

HMM Tagging Exercise: Background

Jean Mark Gawron

April 14, 2004

1 HMM Tagging model

The tagging model being used:

$$P(w_{1,n}, t_{1,n}) = \prod_{i=1}^n P(w_i | t_i) * P(t_i | t_{i-1}) \quad (1)$$

Or, equivalently:

$$\log P(w_{1,n}, t_{1,n}) = \sum_{i=1}^n \log P(w_i | t_i) + \log P(t_i | t_{i-1}) \quad (2)$$

This is relevant since your implementation of this model will use log probabilities.

2 Smoothing

To do the unsmoothed model of any conditional probability, you have

$$P(A | B) = \frac{\text{Count}(A, B)}{\text{Count}(B)} \quad (3)$$

To do add .5 smoothing,

$$\hat{P}(A | B) = \frac{\text{Count}(A, B) + .5}{\text{Count}(B) + (T * .5)} \quad (4)$$

where T is the total number of **types** of events of which event type A is one instance.

For example, in the simple bigram case, consider the conditional probability $P(\textit{away} | \textit{walk})$, where the count of *walk away* is 30, the count *walk* is 200, and the vocabulary size is 1500. We have:

$$P(\textit{away} | \textit{walk}) = \frac{30}{200} = .15 \quad (5)$$

The total number of types events of which *away* is one instance is the vocabulary size 1500. After smoothing:

$$\hat{P}(\textit{away} \mid \textit{walk}) = \frac{30 + .5}{200 + (1500 * .5)} = .03 \quad (6)$$

Note it's crucial that the number of types we add in the denominator is the number of types of which **A** is an exemplar. So if we were smoothing conditional probabilities of car makes given car colors, we want to add the number of **car makes** to the denominator.

Making this concrete, assume there are 30 red VWs and 200 red cars:

$$P(\textit{VW} \mid \textit{red}) = \frac{30}{200} = .15 \quad (7)$$

And if 6 is the number of colors and 8 the number of makes, then after smoothing we have:

$$P(\textit{VW} \mid \textit{red}) = \frac{30 + .5}{200 + (8 * .5)} = .074 \quad (8)$$

Yes, the number of colors is irrelevant here.

3 Combining the probability model and Smoothing

An important part of this assignment is to correctly combine the model of Section 1 with the smoothing in Section 2. Thus the first thing you might want to do is sit down and write out an equation combining equations (2) and (4).

4 Charniak's Viterbi HMM example

Figure 1 shows a simple HMM due to Charniak with an alphabet "ab". To discuss this example, all we need is the concept of a transition probability. There is a probability assigned to the transition from one state to another on a particular *observation* (or sometimes we say *emission*).

The key property is that the sum of all the transitions probabilities from a given state is 1.

The task of the Viterbi algorithm is to find the optimal path through the HMM for a given piece of input. In this case optimal path means *highest probability*.

Figure 2 shows the application of the Viterbi algorithm to the HMM. The view of the computation is a computational tree. Time is vertical. That is, each step down through the diagram reveals one more input symbol.

