Part of Speech Tagging
& Hidden Markov Models (Part 1)

Mitch Marcus
CIS 421/521
NLP Task I – Determining Part of Speech Tags

- Given a text, assign each token its correct *part of speech (POS) tag*, given its context and a list of *possible* POS tags for each word type

<table>
<thead>
<tr>
<th>Word</th>
<th>POS listing in Brown Corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>heat</td>
<td><strong>noun</strong></td>
</tr>
<tr>
<td>oil</td>
<td><strong>noun</strong></td>
</tr>
<tr>
<td>in</td>
<td><strong>prep</strong></td>
</tr>
<tr>
<td>a</td>
<td><strong>det</strong></td>
</tr>
<tr>
<td>large</td>
<td><strong>adj</strong></td>
</tr>
<tr>
<td>pot</td>
<td><strong>noun</strong></td>
</tr>
</tbody>
</table>
What is POS tagging good for?

- **Speech synthesis:**
  - How to pronounce “lead”?
  - INsult \(→\) inSULT
  - OBject \(→\) obJECT
  - OVERflow \(→\) overFLOW
  - DIScount \(→\) disCOUNT
  - CONtent \(→\) content

- **Machine Translation**
  - translations of nouns and verbs are different

- **Stemming for search**
  - Knowing a word is a V tells you it gets past tense, participles, etc.
  - Can search for “walk”, can get “walked”, “walking,…”
Equivalent Problem in Bioinformatics

- From a sequence of amino acids (primary structure): ATCPELELLLD

- Infer secondary structure (features of the 3D structure, like helices, sheets, etc.): HHHBBBBBCC..
## Penn Treebank Tagset I

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>coordinating conjunction</td>
<td>and</td>
</tr>
<tr>
<td>CD</td>
<td>cardinal number</td>
<td>1, third</td>
</tr>
<tr>
<td>DT</td>
<td>determiner</td>
<td>the</td>
</tr>
<tr>
<td>EX</td>
<td>existential <em>there</em></td>
<td><em>there</em> is</td>
</tr>
<tr>
<td>FW</td>
<td>foreign word</td>
<td>d'hoevre</td>
</tr>
<tr>
<td>IN</td>
<td>preposition/subordinating conjunction</td>
<td>in, of, like</td>
</tr>
<tr>
<td>JJ</td>
<td>adjective</td>
<td>green</td>
</tr>
<tr>
<td>JJR</td>
<td>adjective, comparative</td>
<td>greener</td>
</tr>
<tr>
<td>JJS</td>
<td>adjective, superlative</td>
<td>greenest</td>
</tr>
<tr>
<td>LS</td>
<td>list marker</td>
<td>1)</td>
</tr>
<tr>
<td>MD</td>
<td>modal</td>
<td>could, will</td>
</tr>
<tr>
<td>NN</td>
<td>noun, singular or mass</td>
<td>table</td>
</tr>
<tr>
<td>NNS</td>
<td>noun plural</td>
<td>tables (supports)</td>
</tr>
<tr>
<td>NNP</td>
<td>proper noun, singular</td>
<td>John</td>
</tr>
<tr>
<td>NNPS</td>
<td>proper noun, plural</td>
<td>Vikings</td>
</tr>
</tbody>
</table>
# Penn Treebank Tagset II

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDT</td>
<td>predeterminer</td>
<td><em>both</em> the boys</td>
</tr>
<tr>
<td>POS</td>
<td>possessive ending</td>
<td>friend 's</td>
</tr>
<tr>
<td>PRP</td>
<td>personal pronoun</td>
<td>I, me, him, he, it</td>
</tr>
<tr>
<td>PRP$</td>
<td>possessive pronoun</td>
<td>my, his</td>
</tr>
<tr>
<td>RB</td>
<td>adverb</td>
<td>however, usually, here, good</td>
</tr>
<tr>
<td>RBR</td>
<td>adverb, comparative</td>
<td>better</td>
</tr>
<tr>
<td>RBS</td>
<td>adverb, superlative</td>
<td>best</td>
</tr>
<tr>
<td>RP</td>
<td>particle</td>
<td>give <em>up</em></td>
</tr>
<tr>
<td>TO</td>
<td><em>to</em></td>
<td><em>to go, to</em> him</td>
</tr>
<tr>
<td>UH</td>
<td>interjection</td>
<td>uhhuhhuhhh</td>
</tr>
</tbody>
</table>
## Penn Treebank Tagset III

