STAT 534
Lecture 12
Dynamic Programming: Viterbi and LCS
May 9, 2019

©2019 Marina Meilă
mmp@stat.washington.edu
Scribes: Kellie MacPhee, Jacqueline Zhou

1 Viterbi Algorithm

Recall the setup of the Viterbi Algorithm.

Given: model A, B, π and observations $O_{1:T}$
Want: $q^*_{1:T} := \underset{q_{1:T}}{\operatorname{argmax}} P[q_{1:T}|O_{1:T}, A, B, \pi]$

Note that $q^*_{1:T}$ is a function of T. We also define $p^* = P[q^*_{1:T}|O_{1:T}, A, B, \pi]$.

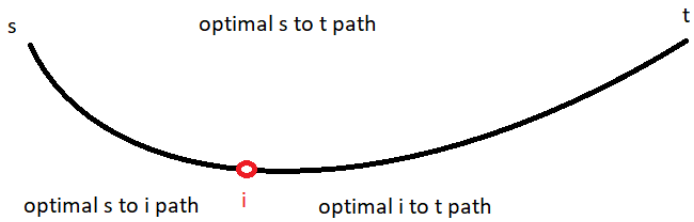
Compare this to the Forward-Backward Method, in which rather than a maximum we are computing a summation:

$$P[O_{1:T}|A, B, \pi] = \sum_{q_{1:T}} P[O_{1:T}, q_{1:T}|A, B, \pi] \tag{1}$$

From now on, we will suppress the model parameters A, B, π in the probabilities, but we assume that these are known.

1.1 Optimality Principle

A problem is said to fulfill the **optimality principle** or **maximum principle** if the sub-paths of the optimal path are themselves optimal paths for their sub-intervals. Consider the following: suppose we are searching for the optimal path from a source s to a target t, and we happen to know that the optimal path $P = (s, p_1, \ldots, p_N, t)$ goes through some p_i. Then the optimal path from s to p_i must be the sub-path of P given by $(s, p_1, \ldots, p_i)$ and similarly the optimal path from p_i to t must be the sub-path of P given by $(p_i, \ldots, p_N, t)$. (Proof: by contradiction.)



1.2 Algorithm

Define the following quantity, which corresponds to the most likely path from time 1 to t that ends at i:

$$\delta_t(i) := P[q^*_{1:t-1}, q_t = i, O_{1:t}] \tag{2}$$

Note that $p^* = \max_{i=1,\ldots,N} \delta_T(i)$ and we have the following recursion (for $t > 1$):

$$\delta_t(i) := \max_j \{\delta_{t-1}(j)a_{ji}\} b_{iO_t} \tag{3}$$

We also define the argmax of the above as the following pointer:

$$\psi_t(i) := \operatorname*{argmax}_j \{\delta_{t-1}(j)a_{ji}\} \tag{4}$$

The algorithm proceeds as follows:

Initialization

When $t = 1$, we have the base case:

$$\delta_1(i) = \pi_i b_{iO_1}, \quad 1 \leq i \leq N \tag{5}$$

$$\psi_1(i) = 0 \tag{6}$$

Recursion

The Viterbi Algorithm then consists of a forward iteration in which we compute each $\delta_t(i)$ and corresponding $\psi_t(i)$:

$$\delta_t(i) := \max_{1 \le j \le N} \{\delta_{t-1}(j) a_{ji}\} b_{iO_t}, \quad 2 \le t \le T, 1 \le i \le N \tag{7}$$

$$\psi_t(i) := \operatorname*{argmax}_{1 \le j \le N} \{\delta_{t-1}(j) a_{ji}\}, \quad 2 \le t \le T, 1 \le i \le N \tag{8}$$

Termination

Once we have computed all of the $\delta_T(i)$ we obtain $q^*_{1:T}$ as follows:

1. $q^*_T = \operatorname*{argmax}_{1 \le i \le N} \delta_T(i)$
2. For $t = T-1, \ldots, 1$, set $q_t = \psi_{t+1}(q^*_{t+1})$.

2 Longest Common Subsequence (LCS)

The longest common subsequence problem arises in computational biology, e.g. genetics. Here we have two sequences $X = X_{1:m}$ and $Y = Y_{1:n}$ and we want to know: how many letters must I delete from each sequence in order to make them coincide? The result is the longest common subsequence (LCS) $Z = Z_{1:k}$, where $k \le \min\{n, m\}$. As an example, consider $X = abcde$ and $Y = atcdql$. Then we would have $Z = acd$. There are also more complex notions of similarity between sequences, such as edit distance.

With LCS, the optimality principle is as follows. Define

$$l(i, j) = length(LCS(X_{1:i}, Y_{1:j})). \tag{9}$$

Considering the last letter of each sequence, we have two cases:

1. $X_m = Y_n$. In this case, $Z_k = X_m = Y_n$ and $Z_{1:k-1} = LCS(X_{1:m-1}, Y_{1:n-1})$. This yields the formula
$$l(m, n) = l(m-1, n-1) + 1 \tag{10}$$
2. $X_m \neq Y_n$. In this case, we either have $Z_{1:k} = LCS(X_{1:m-1}, Y_{1:n})$ or $Z_{1:k} = LCS(X_{1:m}, Y_{1:n-1})$; whichever of these two options gives a longer subsequence is our answer. This yields the formula
$$l(m, n) = \max\{l(m, n-1), l(m-1, n)\} \tag{11}$$

Writing X and Y along the top and left sides of a table, we create a table where entry (i, j) will correspond to the length of the longest subsequence of X_i and Y_j. Note that when $i = 0$ or $j = 0$, i.e. X_i or Y_j is an empty string, the

longest subsequence has length zero. We first fill in these initializations.

			1 S	2 T	3 A	4 T	5 S
		0	0	0	0	0	0
1	A	0					
2	S	0					
3	P	0					
4	E	0					
5	T	0					
6	S	0					

Now we fill out the rest of the table as follows. For each entry (i, j) in the table, if $X_i = Y_j$ we add 1 to entry $(i-1, j-1)$ (the number in the upper left diagonal space). Such locations are marked in red.

In all other locations, where $X_i \neq Y_j$, we simply take the max of the two cells immediately above and to the left. This results in the following:

			1 S	2 T	3 A	4 T	5 S
		0	0	0	0	0	0
1	A	0	0	0	1	1	1
2	S	0	1	1	1	1	2
3	P	0	1	1	1	1	2
4	E	0	1	1	1	1	2
5	T	0	1	2	2	2	2
6	S	0	1	2	2	2	3

Finally, we read off the bottom right entry as the length of the LCS. To obtain the actual LCS, we can also include pointers in each cell to where the value came from (options are the three cells above, to the left, and above and to the left).

			1 S	2 T	3 A	4 T	5 S
		0	0	0	0	0	0
1	A	0	$\leftarrow 0$	$\leftarrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$
2	S	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\nwarrow 2$
3	P	0	$\uparrow 1$	$\uparrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 2$
4	E	0	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 1$	$\uparrow 2$
5	T	0	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\nwarrow 2$	$\leftarrow 2$
6	S	0	$\nwarrow 1$	$\uparrow 2$	$\leftarrow 2$	$\leftarrow 2$	$\nwarrow 3$