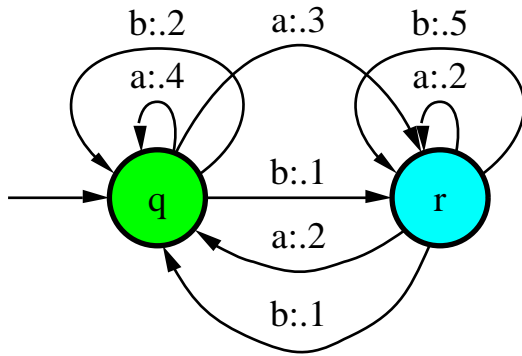


Figure 1: Charniak's simple example HMM

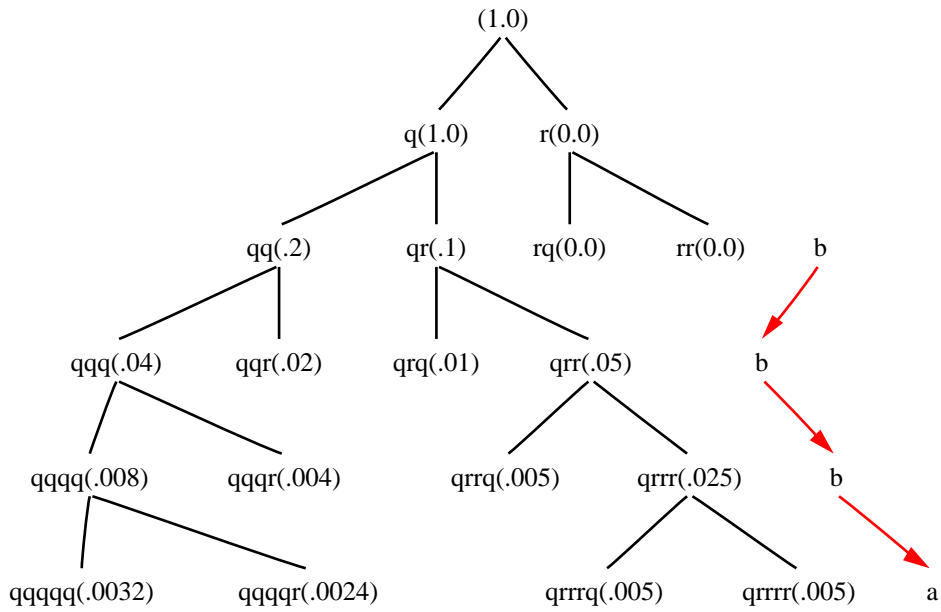


Figure 2: Computational Tree of Viterbi on Charniak's HMM

The HMM has two states that both accept a's and b's. From each state, for any input, there are two possible states one could go to next. Thus, if we were doing a naive search, computing the total cost of all paths covering the input, the number of active paths at each moment in time would double.

Instead, the number of active paths trees remains constant as we descend through the computation. Why? That's the trick of the algorithm.

5 The Viterbi HMM example

Figure 3 shows the simple HMM used in the Viterbi implementation in the code model *viterbi* (see code model list in the assignment).

This is a simple coin flipping HMM. The state H does not expect a tail. The state T does not expect a head. However, when in H, you can accept an “h” either by remaining in H or, with equal probability, by moving to T. When in T you can accept a “t” either by moving to H or by remaining in T.

Figure 4 shows the same HMM with zero-probability transitions removed. This makes the overall picture clearer. Only the start state 0 is flexible about what it accepts, But when it accepts an “h” it can do so either by moving into an h-accepting or t-accepting state. Thus, each time input is accepted, a “prediction” is made about what will come next.

The transition probabilities are computed using the following probability model:

$$\Pr(o_1, \dots, o_n, s_1, \dots, s_n) = \prod_{i=1}^n \Pr(o_i \mid s_{i-1}, s_i) * \Pr(s_i \mid s_{i-1}) \quad (9)$$

We call:

$$\Pr(s_i \mid s_{i-1}) \quad (10)$$

the *state transition probability*. These probabilities are kept in the array of arrays \$A in the implementation.

We call

$$\Pr(o_i \mid s_{i-1}, s_i) \quad (11)$$

the *observation transition probability* These probabilities are kept in the hash of arrays \$B in the implementation.

We call the product of these two probabilities the *transition probability*:

$$\Pr(o_i \mid s_{i-1}, s_i) * \Pr(s_i \mid s_{i-1}) \quad (12)$$

At each stage in the Viterbi algorithm this is the cost of moving from a previous state to the current state given the output at the current moment in time.

The transition probabilities of Figure 3 are computed as follows. (For each transition, the observation probability is given first and the transition probability second).

