Part-of-Speech Tagging

- Each word must be assigned its correct part-of-speech, such as noun, verb, adjective, or adverb, based on its function in a sentence.

- Simple heuristics go a long way! You get about 90% accuracy by choosing the most frequent tag for a word based on a large training corpus.

- Most POS taggers are statistical or rule-based.

- Statistical taggers can achieve about 97% accuracy, but they require training data and do not explicitly represent intuitive rules.

POS Tag Sets

- There is no universally agreed upon set of part-of-speech tags!

- The size of different tag sets can vary a lot.
  
  Penn Treebank uses 45 tags
  
  Original Brown Corpus used 87 tags
  
  British National Corpus Basic Tagset (C5) used 61 tags
  
  Enriched C6 tagset used 160 tags
  
  CMU POS tagger for Twitter used 25 tags

Rule-based POS Tagging

- Rule-based taggers rely on a dictionary to provide possible POS tags for a word, or rules can be learned using training data.

- Manually developed disambiguation rules can perform reasonably well.

Example rules:

If preceding word = ART, then disambiguate {NOUN, VERB} as NOUN.

If a possible verb does not agree in number with the preceding NP, then eliminate the verb tag.

If the preceding word takes an S complement, then tag “that” as a subordinating conjunction (vs. determiner).

Statistical Part-of-Speech Tagging

- Statistical part-of-speech tagging involves selecting the most likely sequence of tags for the words in a sentence.

- What we really want to calculate is: $P(T_1...T_n \mid w_1...w_n)$ but this would require an unreasonable amount of data.

- We could apply Bayes’ rule and calculate:

  $\frac{(P(T_1...T_n) \ast P(w_1...w_n \mid T_1...T_n))}{P(w_1...w_n)}$ but this still requires too much data.

  Instead, we can approximate this function by making independence assumptions based on part-of-speech tag bigrams and lexical generation probabilities.
A Complete POS Tagging Model using Tag Bigrams

\[ P(T_1...T_n \mid w_1...w_n) \]

\[ \Rightarrow \frac{P(T_1...T_n) \cdot P(w_1...w_n \mid T_1...T_n)}{P(w_1...w_n)} \]

\[ \Rightarrow \prod_{i=1}^{n} P(T_i \mid T_{i-1}) \cdot P(w_i \mid T_i) \]

Probability Definitions

We use bigram transition probabilities to estimate the probability that one POS tag will follow another:

\[ P(T_1...T_n) = \prod_{i=1}^{n} P(T_i \mid T_{i-1}) \]

We use lexical generation probabilities to estimate the probability that a POS tag will generate a particular word, independent of the surrounding words:

\[ P(w_1...w_n \mid T_1...T_n) = \prod_{i=1}^{n} P(w_i \mid T_i) \]

Estimating Probabilities

Given a training corpus in which each word has been labeled with its correct part-of-speech tag, you can compute probability estimates from the frequency counts.

Ex: the probability that \( T_i \) follows \( T_{i-1} \) is computed as:

\[ P(T_i \mid T_{i-1}) = \frac{\#(T_i \text{ immediately follows } T_{i-1})}{\#(\text{any tag immediately follows } T_{i-1})} \]

Given some tag \( T_i \) the probability of a particular word \( w_i \) is:

\[ P(w_i \mid T_i) = \frac{\#(w_i \text{ with tag } T_i)}{\#(\text{any word with tag } T_i)} \]

Sample Probabilities

<table>
<thead>
<tr>
<th>Tag Frequencies</th>
<th>Tag Frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td>ART</td>
</tr>
<tr>
<td>300</td>
<td>633</td>
</tr>
</tbody>
</table>

