned} f_k(i) &= \sum_{j=1}^{s} P(x_i \mid y_i = k) P(y_i = k, x_1, \ldots, x_{i-1}, y_{i-1} = j) \\ &= \sum_{j=1}^{s} e_k(x_i) P(y_i = k \mid x_1, \ldots, x_{i-1}, y_{i-1} = j) P(x_1, \ldots, x_{i-1}, y_{i-1} = j) \\ &= \sum_{j=1}^{s} e_k(x_i) P(y_i = k \mid, y_{i-1} = j) P(x_1, \ldots, x_{i-1}, y_{i-1} = j) \\ &= e_k(x_i) \sum_{j=1}^{s} T_{j,k} f_j(i-1) \end{aligned}$$

Now that we have a recursive equation for $f_k(i)$ we have an algorithm for the forward equation.

1. Initialization: Define a $s \times (n+1)$ matrix F where $F_{i,j} = f_i(j)$.
   (a) $f_1(1) = 1$
   (b) $f_k(1) = 0$ for $k = 2, \ldots, s$
2. Recursion: $\hat{f}_k(j) = e_k(x_i) \sum_{i=1}^{s} T_{i,k} f_i(j-1)$ for $k = 1, \ldots, s$ and $j = 1, \ldots, n$
   (a) Let $s(j) = \sum_{i=1}^{s} \hat{f}_i(j)$, which is our scaling factor. Before moving to the next column of the matrix by iterating j let $f_k(j) = \hat{f}_k(j)/s(j)$ for $k = 1, \ldots, s$.

2.3.2 Backward Algorithm

Now we derive a recursive equation for the backward equation $b_k(i)$. The derivation is very similar to the forward equation.

$$\begin{aligned}
b_k(i) &= P(x_{i+1}, \ldots, x_n \mid y_i = k) \\
&= \sum_{j=1}^{s} P(x_{i+1}, \ldots, x_n, y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} P(x_{i+1} \mid x_{i+2}, \ldots, x_n, y_{i+1} = j, y_i = k) P(x_{i+2}, \ldots, x_n, y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} P(x_{i+1} \mid y_{i+1} = j) P(x_{i+2}, \ldots, x_n, y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} e_j(x_{i+1}) P(x_{i+2}, \ldots, x_n \mid y_{i+1} = j, y_i = k) P(y_{i+1} = j \mid y_i = k) \\
&= \sum_{j=1}^{s} e_j(x_{i+1}) b_j(i+1) T_{k,j}
\end{aligned}$$

This yields the following algorithm for the backward equation.

1. Initialization: Define $s \times n$ matrix B with elements $B_{i,j} = b_i(j)$
   (a) $b_k(n) = T_{k,1}$ for $k = 1, \ldots, s$
2. Recursion: Let $\hat{b}_k(j) = \sum_{i=1}^{s} e_i(x_{j+1}) b_i(j+1) T_{k,j}$ for $j = n-1, \ldots, 1$ and $k = 1, \ldots, s$
   (a) Let $s(j) = \sum_{k=1}^{s} \hat{b}_k(j)$ and before each time you iterate i make sure to scale it by letting $b_k(j) = \hat{b}_k(j)/s(j)$.

2.4 Forward-Backward Algorithm for Transitions

Suppose that we are trying to estimate the states of a HMM but we don't know the matrix T or the emission probabilities $e_k(x_j)$ or both. This alternative forward-backward algorithm along with the Baum-Welch algorithm, which is presented later, is useful for estimating these probabilities. We start with an initial guess for your transition and emission probabilities. After several iterations of the forward-backward algorithm the Baum-Welch algorithm will help us converge to the true probabilities. We let

$$a_{m,k}(j) = P(y_j = k, y_{j-1} = m \mid \bar{x}) = \frac{f_{m,k}(j) b_{m,k}(j)}{P(\bar{x})}$$

for $j = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$.

1. Initialization: Let F be an $s^2 \times n$ matrix where $F_{i,j} = f_{m,k}(j)$. The reason for having s^2 rows is that at each time j we must consider all possible state transitions. For example, a HMM with 2 states we may wish to let $F_{1,j} = f_{1,1}(j)$, $F_{2,j} = f_{1,2}(j)$, $F_{3,j} = f_{2,1}(j)$, and $F_{4,j} = f_{2,2}(j)$.
   (a) $f_{1,1}(1) = 1$ and $f_{m,k}(1) = 0$ for $m, k > 1$.
2. Recursion: for $i = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$ set
$$f_{m,k}(j) = e_k(x_j) T_{m,k} \sum_{i=1}^{s} f_{i,m}(j-1)$$
   (a) Remember to scale.

It is left to the reader to derive this algorithm as well as the backward algorithm in the exercises. You will need to show that $b_{m,k}(j) = b_k(j)$ and that the backward equation is in fact the same as the backward equation for $P(y_i = k \mid \bar{x})$.

