Part-Of-Speech (POS) Tagging: Hidden Markov Model (HMM) algorithm
Hidden Markov Models

- Recall that we estimated the best probable tag sequence for a given sequence of words as:

\[
\hat{t}_1^n = \arg\max_{t_1^n} P(t_1^n|w_1^n) \approx \arg\max_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})
\]

with the word likelihood x the tag transition probabilities

- When we evaluated the probabilities by hand for a sentence, we could pick the optimum tag sequence

- But in general, we need an optimization algorithm to most efficiently pick the best tag sequence without computing all possible combinations of tag sequence probabilities

- What we’ve described with these two kinds of probabilities is a Hidden Markov Model
  - the Markov Model is the sequence of words and the hidden states are the POS tags for each word.
Tag Transition Probabilities for an HMM

- The HMM hidden states, the POS tags, can be represented in a graph where the edges are the transition probabilities between POS tags.
Word likelihoods for POS HMM

• For each POS tag, give words with probabilities
The A matrix for the POS HMM

• Example of tag transition probabilities represented in a matrix, usually called the A matrix in an HMM:
  - The probability that VB follows <s> is .019, ...

<table>
<thead>
<tr>
<th></th>
<th>VB</th>
<th>TO</th>
<th>NN</th>
<th>PPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;s&gt;</td>
<td>.019</td>
<td>.0043</td>
<td>.041</td>
<td>.067</td>
</tr>
<tr>
<td>VB</td>
<td>.0038</td>
<td>.035</td>
<td>.047</td>
<td>.0070</td>
</tr>
<tr>
<td>TO</td>
<td>.83</td>
<td>0</td>
<td>.00047</td>
<td>0</td>
</tr>
<tr>
<td>NN</td>
<td>.0040</td>
<td>.016</td>
<td>.087</td>
<td>.0045</td>
</tr>
<tr>
<td>PPSS</td>
<td>.23</td>
<td>.00079</td>
<td>.0012</td>
<td>.00014</td>
</tr>
</tbody>
</table>

Figure 4.15 Tag transition probabilities (the a array, \( p(t_i|t_{i-1}) \)) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus \( P(PPSS|VB) \) is .0070. The symbol <s> is the start-of-sentence symbol.
The B matrix for the POS HMM

- Word likelihood probabilities are represented in a matrix, where for each tag, we show the probability that the tag on that word

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>want</th>
<th>to</th>
<th>race</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td>0</td>
<td>.0093</td>
<td>0</td>
<td>.00012</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
<td>0</td>
<td>.99</td>
<td>0</td>
</tr>
<tr>
<td>NN</td>
<td>0</td>
<td>.000054</td>
<td>0</td>
<td>.00057</td>
</tr>
<tr>
<td>PPSS</td>
<td>.37</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 4.16** Observation likelihoods (the $b$ array) computed from the 87-tag Brown corpus without smoothing.
Using HMMs for POS tagging

• From the tagged corpus, create a tagger by computing the two matrices of probabilities, $A$ and $B$
  – Straightforward for bigram HMM, done by counting
  – For higher-order HMMs, efficiently compute matrix by the forward-backward algorithm

• To apply the HMM tagger to unseen text, we must find the best sequence of transitions
  – Given a sequence of words, find the sequence of states (POS tags) with the highest probabilities along the path
  – This task is sometimes called “decoding”
  – Use the Viterbi algorithm
Viterbi intuition: we are looking for the best ‘path’

Each word has states representing the possible POS tags:

\[ S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5 \]

promised to back the bill
Viterbi example

Each pair of tags labeled with an edge giving transition probability
Each tag in a state labeled with a Viterbi value giving max over states in previous word of:
(its viterbi value * transition prob * word likelihood), representing “best path to this node”
Viterbi Algorithm sketch

- This algorithm fills in the elements of the array viterbi in the previous slide (cols are words, rows are states (POS tags))

function Viterbi

    for each state s, compute the initial column
    viterbi[s, 1] = A[0, s] * B[s, word1]

    for each word w from 2 to N (length of sequence)
    for each state s, compute the column for w
    viterbi[s, w] =
        max over s’ ( viterbi[s’,w-1] * A[s’,s] * B[s,w])

<save back pointer to trace final path>
return the trace of back pointers

where A is the matrix of state transitions
    and B is the matrix of state/word likelihoods