The Viterbi Algorithm

Last time, we modeled the speech recognition problem. We let $Y_i$ be the phoneme at time $i$ and $X_i$ be the corresponding feature extracted from the sound picked up at the microphone.

![Figure 1: A system diagram for the speech recognition problem.](image)

The relation between the phonemes and the features can be represented by the following graphical model:

$$
\begin{align*}
X_1 & \quad X_2 & \cdots & \quad X_{n-1} & \quad X_n \\
Y_1 & \quad Y_2 & & & \quad Y_{n-1} & \quad Y_n
\end{align*}
$$

The goal in this lecture is to derive an efficient yet optimal algorithm to infer the sequence of phonemes from the sequence of features. The algorithm is called the Viterbi algorithm, used not only in speech recognition software but also many other applications.

How do we estimate $Y_1 \cdots Y_n$ given $X_1 = x_1, \cdots X_n = x_n$? We want to choose the sequence $y_1, \cdots, y_n$ to maximize

$$
P(Y_1 = y_1 \cdots Y_n = y_n | X_1 = x_1 \cdots X_n = x_n),
$$

which is the posterior probability. This is the the MAP rule. How do we efficiently solve this optimization problem? If we solve this by brute force (meaning by computing all possible a posteriori probabilities for all possible sequences of phonemes), what would be the complexity of the algorithm? Suppose that there are 40 different phonemes. We would have to compare $40^n$ possibilities. Even for our boring language with 2 phonemes, we are talking about $2^n$ possibilities. Can we solve this optimization more efficiently by exploiting the hidden Markov model?

Consider using Bayes’ rule:

$$
P(Y_1 = y_1, \cdots, Y_n = y_n | X_1 = x_1, \cdots, X_n = x_n) = \frac{f(x_1, x_2, \ldots, x_n | Y_1 = y_1, \cdots, Y_n = y_n)P(Y_1 = y_1, \cdots, Y_n = y_n)}{f(x_1, x_2, \ldots, x_n)}
$$

Notice that the denominator does not depend on the sequence of phonemes we choose. Therefore, we can just compare the numerator for different phoneme sequences. Now, using the fact that the sequence of
phonemes is modeled as a Markov chain and the fact that the noise $Z_i$’s are independent:

$$f(x_1, \ldots, x_n|Y_1 = y_1, \ldots, Y_n = y_n)P(Y_1 = y_1, \ldots, Y_n = y_n)$$

$$= f(x_1|Y_1 = y_1) \cdots f(x_n|Y_n = y_n)P(Y_1 = y_1)P(Y_2 = y_2|Y_1 = y_1) \cdots P(Y_n = y_n|Y_{n-1} = y_{n-1})$$

Now, conditional on $Y_i = y_i$, $X_i \sim N(\mu_{y_i}, \sigma^2)$. Hence,

$$f(x_i|Y_i = y_i) = \frac{1}{\sqrt{2\pi}\sigma^2}e^{(x_i-\mu_{y_i})^2/(2\sigma^2)}.$$ 

Substituting this into the previous equation, the MAP rule becomes this optimization problem:

$$\max_{y_1, \ldots, y_n} \frac{1}{\sqrt{2\pi}\sigma^2}e^{\sum_{i=1}^n(x_i-\mu_{y_i})^2/(2\sigma^2)}P(Y_1 = y_1)P(Y_2 = y_2|Y_1 = y_1) \cdots P(Y_n = y_n|Y_{n-1} = y_{n-1}).$$

It is convenient to take the logarithm of the above expression and then take the negative of the expression. The above maximization problem becomes the following minimization problem (after getting rid of the term that do not depend on the $y_i$’s):

$$\min_{y_1, \ldots, y_n} \sum_{i=1}^n (x_i - \mu_{y_i})^2/(2\sigma^2) - \ln P(Y_1 = y_1) - \sum_{i=2}^n \ln P(Y_i = y_i|Y_{i-1} = y_{i-1})$$