2.4.1 Baum-Welch (Expectation-Maximation) Algorithm

This algorithm uses the forward-backward algorithm for transitions to estimate T and $e_k(x_j)$.

1. Initialization: Choose or guess initial values for your transition probabilities.
2. E-step: Apply a forward-backward algorithm for transitions to estimate $a_{m,k}(j)$ for $j = 2, \ldots, n$; $m = 1, \ldots, s$ and $k = 1, \ldots, s$. Note that the sequence state probabilities, denoted $p_k(j)$, can be obtained from
$$\hat{p}_k(j) = P(y_i = k \mid \bar{x}) = \sum_{m=1}^{s} P(y_i = k, y_{i-1} = m \mid \bar{x}) = \sum_{m=1}^{s} \hat{a}_{m,k}(j)$$
   where $\hat{a}_{m,k}(j)$ and $\hat{p}_k(j)$ represent estimates of $a_{m,k}(j)$ and $p_k(j)$ respectively. These estimates are based on your guesses for T and $e_k(x_j)$.
3. M-step: Estimate the emission and transition probabilities.
   (a) Loaded die emissions: Letting $I(x_i = t) = 1$ if emission $x_i = t$ and 0 otherwise, then
$$\hat{e}(t) = \frac{\sum_{i=1}^{n} \hat{p}_k(i) I(x_i = t)}{\sum_{i=1}^{n} \hat{p}_k(i)}$$
   for $t = 1, \ldots, 6$.
   (b) Transitions:
$$\hat{T}_{m,k} = \frac{\sum_{i=2}^{n} \hat{a}_{m,k}(i)}{n-1}$$
4. Repeat steps 2-3 until convergence.

3 Exercises

Problem 1 *Download http://statistics.byu.edu/johnson/DNAsequences.fa. This file contiains two long DNA sequences, one from the C. elegans pha4 gene and the other is from the C. Briggsae version of the gene (both species are nematode worms). Note that both sequences are very long (5000 bases). Use Needleman-Wunsch algorithm to align sequences int he file. Print the DP matrix to a file, and display your aligned result in a text file containing three lines in the following format:*

```
-  C  A  G  A  T  G  A  G  C  A
      |  |     |  |  |  |     |  |
T  C  A  -  A  T  G  A  A  C  A
```

where '-' indicates a gap, '|' indicates that the sequences are the same for the base and leave a blank space on the second line for the bases that are aligned but are not the same. Turn in the dynamic programming matrix file and the sequence alignment file for this problem

Problem 2 *Using the same DNA sequences as in the previous problem, align the sequences using the Smith-Waterman local alignment algorithm. Turn in the DP matrix and the sequence alignment as in the previous problem. What differences do you observe between the two alignments?*

Problem 3 *Download http://statistics.byu.edu/johnson/dierolls.txt. This file contains two lines of data from the 'Occasionally Dishonest Casino' example from class. The first line contains the die rolls. The second line is the (hidden) Markov state process (which die was used Fair/Loaded). Using only die rolls and the transition and emission probabilities form class, apply the Viterbi algorithm to estimate the most likely sequence of the hidden Markov Chain. Print the DP matrix to a file. Also print both your estimated Markov state sequence as well as the actual sequence to a file. Compare and comment on how the algorithm did in estimating the Markov Chain. Turn in you DP matrix file, your state sequence file, and your brief comments.*

Problem 4 *Using the dieroll data at http://statistics.byu.edu/johnson/dierolls.txt, apply a forward-backward algorithm to estimate the die state probabilities. Plot this result as well as your Viterbi result and compare the results. Comment on what you observe.*

Problem 5 *Download the die roll data set from http://statistics.byu.edu/johnson/dierolls2.txt. The data set comes from an 'Occasionally Dishonest Casino' example where the emission probabilities for the loaded die are not known. Assume you know the fair die probabilities and the transition probabilities are the same as given in the example above. Use an iterative Baum-Welch (EM) algorithm to estimate both the loaded die emission probabilities as well as the state probabilities (Hint: Start with a guess for the loaded die emissions, apply a forward-backward, estimate emissions, apply forward-backward, etc., until convergence). Write your forward, backward, probability and emission results to a file and turn this in.*

Problem 6 *Derive the recursion for the forward-backward algorithm for estimating*

$$P(y_i = m, y_{i-1} = k \mid \bar{x}).$$

You will be given the complete forward-backward algorithm in lab, so deriving the recursion is sufficient for this problem.

Problem 7 *Download the die roll data set from http://statistics.byu.edu/johnson/dierolls3.txt. The data set comes from an 'Occasionally Dishonest Casino' example where the emission probabilities for the loaded die and the transitions are not known. Assume you know the fair die probabilities. Use an iterative Baum-Welch (EM) algorithm to estimate the loaded die emission probabilities, the transition probabilities and the state probabilities. Write your forward, backward, probability, transition, and emission results to a file and turn this in.*