

HMMs and Tagging

Jean Mark Gawron

April 7, 2005

1 HMM Tagging model

The HMM tagging model is the following:

$$P(w_{1,n}, t_{0,n}) = \prod_{i=1}^n P(w_i | t_i) * P(t_i | t_{i-1}) \quad (1)$$

This means to find the joint probability of a word-tag sequence, we multiply together

$$P(w_i | t_i) * P(t_i | t_{i-1})$$

for all word-tag pairs w_i, t_i . Or, equivalently we can express this as a sum using log probabilities.

$$\log P(w_{1,n}, t_{0,n}) = \sum_{i=1}^n \log P(w_i | t_i) + \log P(t_i | t_{i-1}) \quad (2)$$

Why? Because:

$$a_1 \times a_2 \times \dots \times a_n = P \quad \text{iff} \quad \log a_1 + \log a_2 + \dots + \log a_n = \log P$$

So what does this mean? This means we take the probability of a sequence of word tag pairs to be approximated in a certain way. For each word get

$P(w_i | t_i)$ the probability of the current word given its tag
 $P(t_i | t_{i-1})$ the probability of this word's tag given the previous word's tag

Call the product of these two components the *current word factor*. Multiply all the current word factors together and you have the probability of the entire tagged sequence.

Compare this to what the chain rule would have us do:

$$\text{HMM: } P(w_{1,n}, t_{1,n+1}) = \prod_{i=1}^n P(w_i | t_i) * P(t_i | t_{i-1}) \quad (3)$$

$$\text{Chain: } = \prod_{i=1}^n P(w_i | w_{1,i-1}, t_{0,i}) * P(t_i | w_{1,i-1}, t_{0,i-1}) \quad (4)$$

So in effect we are making the following two approximations:

$$\begin{aligned} \text{(a) } P(w_i | w_{1,i-1}, t_{0,i}) &\cong P(w_i | t_{i-1}) \\ \text{(b) } P(t_i | w_{1,i-1}, t_{0,i-1}) &\cong P(t_i | t_{i-1}) \end{aligned}$$

(a) in particular looks strange, But hey, it might work....

Now why does THIS model lend itself to an HMM?

2 The Key property of an HMM

What is an HMM? Formally, it has the following ingredients:

1. a set of states: S
2. a set of final states: F
3. an initial state: $s_0 \in S$
4. an input/observation alphabet: Σ
5. a transition function A which takes any pair of states and returns a probability. The transition probabilities LEAVING any state add up to 1. Why? because our interpretation of these transition probabilities is the following:

$$A(s_i, s_{i+1}) = \Pr(s_{i+1} | s_i)$$

And it just a consequence of the definition of conditional probabilities that

$$\sum_{x \in S} \Pr(x | y) = 1$$

for any sample space S .

6. An observation function B.

$$B(s_i, o_i)$$

is the probability of observing ‘emission’ o_i (this is the way these guys talk) at state s_i . s_i is a state and o_i is a member of the input (observation) alphabet. The observation probabilities for any state add up to 1. Why?

$$B(s_i, o_i) = \Pr(o_i | s_i)$$

We will talk further about the interpretation of HMMs below. For now, let’s just contemplate some practicalities.

First, what are you supposed *do* with this? You can predict the probability of sequences of observations.

To get to THAT concept you first need the concept of the probability of a PATH through the HMM. This is just the product of all the transition and observation probabilities in the path involved. That is, for input symbol o_i , there is a state we’re in s_i and a state we go to s_{i+1} . And the factor for THAT piece of input is:

$$A(s_i, s_{i+1}) \times B(s_{i+1}, o_i)$$

So the total path probability is:

$$P(o_{1,n}, s_{1,n+1}) = \prod_{i=1}^n A(s_i, s_{i+1}) \times B(s_{i+1}, o_i)$$

Then the total probability for a sequence of observations is the sum of all the path probabilities compatible with that sequence.

We can rearrange the path probability as follows:

$$\begin{aligned} P(o_{1,n}, s_{0,n}) &= \begin{array}{c} \text{Transition} \\ \prod_{i=1}^n A(s_{i-1}, s_i) \end{array} \times \begin{array}{c} \text{Observation} \\ B(s_i, o_i) \end{array} \\ &= \begin{array}{c} \text{Prior} \\ \prod_{i=1}^n \Pr(s_i | s_{i-1}) \end{array} \times \begin{array}{c} \text{Likelihood} \\ \prod_{i=1}^n \Pr(o_i | s_i) \end{array} \end{aligned}$$

That is you can divide the path probability into a “prior” component and a “likelihood” component. The former is just the product of the transition probabilities and the latter the product of the observation probabilities. So this hooks up with our noisy channel model metaphor, which is what explains all this strange talk about “emissions”. More on this next section.