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td>verb, base form</td>
<td>take (support)</td>
</tr>
<tr>
<td>VBD</td>
<td>verb, past tense</td>
<td>took</td>
</tr>
<tr>
<td>VBG</td>
<td>verb, gerund/present participle</td>
<td>taking</td>
</tr>
<tr>
<td>VBN</td>
<td>verb, past participle</td>
<td>taken</td>
</tr>
<tr>
<td>VBP</td>
<td>verb, sing. present, non-3d</td>
<td>take</td>
</tr>
<tr>
<td>VBZ</td>
<td>verb, 3rd person sing. present</td>
<td>takes (supports)</td>
</tr>
<tr>
<td>WDT</td>
<td>wh-determiner</td>
<td>which</td>
</tr>
<tr>
<td>WP</td>
<td>wh-pronoun</td>
<td>who, what</td>
</tr>
<tr>
<td>WPS</td>
<td>possessive wh-pronoun</td>
<td>whose</td>
</tr>
<tr>
<td>WRB</td>
<td>wh-abverb</td>
<td>where, when</td>
</tr>
</tbody>
</table>
NLP Task I – Determining Part of Speech Tags

- **The Old Solution**: *Depth First search.*
  - If each of $n$ word tokens has $k$ tags on average, try the $k^n$ combinations until one works.

- **Machine Learning Solutions**: *Automatically learn Part of Speech (POS) assignment.*
  - The best techniques achieve 97+% accuracy per word on new materials, given a POS-tagged training corpus of $10^6$ tokens with 3% error on a set of ~40 POS tags (tags on the last three slides)
Simple Statistical Approaches: Idea 1

Simply assign each word its most likely POS.
Success rate: 91%!

<table>
<thead>
<tr>
<th>Word</th>
<th>POS listings in Brown</th>
</tr>
</thead>
<tbody>
<tr>
<td>heat</td>
<td>noun/89, verb/5</td>
</tr>
<tr>
<td>oil</td>
<td>noun/87</td>
</tr>
<tr>
<td>in</td>
<td>prep/20731, noun/1, adv/462</td>
</tr>
<tr>
<td>a</td>
<td>det/22943, noun/50, noun-proper/30</td>
</tr>
<tr>
<td>large</td>
<td>adj/354, noun/2, adv/5</td>
</tr>
<tr>
<td>pot</td>
<td>noun/27</td>
</tr>
</tbody>
</table>
Simple Statistical Approaches: Idea 2

For a string of words

\[ W = w_1w_2w_3...w_n \]

find the string of POS tags

\[ T = t_1t_2t_3...t_n \]

which maximizes \( P(T \mid W) \)

- i.e., the most likely POS tag \( t_i \) for each word \( w_i \) given its surrounding context
The Sparse Data Problem …

A Simple, Impossible Approach to Compute $P(T \mid W)$:

Count up instances of the string "heat oil in a large pot" in the training corpus, and pick the most common tag assignment to the string.
What parameters can we estimate with a million words of hand tagged training data?

- Assume a uniform distribution of 5000 words and 40 part of speech tags.

