CS378 Lecture Note: Viterbi Algorithm

The Viterbi algorithm is an algorithm for performing inference in Hidden Markov Models. A Hidden Markov Model is defined by

$$P(\mathbf{y}, \mathbf{x}) = P_S(y_1)P_E(x_1|y_1)\left[\prod_{i=2}^{n} P_T(y_i|y_{i-1})P_E(x_i|y_i)\right]P_T(\text{STOP}|y_n) \quad (1)$$

(although note that there are other ways to write this formula). We want to compute $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$, the most likely tag sequence given some input words $\mathbf{x}$. This is equivalent to computing $\arg\max_{\mathbf{y}} \frac{P(\mathbf{y},\mathbf{x})}{P(\mathbf{x})} = \arg\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x})$, due to Bayes' rule and the fact that the ensuing denominator does not depend on $\mathbf{y}$.

Let $|U|$ be the number of tags and let n be the length of a sentence under consideration. Assume that tags and words are both *indexed*, so that both words and tags can be represented as integers. The set of tags should contain a STOP token. When doing English tagging, correctly accounting for STOP at the end doesn't actually change the results that much since most sentences end in punctuation; however, it's necessary to make the model a correct probabilistic model (sums to 1 over all possible tagged sequences).

Model Viterbi requires the model parameters as input, which are typically estimated from training data.

1. Initial (start) probabilities: $\log P_S(y_1 = i)$. These can be stored in a vector $S[i]$.
2. Transition probabilities: $\log P_T(y = j|y_{\text{prev}} = i)$. These can be stored in a matrix $T[i, j]$.
3. Emission probabilities: $\log P_E(x = j|y = i)$. These can be stored in a matrix $E[i, j]$.

Algorithm The Viterbi algorithm is a dynamic programming approach to computing the highest scoring path. At its core is an $n \times |U|$ matrix v. $v[i, y]$ stores the log probability *so far* of the highest probability path ending in y at state i.

We're now ready to describe the algorithm (see next page):

Algorithm 1 Viterbi Algorithm

function VITERBI(x, S, T, E) $\triangleright$ x: sentence of length n, S: initial log probs, T: transition log probs, E: emission log probs
    Initialize v, a $n \times |U|$ matrix
    for $y = 1$ to $|U|$ **do** $\triangleright$ Handle the initial state
        $v[1, y] = S[y] + E[y, x_1]$
    end for
    for $i = 2$ to n **do**
        for $y = 1$ to $|U|$ **do**
            $v[i, y] = E[y, x_i] + \max_{y_{\text{prev}}} \big(T[y_{\text{prev}}, y] + v[i-1, y_{\text{prev}}]\big)$
        end for
    end for
    for $y = 1$ to $|U|$ **do** $\triangleright$ Handle the final state
        $v[n, y] = v[n, y] + T[y, \text{STOP}]$
    end for
    Best final state = $\arg\max_y v[n, y]$
    To reconstruct the answer: track argmaxes as well (where the max element values come from) and walk backwards to find the sequence
end function

For further reference, Wikipedia has a similar implementation,[1] but not in log space. That is, rather than adding up log probabilities, they multiply probabilities, which is riskier numerically.

Debugging Viterbi The main property of Viterbi that's helpful for debugging is that $\max_y v[n, y]$ should contain the log probability of the highest-scoring tag sequence for the sentence. For a short example (less than 4 words), you can explicitly compute this by just looping through all possible tag sequences with nested for loops and computing the score of each using the definition of an HMM. Assuming you write this scoring code correctly, you can double-check that your dynamic program returns the same answer as the explicit computation.

Variants The critical operations in the above algorithm are the + operations (to combine log probability values) and the max operations

1. $\max$, +: Viterbi algorithm in log space, as shown above (expects log-probability matrices as input)
2. $\max$, $\times$: Viterbi algorithm in real space (expects probability matrices as input)
3. +, $\times$: sum-product algorithm (also called the forward algorithm) in real space. Can be used to compute $P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y})$. Can be combined with a version of this algorithm called the backward algorithm to compute $P(y_i|\mathbf{x})$ for each position i in the sentence. This is outside the scope of this course, but is discussed more in the textbook.
4. log-sum, +: sum-product algorithm in log space. log-sum$(a, b) = \log(\exp(a) + \exp(b))$; that is, it takes two log-probabilities, turns them into probabilities, adds them together, and re-logs them. There is no other way to "add" log probabilities correctly.

[1] https://en.wikipedia.org/wiki/Viterbi_algorithm