EE 178/278A Probabilistic Systems Analysis

Spring 2014 Tse/Hussami Lecture 17

Speech Recognition Example Continued

Last time, we modeled the speech recognition problem. We let X_i be the phoneme at time i and Y_i be the corresponding feature extracted from the sound picked up at the microphone. In general, each Y_i is complicated and multi-dimensional. (For example, Y_i can be a vector of Fourier coefficients of the signal.) But to simplify matter we can think of Y_i for our purpose as a discrete random variable obtained perhaps from a discretization of the features.

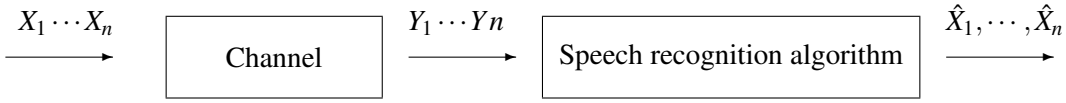


Figure 1: *A system diagram for the speech recognition problem.*

The relation between the input and output of the channel can be represented by the following graphical model:

$$\begin{array}{ccccc} Y_1 & Y_2 & & Y_{n-1} & Y_n \\ | & | & \cdots & | & | \\ X_1 - & X_2 & & X_{n-1} - & X_n \end{array},$$

where $X_1, \cdots, X_n$ is an underlying Markov chain. If we take out one node from the graph, it becomes disconnected. For example if we remove X_3, we now have two subgraphs. Given X_3, the variables X_1, X_2 become independent of the rest of the random variables $X_4 \cdots X_n$. Note that if we specify the initial distribution and the transition probabilities of a Markov chain, we can determine any joint or marginal probabilities of the X_i's. For example:

$$\mathbf{P}(X_2 = a) = \sum_{a'} \mathbf{P}(X_2 = a | X_1 = a') \mathbf{P}(X_1 = a').$$

How do we estimate $X_1 \cdots X_n$ given $Y_1 = b_1, \cdots Y_n = b_n$? We want to choose the sequence $a_1, \cdots, a_n$ to maximize

$$\mathbf{P}(X_1 = a_1 \cdots X_n = a_n | Y_1 = b_1 \cdots Y_n = b_n),$$

which is the posterior probability. We call this maximization the MAP rule, which stands for *Maximum a Posteriori Probability Rule*. How do we efficiently solve this optimization problem? If we solve this by brute force (meaning by computing all possible a posterior probabilities), what would be the complexity of the algorithm? Suppose that there are 40 different phonemes. We would have to compare 40^n possibilities. Can we solve this optimization more efficiently by exploiting the structure of the problem?

Consider using Baye's rule:

$$\begin{aligned}&\mathbf{P}(X_1 = a_1, \cdots, X_n = a_n | Y_1 = b_1, \cdots, Y_n = b_n) \\ &= \frac{\mathbf{P}(Y_1 = b_1, \cdots, Y_n = b_n | X_1 = a_1, \cdots, X_n = a_n)\mathbf{P}(X_1 = a_1, \cdots, X_n = a_n)}{\mathbf{P}(Y_1 = b_1, \cdots, Y_n = b_n)}\end{aligned}$$

Notice that the denominator does not depend on the input sequence we choose. Therefore, we can just compare the numerator for different input sequence. Now, :

$$\begin{aligned}&\mathbf{P}(Y_1 = b_1, \cdots, Y_n = b_n | X_1 = a_1, \cdots, X_n = a_n)\mathbf{P}(X_1 = a_1, \cdots, X_n = a_n) \\ &= \mathbf{P}(Y_1 = b_1 | X_1 = a_1) \cdots \mathbf{P}(Y_n = b_n | X_n = a_n)\mathbf{P}(X_1 = a_1)\mathbf{P}(X_2 = a_2 | X_1 = a_1) \cdots \mathbf{P}(X_n = a_n | X_{n-1} = a_{n-1}) \\ &= Q(b_1|a_1)Q(b_2|a_2) \cdots Q(b_n|a_n) \times \pi(a_1)P(a_2|a_1) \cdots P(a_n|a_{n-1})\end{aligned}$$