Bigram Probabilities

<table>
<thead>
<tr>
<th>Bigram ( (T_{i-1}, T_i) )</th>
<th>Count ( (i, i+1) )</th>
<th>Prob ( (T_i \mid T_{i-1}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi, \text{ART} )</td>
<td>213</td>
<td>.71</td>
</tr>
<tr>
<td>( \phi, \text{N} )</td>
<td>87</td>
<td>.29</td>
</tr>
<tr>
<td>( \text{ART, N} )</td>
<td>633</td>
<td>1</td>
</tr>
<tr>
<td>( \text{N, V} )</td>
<td>358</td>
<td>.32</td>
</tr>
<tr>
<td>( \text{N, N} )</td>
<td>108</td>
<td>.10</td>
</tr>
<tr>
<td>( \text{N, P} )</td>
<td>366</td>
<td>.33</td>
</tr>
<tr>
<td>( \text{V, N} )</td>
<td>134</td>
<td>.37</td>
</tr>
<tr>
<td>( \text{V, ART} )</td>
<td>194</td>
<td>.54</td>
</tr>
<tr>
<td>( \text{P, ART} )</td>
<td>226</td>
<td>.62</td>
</tr>
<tr>
<td>( \text{P, N} )</td>
<td>140</td>
<td>.38</td>
</tr>
</tbody>
</table>
Sample Lexical Generation Probabilities

<table>
<thead>
<tr>
<th></th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(the</td>
<td>ART)</td>
</tr>
<tr>
<td>P(a</td>
<td>ART)</td>
</tr>
<tr>
<td>P(flies</td>
<td>N)</td>
</tr>
<tr>
<td>P(a</td>
<td>N)</td>
</tr>
<tr>
<td>P(flower</td>
<td>N)</td>
</tr>
<tr>
<td>P(birds</td>
<td>N)</td>
</tr>
<tr>
<td>P(like</td>
<td>N)</td>
</tr>
<tr>
<td>P(flies</td>
<td>V)</td>
</tr>
<tr>
<td>P(like</td>
<td>V)</td>
</tr>
<tr>
<td>P(flower</td>
<td>V)</td>
</tr>
<tr>
<td>P(like</td>
<td>P)</td>
</tr>
</tbody>
</table>

Note: the probabilities for a given tag don’t add up to 1, so this table must not be complete. These are just sample values.

Tag Sequences

- In theory, we want to generate all possible tag sequences to determine which one is most likely. But there are an exponential number of possible tag sequences!

- Because of the independence assumptions already made, we can use Markov models. Each node represents a tag and each arc represents the probability of one tag following another.

- The probability of a tag sequence is computed by finding the appropriate path through the network and multiplying the probabilities.

- In a Hidden Markov Model (HMM), the lexical (word | tag) probabilities are stored with each node. For example, all of the (word | ART) probabilities would be stored with the ART node. Then everything can be computed with just the HMM!

The Viterbi Algorithm

- The Viterbi algorithm is used to compute the most likely tag sequence in $O(W \times T^2)$ time, where $T$ is the number of possible part-of-speech tags and $W$ is the number of words in the sentence.

- The algorithm sweeps through all the tag possibilities for each word, computing the best sequence leading to each possibility. The key that makes this algorithm efficient is that we only need to know the best sequences leading to the previous word because of the Markov assumption.

- Statistical part-of-speech taggers can achieve roughly 95-97% accuracy.

Computing the Probability of a Sentence and Tags

We want to find the sequence of tags that maximizes the formula $P(T_1..T_n | w_1..w_n)$, which can be estimated as:

$$\prod_{i=1}^{n} P(T_i | T_{i-1}) \times P(w_i | T_i)$$

$P(T_i | T_{i-1})$ is computed by multiplying the arc values in the HMM.

$P(w_i | T_i)$ is computed by multiplying the lexical generation probabilities associated with each word.
The Viterbi Algorithm

Let \( T \) = \# of part-of-speech tags \( W \) = \# of words in the sentence

for \( t = 1 \) to \( T \) /* Initialization Step */
\[
\text{Score}(t, 1) = \text{Pr}(\text{Word}_1 | \text{Tag}_t) \times \text{Pr}(\text{Tag}_t | \phi)
\]
\( \text{BackPtr}(t, 1) = 0; \)

for \( w = 2 \) to \( W \) /* Iteration Step */
for \( t = 1 \) to \( T \)
\[
\text{Score}(t, w) = \text{Pr}(\text{Word}_w | \text{Tag}_t) \times \max_{j=1,T} (\text{Score}(j, w-1) \times \text{Pr}(\text{Tag}_t | \text{Tag}_j))
\]
\( \text{BackPtr}(t, w) = \text{index of } j \text{ that gave the max above} \)

