Part of Speech Tagging
Informatics 2A: Lecture 12

Bonnie Webber
School of Informatics
University of Edinburgh
bonnie@inf.ed.ac.uk
22 October 2009

Benefits of Part of Speech Tagging

- Can help in determining authorship: People’s use of words varies. Word frequency distributions can help determine if two documents were written by the same person ⇒ forensic linguistics.
- Can help in speech synthesis and recognition. For example, say the following out-loud:
  1. Have you read ‘The Wind in the Willows’? (noun)
  2. The clock has stopped. Please wind it up. (verb)
  3. The students tried to protest. (verb)
  4. The students are pleased that their protest was successful. (noun)

Corpus Annotation

Annotation: adds information that is not explicit in a corpus, increases its usefulness (often application-specific).
Even for English, corpus developers have felt it useful to distinguish a wide variety of Part-of-Speech (POS) classes.
These will then be distinguished in the tag set – the inventory of labels for marking up a text corpus – defined in a POS annotation scheme.

Example: part of speech tag sets
- CLAWS tag (used for BNC): 62 tags;
- Brown tag (used for Brown corpus): 87 tags;
- Penn tag set (used for the Penn Treebank): 45 tags.
Automatic POS Tagging

Corpus Annotation

Tags and Tokens

POS Tag Sets for English

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
<th>CLAWS</th>
<th>Brown</th>
<th>Penn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjective</td>
<td>happy, bad</td>
<td>AJ0</td>
<td>JJ</td>
<td>JJ</td>
</tr>
<tr>
<td>Noun singular</td>
<td>woman, book</td>
<td>NN1</td>
<td>NN</td>
<td>NN</td>
</tr>
<tr>
<td>Noun plural</td>
<td>women, books</td>
<td>NN2</td>
<td>NN</td>
<td>NN</td>
</tr>
<tr>
<td>Noun proper singular</td>
<td>London, Michael</td>
<td>NP0</td>
<td>NP</td>
<td>NNP</td>
</tr>
<tr>
<td>Noun proper plural</td>
<td>Finns, Hearts</td>
<td>NP0</td>
<td>NPS</td>
<td>NNPS</td>
</tr>
<tr>
<td>reflexive pro</td>
<td>itself, ourselves</td>
<td>PNX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plural reflexive pro</td>
<td>ourselves, ...</td>
<td>PPLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verb past participle</td>
<td>given, found</td>
<td>VVN</td>
<td>VBN</td>
<td>VBN</td>
</tr>
<tr>
<td>Verb base form</td>
<td>give, make</td>
<td>VVB</td>
<td>VB</td>
<td>VB</td>
</tr>
<tr>
<td>Verb simple past</td>
<td>ate, gave</td>
<td>VVD</td>
<td>VBD</td>
<td>VBD</td>
</tr>
</tbody>
</table>

All words must be assigned at least one tag. Differences in tags reflects what distinctions are/aren’t drawn.

Tags and Tokens

In NLTK corpus files, tokens and their POS-tags are usually given in the form `text/tag`:

```
Our/PRP\$ enemies/NNS are/VBP innovative/JJ and/CC resourceful/JJ ,/. and/CC so/RB are/VB we/PRP ./ . They/PRP never/RB stop/VB thinking/VBG about/IN new/JJ ways/NNS to/TO harm/VB our/PRP\$ country/NN and/CC our/PRP\$ people/NN, and/CC neither/DT do/VB we/PRP ./.
```

Imported into Python by NLTK, a token and its associated POS tag are represented using a Python tuple:

```
>>> tok = ('fly', 'nn')
>>> print tok[0]
fly
>>> print tok[1]
nn
```

This mapping is done automatically when tagged files are read in from `nltk.corpus`. For example, the Brown corpus:

```
>>> from nltk.corpus import brown
>>> print brown.tagged_sents('ca01')[0]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-TL'), ('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'), ('investigation', 'NN'), ('of', 'IN'), ('Atlanta’s', 'NP$'), ('recent', 'JJ'), ...
```

Extent of POS Ambiguity

POS-tagging a large corpus by hand is a lot of work. We’d prefer to automate POS-tagging if it could be done correctly.