We want to optimize this over values of $a_1, a_2, \cdot, a_n$. Let us define

$$d(a_1) := -\log(\pi(a_1)Q(b_1|a_1)),$$

and

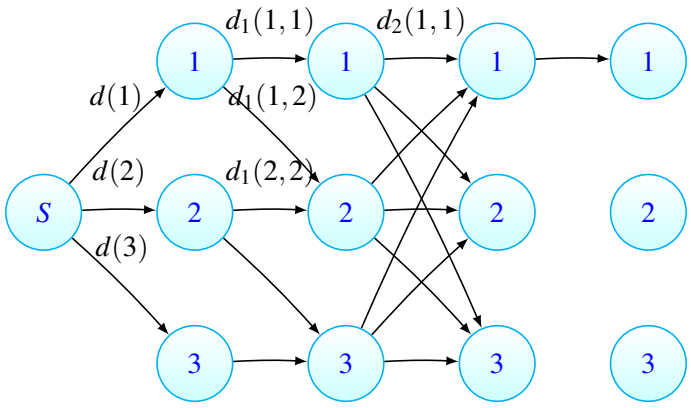
$$d_i(a_{i-1}, a_i) := -\log(P(a_i|a_{i-1})Q(b_i|a_i))$$

The b_i's are omitted from the argument list because they are constants in the optimization. We rewrite our problem in terms of these quantities:

$$\min_{a_1 \cdots a_n} d(a_1) + \sum_{i=2}^{n} d_i(a_{i-1}, a_i)$$

Suppose we have a sequence $a_1, \cdots, a_n$ that contains two consecutive states for which the transition probability is 0. Then the corresponding logarithm term will be $-\infty$. Therefore, this sequence will not be a possible answer to our minimization.

The problem becomes a shortest path problem if we assign d to be the cost of the edges. The following graph shows the model for $n = 4$, and 3 possible states at each stage. A subset of the edges and corresponding d values are indicated on the graph. The complete 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 40 phonemes or states at every stage, represented by 40 nodes in each column. n is the total number of phonemes in the sentence to recognize. There are 1600 possible edges between any two stages. We populate this matrix of costs, and find the shortest path. The initial term $d(a_1)$ is represented by adding a starting node S and connecting it to each node in stage 1. We want to find the shortest path from S to the last column. This shortest path problem can be solved very efficiently, without requiring exponential search.



Let us think about this recursively. Suppose we have found the shortest path from S to each of the nodes at stage i, meaning the ith column of nodes in the graph. We want to compute the shortest path to each node in the $i+1$th stage as well. Can we use what we have already computed? The answer is yes. The shortest path that goes from node 1 to a node k at level $i+1$ has to consist of the shortest path from node S to a node k in stage i, plus an extra edge from stage i to stage $i+1$ which gives the minimum cost up to stage $i+1$. The simple but key observation that justifies this statement is the following. Consider the shortest path $\mathscr{P}$ to a node k in stage $i+1$ passes through node m in stage i. Then the part of $\mathscr{P}$ from S to node m in stage i must itself be the shortest path from S to node m in stage i. Otherwise if there is another path $\mathscr{Q}$ which is shorter, by concatenating $\mathscr{Q}$ with the edge from node m at stage i to node k in stage $i+1$, we would have gotten a shorter path from S to node m in stage $i+1$, a contradiction.

Thus, computing the shortest path to each node in level $i+1$ requires comparing among 40 paths, one for each node in stage i. This is done for 40 nodes in stage $i+1$. Therefore, it requires 1600 comparisons to get the shortest paths for all nodes in a new stage. For each stage, we do 1600 comparisons. In total, we do $1600 \times n$ computations. In general, the complexity of the algorithm is linear in n: $|\mathbf{X}|^2 \times n$. The principle behind this is simple. It exploits the Hidden Markov model to achieve algorithmic simplification.

More explicitly, we will use the Viterbi algorithm (Bellman-Ford) to perform this optimization. Let

$$U_i(a) = \text{length of the shortest path to state } a \text{ at stage } i.$$

We initialize the algorithm by $U_1(a) = d(a)\ \forall a$, which is the cost due to the initial state. Now assume we have computed $U_{i-1}(a)$ for all a, $i \geq 2$. Then,

$$U_i(a) = \min_{a'}[U_{i-1}(a') + d_i(a', a)].$$

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*.

In the above, we assume the parameters $Q(b|a)$'s and $P(a'|a)$ are known. They can be estimated either by training with known sentences (supervised learning) or without (unsupervised learning). In the next lecture, we will discuss how to estimate these parameters in the unsupervised learning mode using the Expectation Maximization (EM) algorithm.