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Estimate Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>tags</td>
<td>40</td>
<td>Excellent</td>
</tr>
<tr>
<td>tag bigrams</td>
<td>1600</td>
<td>Excellent</td>
</tr>
<tr>
<td>tag trigrams</td>
<td>64,000</td>
<td>OK</td>
</tr>
<tr>
<td>tag 4-grams</td>
<td>2.5M</td>
<td>Poor</td>
</tr>
<tr>
<td>words</td>
<td>5000</td>
<td>Very Good</td>
</tr>
<tr>
<td>word bigrams</td>
<td>25M</td>
<td>Poor</td>
</tr>
<tr>
<td>word x tag pairs</td>
<td>200,000</td>
<td>OK</td>
</tr>
</tbody>
</table>

We can get reasonable estimates of

- Tag bigrams
- Word x tag pairs
Bayes Rule plus Markov Assumptions yields a practical POS tagger!

I. By Bayes Rule

\[ P(T \mid W) = \frac{P(W \mid T) \times P(T)}{P(W)} \]

II. So we want to find

\[ \arg \max_T P(T \mid W) = \arg \max_T P(W \mid T) \times P(T) \]

III. To compute \( P(W \mid T) \):

- use the chain rule + a Markov assumption
- Estimation requires word x tag and tag counts

IV. To compute \( P(T) \):

- use the chain rule + a slightly different Markov assumption
- Estimation requires tag unigram and bigram counts
IV. To compute $P(T)$:

*Just like computing $P(W)$ last lecture*

I. By the chain rule,

$$P(T) = P(t_1) * P(t_2 \mid t_1) * P(t_3 \mid t_1t_2) * \ldots * P(t_n \mid t_1 \ldots t_{n-1})$$

II. Applying the 1st order Markov Assumption

$$P(T) = P(t_1) * P(t_2 \mid t_1) * P(t_3 \mid t_2) * \ldots * P(t_n \mid t_{n-1})$$

*Estimated using tag bigrams/tag unigrams!*
III. To compute $P(W|T)$:

I. Assume that the words $w_i$ are conditionally independent given the tag sequence $T=t_1t_2...t_n$:  
$$P(W | T) = \prod_{i=1}^{n} P(w_i | T)$$

II. Applying a zeroth-order Markov Assumption:

$$P(w_i | T) = P(w_i | t_i)$$

by which

$$P(W | T) = \prod_{i=1}^{n} P(w_i | t_i)$$

So, for a given string $W = w_1w_2w_3...w_n$, the tagger needs to find the string of tags $T$ which maximizes

$$P(T) \times P(W|T) = P(t_1) \times P(t_2|t_1) \times P(t_3|t_2) \times ... \times P(t_n|t_{n-1}) \times P(w_1|t_1) \times P(w_2|t_2) \times ... \times P(w_n|t_n)$$
Hidden Markov Models

This model is an instance of a Hidden Markov Model. Viewed graphically:

P(w|Det)  
ap .4  
the .4

P(w|Adj)  
good .02  
low .04

P(w|Noun)  
price .001  
deal .0001
Viewed as a generator, an HMM:

- Starts in some initial state $t_1$ with probability $\pi(t_1)$,
- On each move goes from state $t_i$ to state $t_j$ according to transition probability $a(t_i, t_j)$.
- At each state $t_i$, it emits a symbol $w_k$ according to the emit probabilities $b(t_i, w_k)$.
Summary: Recognition using an HMM

I. By Bayes Rule

\[ P(T \mid W) = \frac{P(T) * P(W \mid T)}{P(W)} \]

II. We select the Tag sequence \( T \) that maximizes \( P(T \mid W) \):

\[
\arg \max_T P(T \mid W) = \arg \max_{T=t_1t_2\ldots t_n} P(T) * P(W \mid T) = \arg \max_{T=t_1t_2\ldots t_n} \pi(t_1) * \prod_{i=1}^{n-1} a(t_i, t_{i+1}) * \prod_{i=1}^{n} b(t_i, w_i)
\]
Training and Performance

- To estimate the parameters of this model, given an annotated training corpus use the MLE:

  \[
  \text{To estimate } P(t_i|t_{i-1}): \quad \frac{\text{Count}(t_{i-1}t_i)}{\text{Count}(t_{i-1})}
  \]

  \[
  \text{To estimate } P(w_i|t_i): \quad \frac{\text{Count}(w_i \text{ tagged } t_i)}{\text{Count}(\text{ all words tagged } t_i)}
  \]

- Because many of these counts are small, smoothing is necessary for best results…

- Such taggers typically achieve about 95-96% correct tagging, for the standard 40-tag POS set.