Automatic tagging has problems arising from part-of-speech ambiguity and Zipf’s law (ie, “long tail” of infrequent words that may be unknown to the tagger).

### POS Ambiguity in the Brown corpus

In the Brown corpus (1M words: 500 written texts, different genres), there are 39440 different word types:

- 35340 have only 1 POS tag anywhere in corpus (89.6%)
- 4100 (10.4%) have 2–7 POS tags

Why does 10.4% POS-tag ambiguity by word type lead to difficulty?
### Extent of POS Ambiguity

- Recall that words in a large corpus have a Zipfian distribution.
- Word frequency is inversely proportional to word rank.
- Many high frequency words have more than one POS tag:
  - Rank 4: He wants to/TO go.
  - He went to/IN the store.
  - Rank 7: He wants that/DT hat.
  - It is obvious that/CS he wants a hat.
  - He wants a hat that/WPS fits.
- As a result, more than 40% of the word tokens are ambiguous.

### Taggers and Default Tagging

- Taggers differ with respect to:
  - what they know about a word;
  - how they decide to tag words they know;
  - what they decide to tag words they don’t know.
- A default tagger doesn’t know any words.
  - It knows what tag is the most common one in its manually tagged training corpus – eg, in the Brown Corpus:
    - NN 152470
    - JJ 64028
    - VB 33693
    - IN 120557
    - NNS 55110
    - VBN 29186
    - AT 97959
    - RB 36464
    - VBD 26167
  - That tag is assigned to each unknown token in the text.
- Default tagger accuracy on a text is equal to how frequently that tag is actually the correct one. (If the text is the Brown Corpus itself, accuracy = 152470/1000000 ≈ 15%.)

### Rule-based Tagging

Basic idea:
1. Assign each token all its possible tags.
2. Apply rules that eliminate all tags for a token that are inconsistent with its context.

**Example**

<table>
<thead>
<tr>
<th>the</th>
<th>DT (determiner)</th>
<th>the</th>
<th>DT (determiner)</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>can</td>
<td>MD (modal)</td>
<td>can</td>
<td>MD (modal)</td>
<td>X</td>
</tr>
<tr>
<td>NN</td>
<td>(sg noun)</td>
<td>NN</td>
<td>(sg noun)</td>
<td>√</td>
</tr>
<tr>
<td>VB</td>
<td>(base verb)</td>
<td>VB</td>
<td>(base verb)</td>
<td>X</td>
</tr>
</tbody>
</table>

Assign any unknown word tokens a tag that is consistent with its context (eg, the most frequent tag).

**Problem:** Cannot eliminate all POS ambiguity.

### Statistical Tagging: Unigram

- A Unigram tagger knows the most frequent tag for each word in its training corpus.
  1. Take a manually tagged corpus for training.
  2. For each word type, record the frequency of each tag it has been assigned.
  3. Given a new text, label each word with its most frequent tag from the training corpus.
  4. Words in the new text not found in the training corpus (unknown words) get no tag.
### Statistical Tagging: Unigram

```python
>>> from nltk.corpus import brown
>>> unigram_tagger = UnigramTagger(brown.tagged_sents(['mystery']))
```

```python
>>> t1 = 'Who said the jury took 49 days to reach a verdict'.split()
>>> unigram_tagger.tag(t1)
[('Who', 'WPS'), ('said', 'VBD'), ('the', 'AT'), ('jury', 'NN'),
 ('took', 'VBD'), ('49', 'None'), ('days', 'NNS'), ('to', 'TO'),
 ('reach', 'VB'), ('a', 'AT'), ('verdict', 'None')]
```

```python
>>> t2 = 'Change it from one to two'.split()
>>> unigram_tagger.tag(t2)
[('Change', 'None'), ('it', 'PP0'), ('from', 'IN'), ('one', 'CD'),
 ('to', 'TO'), ('two', 'CD'), ('.', '.')]
```

Problems with unigram tagging?

### Bigram Tagging

Bigram frequency has the potential to improve tagging accuracy by considering both the preceding word and its PoS when tagging the current word.