But how does this map on to our tagging problem? More or less like this:

$$\begin{aligned} P(o_{1,n}, s_{0,n}) &= \begin{array}{c} \text{Prior} \\ \prod_{i=1}^n \Pr(s_i | s_{i-1}) \end{array} \times \begin{array}{c} \text{Likelihood} \\ \prod_{i=1}^n \Pr(o_i | s_i) \end{array} \\ P(w_{1,n}, t_{0,n}) &= \prod_{i=1}^n P(t_i | t_{i-1}) \times \prod_{i=1}^n P(w_i | t_i) \end{aligned}$$

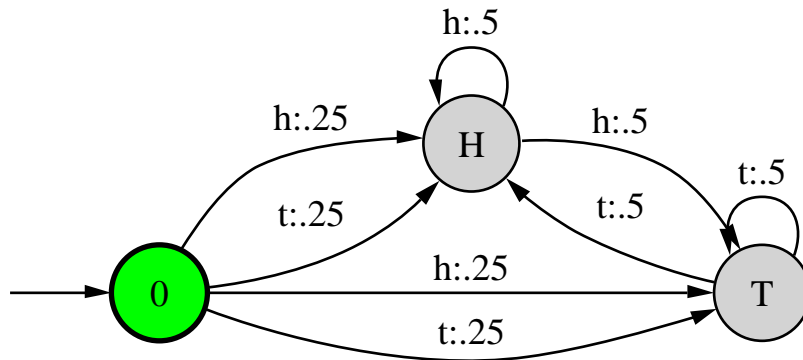


Figure 1: The coin flipping HMM

So the words become the observations. The tags become states. Tagging: given some words find the best sequence of tags. HMM Tagging: given some words find the best sequence of states, where best sequence means “most probable path”.

Example: Figure 1 shows a simple HMM for predicting coin-flipping sequences: 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, in effect betting that the next flip will be Tails. When in T you can accept a “t” either by moving to H or by remaining in T. 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. Similarly for “t”, giving us 4 ways of leaving the start state and no way of staying there. Thus, in every state, each time input is accepted, a “prediction” has to be made about what will come next.

We can facilitate the computation of path probabilities by combining the transition and and observation probabilities for each state. The following table does this for Figure 1. (For each transition, the observation probability is given first and the transition probability second).

```

0 :
  h
    0=>0: 0 * 0 = 0
    0=>H: 0.5 * 0.5 = 0.25
    0=>T: 0.5 * 0.5 = 0.25
  t

```

$$\begin{aligned}
0 \Rightarrow 0: & 0 * 0 = 0 \\
0 \Rightarrow H: & 0.5 * 0.5 = 0.25 \\
0 \Rightarrow T: & 0.5 * 0.5 = 0.25
\end{aligned}$$

$$0 \text{ (Total Probs)} = 1$$

H :

h

$$\begin{aligned}
H \Rightarrow 0: & 0 * 0 = 0 \\
H \Rightarrow H: & 1 * 0.5 = 0.5 \\
H \Rightarrow T: & 1 * 0.5 = 0.5
\end{aligned}$$

t

$$\begin{aligned}
H \Rightarrow 0: & 0 * 0 = 0 \\
H \Rightarrow H: & 0 * 0.5 = 0 \\
H \Rightarrow T: & 0 * 0.5 = 0
\end{aligned}$$

$$H \text{ (Total Probs)} = 1$$

T :

h

$$\begin{aligned}
T \Rightarrow 0: & 0 * 0 = 0 \\
T \Rightarrow H: & 0 * 0.5 = 0 \\
T \Rightarrow T: & 0 * 0.5 = 0
\end{aligned}$$

t

$$\begin{aligned}
T \Rightarrow 0: & 0 * 0 = 0 \\
T \Rightarrow H: & 1 * 0.5 = 0.5 \\
T \Rightarrow T: & 1 * 0.5 = 0.5
\end{aligned}$$

$$T \text{ (Total Probs)} = 1$$

Notice that for each state the combined transition/observation probabilities sum to 1.

Here are the partial path probabilities for one of two paths that will accept 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 1, and it is halved by each succeeding transition.

2.1 The Markov assumption and the noisy channel model

The Noisy Channel Model (NCM) says the the following:

$$\hat{s} = \operatorname{argmax}_{s \in S} \Pr(s) \Pr(o | s)$$

The “decoded” signal \hat{s} is the one that maximizes the probability of the signal s (the prior) times the probability of the observation given the signal (the likelihood). So this is where “emissions” comes into the language: “Emissions” (or “observations”) are signals emitted from our signal source as distorted by noise.

We can think of an HMM as being a way of extending the task of “decoding” a noisy signal to the task of decoding a **sequence of signals**, if we can make a certain key assumption called the Markov assumption.