\( \text{Seq}(W) = t \text{ that maximizes } \text{Score}(t,W) \) /* Sequence Identification */
for \( w = W-1 \) to \( 1 \)
\( \text{Seq}(w) = \text{BackPtr}(\text{Seq}(w+1),w+1) \)

Using the POS tagger

- We could use our statistical POS tagger to determine the most likely part-of-speech for each word and give them to the parser.
- This approach would maximally reduce the amount of lexical ambiguity that the parser needs to deal with. It would only receive one part-of-speech per word.
- However, if the tagger makes any mistakes then the parser will be doomed to fail!
- Remember that even though statistical taggers can get about 95% accuracy, that still means that 1 in 20 words is mistagged.

Assigning Tags Probabilistically

- Instead of identifying only the best tag for each word, a better approach is to assign a probability to each tag.
- We could use simple frequency counts to estimate context-independent probabilities.
\[
P(\text{tag} \mid \text{word}) = \frac{\# \text{times word occurs with the tag}}{\# \text{times word occurs}}
\]
- But these estimates are unreliable because they do not take context into account.
- A better approach considers how likely a tag is for a word given the specific sentence and words around it!

An Example

Consider the sentence: *Outside pets are often hit by cars.*

Assume “outside” has 4 possible tags: ADJ, NOUN, PREP, ADVERB.
Assume “pets” has 2 possible tags: VERB, NOUN.

If “outside” is a ADJ or PREP then “pets” has to be a NOUN.
If “outside” is a ADV or NOUN then “pets” may be a NOUN or VERB.

Now we can sum the probabilities of all tag sequences that end with “pets” as a NOUN and sum the probabilities of all tag sequences that end with “pets” as a VERB. For this sentence, the chances that “pets” is a NOUN should be much higher.
Forward Probability

- The forward probability $\alpha_i(m)$ is the probability of words $w_1...w_m$ with $w_m$ having tag $T_i$.

  $$\alpha_i(m) = P(w_1...w_m \& w_m/T_i)$$

- The forward probability is computed as the sum of the probabilities computed for all tag sequences ending in tag $T_i$ for word $w_m$.

  Ex: $\alpha_1(2)$ would be the sum of probabilities computed for all tag sequences ending in tag #1 for word #2.

- The lexical tag probability is computed as:

  $$P(w_m/T_i | w_1...w_m) = \frac{P(w_m/T_i, w_1...w_m)}{P(w_1...w_m)}$$

  which we estimate as:

  $$P(w_m/T_i | w_1...w_m) = \frac{\alpha_i(m)}{\sum_{j=1}^{T} \alpha_j(m)}$$

Backward Probability

- Backward probability $\beta_i(m)$ is the probability of words $w_m...w_N$ with $w_m$ having tag $T_i$.

  $$\beta_i(m) = P(w_m...w_N \& w_m/T_i)$$

- The backward probability is computed as the sum of the probabilities computed for all tag sequences beginning with tag $T_i$ for word $w_m$.

- The algorithm for computing the backward probability is analogous to the forward probability except that we start at the end of the sentence and sweep backwards.

- The best way to estimate lexical tag probabilities uses both forward and backward probabilities:

  $$P(w_m/T_i) = \frac{\sum_{j=1}^{T} (\alpha_i(m) \times \beta_j(m))}{\sum_{j=1}^{T} (\alpha_j(m) \times \beta_j(m))}$$

The Forward Algorithm

Let $T = \#$ of part-of-speech tags  \hspace{1em} $W = \#$ of words in the sentence

for $t = 1$ to $T$ \hspace{1em} /* Initialization Step */

  $SeqSum(t, 1) = Pr(Word_1 | Tag_t) \times Pr(Tag_t | \phi)$

for $w = 2$ to $W$ \hspace{1em} /* Compute Forward Probs */

  for $t = 1$ to $T$

    $SeqSum(t, w) = Pr(Word_w | Tag_t) \times \sum_{j=1,T} (SeqSum(j, w-1) \times Pr(Tag_t | Tag_j))$

for $w = 1$ to $W$ \hspace{1em} /* Compute Lexical Probs */

  for $t = 1$ to $T$

    $Pr(Seq_w=Tag_t) = \sum_{j=1,T} \frac{SeqSum(t, w)}{SeqSum(j, w)}$