Basic idea: Choose the tag \( t_i \) for word \( w_i \) that maximizes the probability of \( t_i \) given the previous word tag \( t_{i-1} \) and \( w_i \).

\[
t_i = \arg \max_j P(t_j|t_{i-1}, w_i)
\]

Bigram tagging chooses the most probable sequence of tags, considering two-token sequences.

NLTK: `BigramTagger`

**Problem:** Need a lot of data for training before one can reap benefit. Without it, most words get tagged None.

### Transformation-based Tagging

**Basic idea:** combine features of rule-based and statistical methods:

1. Label each word with its most frequent tag from a training corpus (i.e., unigram tagging).
2. Apply context-sensitive transformational rules that change the most frequent tag to one that most improves labeling with respect to a manually tagged "gold standard".
3. Apply the combination of unigram tagging and these transformational rules in sequence to new text.

**Example:** assume the following unigram probabilities:

\[
P(\text{NN}|\text{race}) = .98 \quad P(\text{VB}|\text{race}) = .02
\]

Tag the sentence *the human race is expected to race tomorrow*:

**Rule:** Change NN to VB when previous tag is TO. This yields:

\[
\text{the/DT human/NN race/NN is/VBN expected/TO to/TO race/NN tomorrow/NN}
\]

\[
\text{the/DT human/NN race/NN is/VBN expected/TO to/TO race/VB tomorrow/NN}
\]
Unknown Words

Since the distribution of words is Zipfian, there are likely to be words in a new text not seen in the training corpus. What to do?

Recall that sometimes the Default tagger is correct. Since most “new” (ie, previously unseen) words are nouns, default to “noun” whenever word is unknown.

```python
>>> from nltk import DefaultTagger
>>> tokens = 'John saw 49 Siberian oryxes'.split()
>>> print(tokens)
['John', 'saw', '49', 'Siberian', 'oryxes']
>>> my_tagger = DefaultTagger('NN')
>>> my_tagger.tag(tokens)
[('John', 'NN'), ('saw', 'NN'), ('49', 'NN'), ('Siberian', 'NN'),
 ('oryxes', 'NN')]
```

This isn’t particularly accurate.

We can also use one of these unknown word taggers as a backoff strategy for a separate tagger (eg, a unigram tagger) for known words.

```python
>>> from nltk.corpora import brown
>>> from nltk import UnigramTagger, RegExpTagger
>>> mysteries = brown.fileids(['mystery'])
>>> unigram_tagger1 = UnigramTagger(brown.tagged_sents(mysteries))
>>> tokens = 'John saw 49 Siberian oryxes'.split()
>>> unigram_tagger1.tag(tokens)
[('John', 'None'), ('saw', 'VBD'), ('49', 'None'), ('Siberian', None),
 ('oryxes', None)]
```

Backoff says to use the other tagger if the word isn’t known to the first tagger.

Here is our tagger for unknown words – guessing them to be numbers or plural nouns, if they match the given patterns.

```python
>>> patterns2 = [(r'^\d{1,2}(\d{1,2})\d+$', 'cd'),
 (r'.*\d+$', 'nns'), (r'.*', 'nn')]
>>> p2_tagger = RegExpTagger(patterns2)
>>> unigram_tagger1 = UnigramTagger(brown.tagged_sents(mysteries),
 backoff=p2_tagger)
>>> unigram_tagger1.tag(tokens)
[('John', 'NN'), ('saw', 'VBD'), ('49', 'cd'), ('Siberian', 'NN'),
 ('oryxes', 'nns')]
```

Recall from Lecture 11, that formal criteria (i.e., the internal structure of a token) can be used to recognize the PoS of an unknown token.

This can be implemented using NLTK’s regular expression tagger:
Summary

- A number of POS tag sets exist for English (e.g. Brown, CLAWS, Penn).
- Automatic POS tagging makes errors because many high frequency words are part-of-speech ambiguous.
- Rule-based tagging assigns a word all possible tags and the uses context rules to disambiguate.
- Statistical tagging assigns a word its most likely tag, based on the unigram or bigram frequencies in a training corpus.
- Transformation-based tagging combines the two approaches.
- Unknown words can be handled by assigning them a default POS or by looking at the word’s internal structure.
- Current taggers, applied to texts similar to those on which they’ve been trained, reach high levels of accuracy.