Here’s how that works. Let’s use $s_{0,n}$ for the sequence of signals and $o_{1,n}$ for the sequence of observations. Then we get:

$$\hat{s} = \operatorname{argmax}_{s \in S} \Pr(s_{0,n}) \Pr(o_{1,n} | s_{0,n})$$

And the chain rule tells us:

$$\Pr(s_{0,n}) = \prod_{i=1}^n \Pr(s_i | s_{0,i-1}) \tag{5}$$

$$\Pr(o_{1,n} | s_{0,n}) = \Pr(o_1 | s_{0,n}) \prod_{i=2}^n \Pr(o_i | s_{0,i}, o_{1,i-1}) \tag{6}$$

When does the path probability through an HMM give us the product of these two probabilities?

It gives us these two probabilities when the following two equivalences hold, either approximately or actually:

$$\Pr(s_i | s_{1,n}) \cong \Pr(s_i | s_{i-1}) \tag{7}$$

$$\Pr(o_i | s_{1,n}, o_{1,i-1}) \cong \Pr(o_i | s_i) \tag{8}$$

Markov assumptions:

1. The probability of a signal depends only on the probability of the previous signal
2. The probability of an observation depend only the current signal

Returning to tagging, this turns into exactly the two locality assumptions we made for tagging.

$$\begin{aligned} \text{(a)} \quad & P(w_i | t_{0,i}, w_{1,i-1}) \cong P(w_i | t_i) \\ \text{(b)} \quad & P(t_i | w_{1,i-1}, t_{1,i}) \cong P(t_i | t_{i-1}) \end{aligned}$$

So given these assumptions, we may have the noisy channel metaphor, and given the noisy channel metaphor, we can formulate a more useful view of HMMs. HMMs are useful for *decoding* tasks. In a decoding task the goal is to find the sequence of states that gives the most probable path through the HMM for the given observations (emissions). The “hidden” states represent source signals, the observations their noise-distorted results. The most probable path is our best guess of the source signal. The truth is: you don’t know what path through the HMM produced the observations. But you do know what the most likely source is.

So in terms of tagging task, here is the strange picture we have: our source signals are tags. Their “noisy” results are words. By using a probability model that tells us the probability of words given tags and the probabilities of tags given tags, we are to “recover” (guess) the tags from the words they “emitted.”

But this really just shows the richness of the model. In fact HMMs can be used for any task where there is a probabilistic relation between sequences of observable data and some set of “hidden” properties of it. With the caveat, of course, that the Markov assumption is a reasonable approximation.

Is the Markov assumption a reasonable one for the tagging task? In some sense, no. But look at the numbers. The results are reasonably good.

3 HMMs and Viterbi: Locality Revisited

For the best path through an HMM we will use the Viterbi algorithm.

It is worth thinking through why HMMs and Viterbi dovetail so nicely. The reason is that they both build in locality assumptions.

Remember the basic Dynamic programming assumption of Viterbi: Suppose $s_{1,i}$ is a subpath of the best path through the network $s_{1,n}$ for the given observations $o_{1,n}$. Then $s_{1,i}$ must be the best path through the network for observations $o_{1,i}$. The reasoning is as follows: If there were a cheaper way to get from 1 to i than $s_{1,i}$, we could have used that for the $1, i$ segment and we could have found a cheaper path than $s_{1,n}$. So IF $s_{1,i}$ is a subpath of the best path through the network $s_{1,n}$, THEN $s_{1,i}$ must be the best path through the network for $o_{1,i}$.

But the only holds water if the cost of the path from s_i onward is independent of how you got to s_i . If the cost of getting from s_i to s_{i+1} is 1 if you got to s_i by one route and 100 if you got s_i by a different route, then all bets are off.

This situation arises when you try to use Viterbi for for speech recognition. Let's simplify by pretending states correspond to words. As we know, trigram dependencies do exist in natural language:

1. glazed baked ham
2. oven baked ham

When we consider the best way to get to *baked* at time t and compare coming from *glazed* to coming from *oven*, *oven* may well win:

$$\text{baked} \mid \text{X oven} > \text{baked} \mid \text{X glazed}$$

So the best path from the beginning of input to *baked* at time t may well take us through *oven* at time $t - 1$. But suppose the best path overall includes *baked* at time t . It may not include *oven baked*. It may include *glazed baked*. This will be true if:

$$\text{ham} \mid \text{glazed baked} > \text{ham} \mid \text{oven baked}$$

So this violates the dynamic programming assumption. The best path overall went through *baked* at time t . But it did not include the best path to *baked* at time t . This happens because the cost of the overall path takes into account how you got to *baked*. That is, the cost of moving from *baked* onward differs depending on what word comes before....

In brief, it depends on more than just the previous state. In this example, you had to look two states back because we hypothesized a trigram dependency.

This kind of situation is exactly what the Markov assumption precludes. So using the Viterbi algorithm for decoding is exactly as valid as using the HMM for decoding is. No more. No less.