SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.007 Machine Learning, Fall 2015
Lecture Notes for Week 8

Hidden Markov Models (II)

Last update: Wednesday 11th November, 2015 20:48

1 Decoding

Suppose now that we have the HMM parameters (see above) and the problem is to predict the underlying tags $y_1, \ldots, y_n$ corresponding to an observed sequence of words $x_1, \ldots, x_n$. In other words, we wish to find:

$$\arg\max_{y_1,\ldots,y_n} p(y_0, y_1, \ldots, y_{n+1} | x_1, \ldots, x_n; \theta) \tag{1}$$

where we have:

$$p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta) = \prod_{i=1}^{n+1} a_{y_{i-1}, y_i} \prod_{i=1}^{n} b_{y_i}(x_i) \tag{2}$$

and $y_0 = \texttt{START}$ and $y_{n+1} = \texttt{STOP}$.

Discussions We have the joint distribution here, but we are interested in a conditional distribution above. What's the connection here?

$$\arg\max_{y_1,\ldots,y_n} p(y_0, y_1, \ldots, y_{n+1} | x_1, \ldots, x_n; \theta) = \arg\max_{y_1,\ldots,y_n} \frac{p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta)}{p(x_1, \ldots, x_n; \theta)} \tag{3}$$

Since the term $p(x_1, \ldots, x_n)$ is a constant that is independent of ys once the parameters are fixed, we can drop it when taking the $\arg\max$:

$$\arg\max_{y_1,\ldots,y_n} \frac{p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta)}{p(x_1, \ldots, x_n; \theta)} = \arg\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta) \tag{4}$$

This leads to:

$$\arg\max_{y_1,\ldots,y_n} p(y_0, y_1, \ldots, y_{n+1} | x_1, \ldots, x_n; \theta) = \arg\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta) \tag{5}$$

One possible solution for finding the most likely sequence of tags is to do brute force enumeration. Consider the example: {the, dog}, x = "the the the dog". The possible state sequences include:

START 1 1 1 2 STOP
START 1 1 2 2 STOP
START 1 2 2 2 STOP
⋮

But there are $|\mathcal{T}|^n$ possible sequences in total! Solving the tagging problem by enumerating the tag sequences will be prohibitively expensive.

Viterbi Algorithm

The HMM has a simple dependence structure (recall, tags form a Markov sequence, observations only depend on the underlying tag). We can exploit this structure in a dynamic programming algorithm.

Input: $\mathbf{x} = x_1, \ldots, x_n$ and model parameters θ.

Output: $\arg\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta)$.

Now, let's look at a truncated version of the joint probability, focusing on the first k tags for any $k \in \{1, \ldots, n\}$ In other words, we define

$$r(y_1, \ldots, y_k) = \prod_{i=1}^{k} a_{y_{i-1}, y_i} \prod_{i=1}^{k} b_{y_i}(x_i) \tag{6}$$

where $k \neq n+1$. Note that our notation $r(y_1, \ldots, y_k)$ suppresses any dependence on the observation sequence. This is because we view $x_1, \ldots, x_n$ as given (fixed). Note that, according to our definition,

$$p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}) = r(y_1, \ldots, y_n) \cdot a_{y_n, y_{n+1}} = r(y_1, \ldots, y_n) \cdot a_{y_n, \texttt{STOP}} \tag{7}$$

Let $S(k, v)$ be the set of tag sequences $y_1, \ldots, y_k$ such that $y_k = v$. In other words, $S(k, v)$ is a set of all sequences of length k whose last tag is v. The dynamic programming algorithm will calculate

$$\pi(k, v) = \max_{(y_1,\ldots,y_k) \in S(k,v)} r(y_1, \ldots, y_k) \tag{8}$$

recursively in the forward direction.

In other words, $\pi(k, v)$ can be thought as solving the maximization problem partially, over all the tags $y_1, \ldots, y_{k-1}$ with the constraint that we use tag v for y_k. If we have $\pi(k, v)$, then $\max_v \pi(k, v)$ evaluates $\max_{y_1,\ldots,y_k} r(y_1, \ldots, y_k)$. We leave v in the definition of $\pi(k, v)$ so that we can extend the maximization one step further as we unravel the model in the forward direction. More formally,

- Base case:

$$\pi(0, \texttt{START}) = 1 \qquad \text{(starting state, no observations)} \tag{9}$$

$$\pi(0, v) = 0, v \neq \texttt{START} \quad \text{(an actual state that has observations)} \tag{10}$$

- Moving forward recursively: for any $k \in \{1, \ldots, n\}$

$$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\} \tag{11}$$

In other words, when extending $\pi(k-1, u), u \in \mathcal{T}$, to $\pi(k, v), v \in \mathcal{T}$, we must transition from $y_{k-1} = u$ to $y_k = v$ (part $a_{u,v}$) and generate the corresponding observation x_k (part $b_v(x_k)$). Then we maximize over the previous tag $y_{k-1} = u$ so that $\pi(k, v)$ only depends on the value of y_k.

- Finally, we must transition from y_n to `STOP` so that

$$\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_0 = \texttt{START}, y_1, \ldots, y_n, y_{n+1} = \texttt{STOP}) = \max_{v \in \mathcal{T}} \{\pi(n, v) \cdot a_{v,\texttt{STOP}}\} \tag{12}$$

The whole calculation can be done in time $O(n|\mathcal{T}|^2)$, linear in length, quadratic in the number of tags.

Now, having values $\pi(k, v)$, how do we reconstruct the most likely sequence of tags which we denote as $y_1^*, \ldots, y_n^*$? We can do this via back-tracking. In other words, at the last step, $\pi(n, v)$ represents maximizations of all $y_1, \ldots, y_n$ such that $y_n = v$. What is the best value for this last tag v, *i.e.*, what is y_n^*? It is:

$$y_n^* = \arg\max_v \{\pi(n, v) \cdot a_{v,\texttt{STOP}}\} \tag{13}$$

Now we can fix y_n^* and work backwards. What is the best value y_{n-1}^* such that we end up with tag y_n^* in position n? It is simply

$$y_{n-1}^* = \arg\max_u \{\pi(n-1, u) \cdot a_{u,y_n^*}\} \tag{14}$$

and so on.

Discussions What is the space complexity of the above algorithm? Can we do something to avoid the back-tracking step? Is it possible to store the optimal transition information together with the $\pi(k, v)$? If we do so, what is the space complexity?

Exercise What if we would like to find the top-k ($k > 1$) most optimal sequences instead of finding the single most optimal tag sequence?

Learning Objectives

You need to know:

1. What is decoding for an HMM
2. How to perform decoding for an HMM using the Viterbi algorithm
3. What is the guarantee of the Viterbi algorithm and how to implement the Viterbi algorithm
4. What is the space and time complexity of the Viterbi algorithm