Let us now define:

$$d(y_1) := (x_1 - \mu_{y_1})^2/(2\sigma^2) - \ln P(Y_1 = y_1)$$

$$d_i(y_{i-1}, y_i) := (x_i - \mu_{y_i})^2/(2\sigma^2) - \ln P(Y_i = y_i|Y_{i-1} = y_{i-1})$$

Then the above minimization problem can be cast as:

$$\min_{y_1, \ldots, y_n} d(y_1) + \sum_{i=2}^n d_i(y_{i-1}, y_i)$$

The problem becomes a shortest path problem if we assign these to be the distances of traversing the edges of a graph. The following graph shows the model for $n = 4$. There is a source $S$ and a destination $D$, with $n$ stages, stage 1, 2, … , $n$ in between. There are two possible states at each stage of the graph, corresponding to the two possible values of the phoneme at that stage. The edges and the corresponding $d$ values are also indicated on the graph. The graph has an edge connecting each node at a stage to every node at the next stage. In our case, we are assuming there are 2 phonemes or states at every stage, represented by 2 nodes in each column. $n$ is the total number of phonemes in the sentence to recognize. There are 4 possible edges between any two stages. The distances on all the edges can be computed from the observations and the prior probabilities on the phonemes. The initial term $d(y_1)$ is represented by adding a starting node $S$ and connecting it to each node in stage 1. The above minimization problem for the MAP rule corresponds to finding the shortest path from $S$ to the final destination node. This shortest path problem can be solved very efficiently, without requiring exponential search.

Let

$$U_i(y) = \text{length of the shortest path to node } y \text{ at stage } i \text{ from the source } S$$

The length of the shortest path from source to destination is simply:

$$\min\{U_n(a), U_n(b)\}.$$
Figure 2: The graphical representation for computing the MAP solution. The optimal solution is the shortest path from the source node $S$ to the destination node $D$.

We compute $U_i(y)$’s successively, starting from $i = 1$ to $i = n$.

The shortest path to a node $y$ in stage 1 from node $S$ is simply the direct edge from $S$ to that node. Hence:

$$U_1(y) = d(y), \quad y = a, b.$$  

The shortest path to a node $y$ in stage 2 from node $S$ is either the shortest path to node $a$ in stage 1 plus an additional edge from that node to node $y$ in stage 2, or the shortest path to node $b$ in stage 1 plus an additional edge from that node to node $y$ in stage 2, whichever is shorter. Hence,

$$U_2(y) = \min\{U_1(a) + d_2(a, y), U_1(b) + d_2(b, y)\}, \quad y = a, b.$$  

More generally, suppose we have already computed $U_i(y)$ for stage $i$, $y = a, b$. Then the shortest path to state $y$ in stage $i+1$, $y = a, b$, can be computed as:

$$U_{i+1}(y) = \min\{U_i(a) + d_{i+1}(a, y), U_i(b) + d_{i+1}(b, y)\}, \quad y = a, b.$$  

When we have computed $U_n(a)$ and $U_n(b)$, then we are done with computing the overall shortest path from $S$ to $D$.

What is the total amount of computations needed? Computing the shortest path to each node in level $i+1$ requires comparing among 2 paths, one for each node in stage $i$. This is done for each of the 2 nodes in stage $i+1$. Therefore, it requires a total of 4 additions and 2 comparisons to get the shortest paths for all nodes in a new stage. For each stage, we do 6 computations. In total, we do $6 \times n$ computations. In general, for a language with $m$ possible phonemes, the complexity of the algorithm is linear in $n$ and quadratic in $m$: $m^2 \times n$.

This algorithm is called the Viterbi algorithm, invented in the context of decoding a type of code used in digital communications called convolutional codes. This algorithm existed before Viterbi. He adapted the Bellman-Ford algorithm to this particular setting where we want to infer a Hidden Markov chain sequence from a sequence of observations. The Bellman-Ford algorithm is itself a special case of the principle of dynamic programming.