**POS Tagsets and POS Tagging**

Def. Part of Speech Tagging

POS Tagging = Assigning word class information to words

ex: the man bought a book
determiner noun verb determiner noun

Linguistic Questions

- How do we divide the text into individual **word tokens**?
- How do we choose a **tagset** to represent all words?
- How do we select appropriate **tags** for individual **words**?

Tokenization: Multiwords

Multiwords:
- *in spite of* the firm promise
- *because of* technical problems

Merged words:
- *I don’t know*
- *he couldn’t come*

Compounds:
- *Great Northern Nekoosa Corp.*
- *an Atlanta-based forest-products company*

Possible Solution: Layered Analysis

```plaintext
<w pos=in>in spite of</w>
<w pos=md+rb>shouldn’t</w>
<w pos=jj>hundreds</w>
-<w pos=in>of</w>
-<w pos=nns>billions</w>
-<w pos=in>of</w>
-<w pos=nns>yen</w>
<w pos=nn>market</w>
```

Issues in Tagset Design

- define which words are considered multiwords: no multiwords, ditto tags, layered annotation
- describe how mergers are treated: combined tags, splitting up
- describe how compounds are treated: surface oriented, layered annotation
- can the solutions be implemented automatically
### Issues in Selecting a Tagset

- **Conciseness**: short labels better than long ones
  - prep ⇒ preposition
- **Perspicuity**: labels that are easily interpreted are better
  - prep ⇒ in
- **Analyzability**: should be possible to decompose in different parts
  - **vmfin**: verb, modal, finite
  - **pds**: pronoun, demonstrative, substituting

### Tagset Size

<table>
<thead>
<tr>
<th>Language</th>
<th>Tagset Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td></td>
</tr>
<tr>
<td>TOSCA</td>
<td>32</td>
</tr>
<tr>
<td>Penn treebank</td>
<td>36</td>
</tr>
<tr>
<td>BNC C5</td>
<td>61</td>
</tr>
<tr>
<td>Brown</td>
<td>77</td>
</tr>
<tr>
<td>LOB</td>
<td>132</td>
</tr>
<tr>
<td>London-Lund Corpus</td>
<td>197</td>
</tr>
<tr>
<td>TOSCA-ICE</td>
<td>270</td>
</tr>
<tr>
<td>Romanian</td>
<td>614</td>
</tr>
<tr>
<td>Hungarian</td>
<td>ca. 2 100</td>
</tr>
</tbody>
</table>

### Annotating POS Tags

Two fundamentally different approaches:
- **start from scratch**, find characteristics in words or context (= rules) which give indication of word class
  - i.e. if word ends in ‘‘ion’’, tag it as noun
- **accumulate lexicon**, disambiguate words with more than one tag
  - i.e. possible categories for ‘‘about’’:
    - preposition, adverb, particle

### POS Representations

#### Horizontal Format

```plaintext
I/PP will/MD then/RB maybe/RB travel/VB
directly/RB on/IN to/IN Berlin/NP
```

#### Vertical Format

```plaintext
I         PP
will      MD
then      RB
maybe     RB
travel    VB
directly  RB
on        IN
to        IN
Berlin    NP
```

### Penn Treebank Tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Prep. / subord. conj.</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PRP$</td>
<td>Possessive pronoun</td>
</tr>
</tbody>
</table>

### Automatic POS Tagging

**Assumption**: local context is sufficient examples:
- for the man: noun or verb?
- we will man: noun or verb?
- I can put: verb base form or past?
- re-cap real quick: adjective or adverb?
Bigram Tagging

- basic assumption: POS tag only depends on word itself and on the POS tag of the previous word
- use lexicon to retrieve ambiguity class for words e.g. word: beginning, ambiguity class: [JJ, NN, VBG]
- for unknown words: use heuristics, e.g. all open class POS tags
- disambiguation: look for most likely path through possibilities

Bigram Tagging – Probabilities

\[ P(t_1 \ldots t_5) = \frac{P(t_1 | S)P(w_1 | t_1)P(t_2 | t_1)P(w_2 | t_2)\ldots}{P(t_5 | t_4)P(w_5 | t_5)P(E | t_5)} \]

green = transition probabilities
blue = lexical probabilities

Bigram Tagging – Counter-Examples

- start before
- start before the course or start before he is done
Bigram Tagging – Counter-Examples

- start before
  - start before the course or start before he is done
- real quick
  - re-cap real quick or a real quick lunch

Maximum Likelihood Estimation

Simplest way to calculate such probabilities from a corpus:

\[
P_{ML}(t_n | t_{n-1}) = \frac{C(t_n, t_{n-1})}{C(t_{n-1})}
\]

\[
P_{ML}(w_n | t_n) = \frac{C(w_n, t_n)}{C(t_n)}
\]

- not a very good estimator

- uses relative frequency
- maximizes the probabilities of the corpus
Maximum Likelihood Estimation (2)

- Not a very good estimator
- Zero probabilities for unseen events: makes them impossible
- Need smoothing or discounting method to give minimal probabilities to unseen events
- Simplest possibility: learn from hapax legomena (words that appear only once)

Example HMM

Assume DET, N, and VB as hidden states, with this transition matrix (A):

<table>
<thead>
<tr>
<th></th>
<th>DET</th>
<th>N</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DET</td>
<td>0.01</td>
<td>0.89</td>
<td>0.10</td>
</tr>
<tr>
<td>N</td>
<td>0.30</td>
<td>0.20</td>
<td>0.50</td>
</tr>
<tr>
<td>VB</td>
<td>0.67</td>
<td>0.23</td>
<td>0.10</td>
</tr>
</tbody>
</table>

... emission matrix (B):

<table>
<thead>
<tr>
<th></th>
<th>dogs</th>
<th>bit</th>
<th>the</th>
<th>chased</th>
<th>a</th>
<th>these</th>
<th>cats</th>
</tr>
</thead>
<tbody>
<tr>
<td>DET</td>
<td>0.0</td>
<td>0.0</td>
<td>0.33</td>
<td>0.0</td>
<td>0.33</td>
<td>0.33</td>
<td>0.0</td>
</tr>
<tr>
<td>N</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.15</td>
</tr>
<tr>
<td>VB</td>
<td>0.1</td>
<td>0.6</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

... and initial probability matrix (π):

<table>
<thead>
<tr>
<th></th>
<th>DET</th>
<th>N</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DET</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VB</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using Example HMM

In order to generate words, we:
1. Choose tag/state from π
2. Choose emitted word from relevant row of B
3. Choose transition from relevant row of A
4. Repeat #2 & #3, until we hit a stopping point
   - Keeping track of probabilities as we go along

We could generate all possibilities this way and find the most probable sequence
- Want a more efficient way of finding most probable sequence

Motivating Hidden Markov Models

Thinking back to Markov models: we are now given a sequence of words and want to find the POS tags

- The underlying event of POS tags can be thought of as generating the words in the sentence
- Each state in the Markov model can be a POS tag
  - We don’t know the correct state sequence (Hidden Markov Model (HMM))

This requires an additional emission matrix, linking words with POS tags (cf. P(arrow|NN))