- A few tricks for unknown words increase accuracy to 97%.
POS from bigram and word-tag pairs??

A Practical compromise

- Rich Models often require vast amounts of data
- Well estimated bad models often outperform badly estimated truer models

(Mutt & Jeff 1942)
Practical Tagging using HMMs

- Finding this maximum can be done using an exponential search through all strings for $T$.

- However, there is a *linear time* solution using *dynamic programming* called *Viterbi decoding*. 
The three basic HMM problems
Parameters of an HMM

- **States**: A set of states $S = \{s_1, \ldots, s_n\}$
- **Transition probabilities**: $A = a_{1,1}, a_{1,2}, \ldots, a_{n,n}$
  
  Each $a_{i,j}$ represents the probability of transitioning from state $s_i$ to $s_j$.

- **Emission probabilities**: a set $B$ of functions of the form $b_i(o_t)$ which is the probability of observation $o_t$ being emitted by $s_i$

- **Initial state distribution**: $\pi_i$ is the probability that $s_i$ is a start state

(This and later slides follow classic formulation by Ferguson, as published by Rabiner and Juang, as adapted by Manning and Schutze. Note the change in notation!!)
The Three Basic HMM Problems

- **Problem 1 (Evaluation):** Given the observation sequence \( O = o_1, \ldots, o_T \) and an HMM model \( \lambda = (A, B, \pi) \), how do we compute the probability of \( O \) given the model?

- **Problem 2 (Decoding):** Given the observation sequence \( O \) and an HMM model \( \lambda \), how do we find the state sequence that best explains the observations?

- **Problem 3 (Learning):** How do we adjust the model parameters \( \lambda = (A, B, \pi) \), to maximize \( P(O \mid \lambda) \)?
Problem 1: Probability of an Observation Sequence

- Q: What is $P(O | \lambda)$?
- A: the sum of the probabilities of all possible state sequences in the HMM.

- Naïve computation is very expensive. Given $T$ observations and $N$ states, there are $N^T$ possible state sequences.
  - (for $T=10$ and $N=10$, 10 billion different paths!!)
- Solution: linear time dynamic programming!
The Crucial Data Structure: The Trellis
Forward Probabilities: $\alpha$

- For a given HMM $\lambda$, for some time $t$, what is the probability that the partial observation $o_1 \ldots o_t$ has been generated and that the state at time $t$ is $i$?

$$\alpha_t(i) = P(o_1 \ldots o_t, q_t = s_i \mid \lambda)$$

- **Forward algorithm computes** $\alpha_t(i)$, $0 < i < N$, $0 < t < T$ in time $O(N^2T)$ using the trellis
Forward Algorithm: Induction step

\[
\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \cdot b_j(o_t)
\]

\[
\alpha_t(i) = P(o_1...o_t, q_t = s_i | \lambda)
\]
Forward Algorithm

- Initialization (probability that $o_1$ has been generated and that the state is $i$ at time $t=1$):
  \[
  \alpha_1(i) = \pi_i b_i(o_1) \quad 1 \leq i \leq N
  \]

- Induction:
  \[
  \alpha_t(j) = \left[ \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(o_t) \quad 2 \leq t \leq T, \quad 1 \leq j \leq N
  \]

- Termination:
  \[
  P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)
  \]
Forward Algorithm Complexity

- Naïve approach requires exponential time to evaluate all $N^T$ state sequences
- Forward algorithm using dynamic programming takes $O(N^2T)$ computations