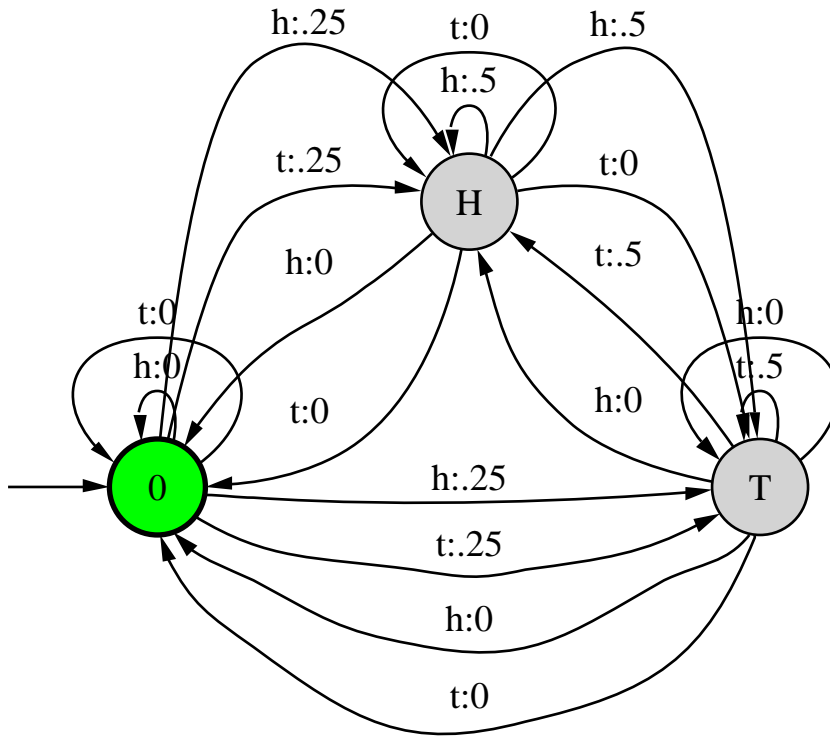


Figure 3: Rob Malouf' Viterbi Implementation Example

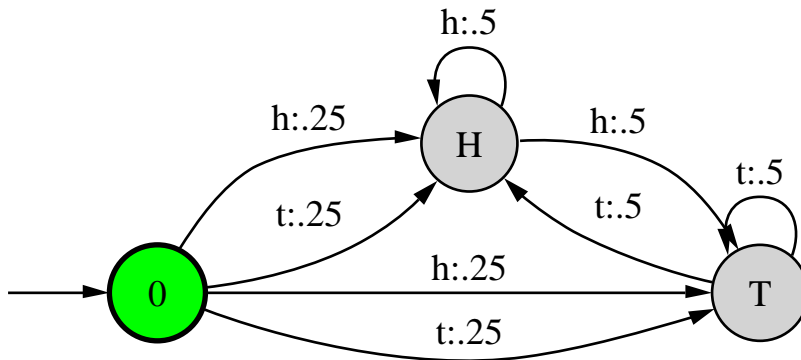


Figure 4: Rob Malouf' Viterbi Implementation Example

```

0 :
h
0=>0: 0 * 0 = 0
0=>H: 0.5 * 0.5 = 0.25
0=>T: 0.5 * 0.5 = 0.25
t
0=>0: 0 * 0 = 0
0=>H: 0.5 * 0.5 = 0.25
0=>T: 0.5 * 0.5 = 0.25
-----
0 (Total Probs) = 1

H :
h
H=>0: 0 * 0 = 0
H=>H: 1 * 0.5 = 0.5
H=>T: 1 * 0.5 = 0.5
t
H=>0: 0 * 0 = 0
H=>H: 0 * 0.5 = 0
H=>T: 0 * 0.5 = 0
-----
H (Total Probs) = 1
  
```

```

T :
h
  T=>0: 0 * 0 = 0
  T=>H: 0 * 0.5 = 0
  T=>T: 0 * 0.5 = 0
t
  T=>0: 0 * 0 = 0
  T=>H: 1 * 0.5 = 0.5
  T=>T: 1 * 0.5 = 0.5
-----
T (Total Probs) = 1

```

Here is the result of running the Viterbi implementation on this for the output “htttt”:

```

Output: htttt
Time State Path Prob
0: 0 1
1: T 0.25
2: T 0.125
3: T 0.0625
4: T 0.03125
5: H 0.015625

```

States: 0TTTTH

The HMM accepts an “h” by going to the T state, then remains there and accepts the final “t” by transitioning to the h-predicting state H. (The final state is always an arbitrary choice since we have not seen the “predicted” output yet.). The path probability after the first transition is .25, as required by Figure 4, and it is halved by each succeeding transition.

The application of Viterbi, using a computational tree diagram, is shown in Figure 5.

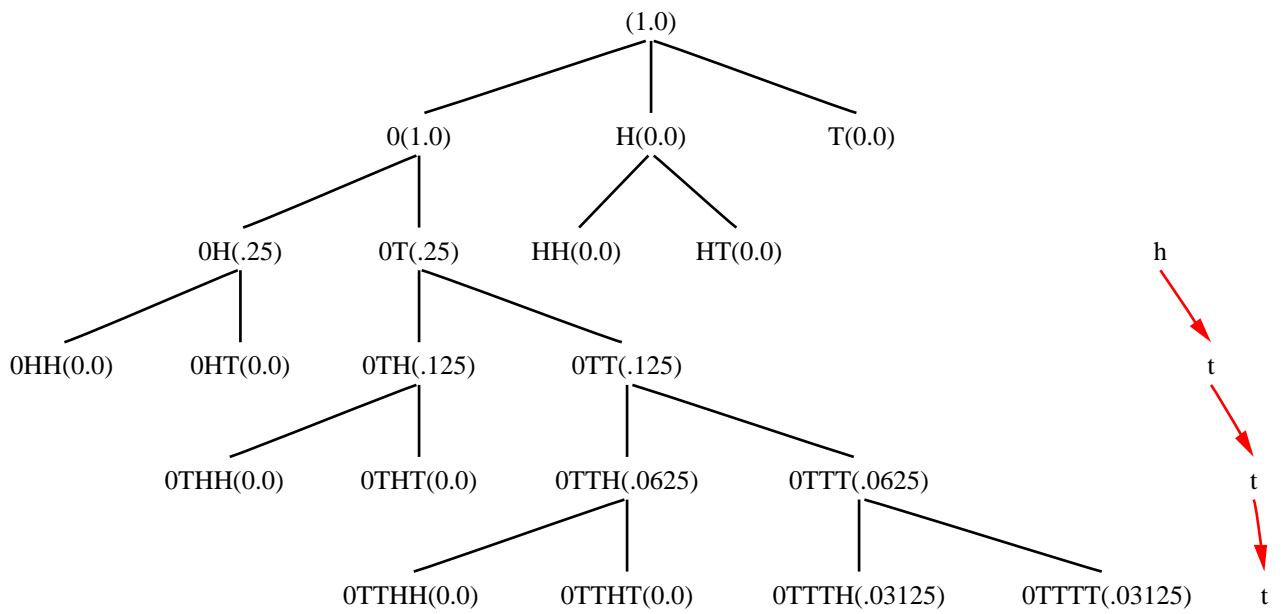


Figure 5: Rob Malouf' Viterbi Implementation Example