Part of Speech Tagging
Informatics 2A: Lecture 13

Bonnie Webber (revised by Frank Keller)

School of Informatics
University of Edinburgh
bonnie@inf.ed.ac.uk

16 October 2007

Benefits of Part of Speech Tagging

Can be used to succinctly characterise the context in which a word is found in spoken or written text. E.g., in the Brown Corpus, the adverb \texttt{often} precedes:

\begin{tabular}{|c|c|c|}
\hline
PoS & Example & Freq \\
\hline
verb: past participle & he had \texttt{often} gone & 61 \\
verb: base form & they \texttt{often} make & 51 \\
verb: simple past & they \texttt{often} saw & 36 \\
adjective & it is \texttt{often} dangerous to & 30 \\
\ldots & & \\
\hline
\end{tabular}

This can help in recognizing similarities and differences between words. E.g., do all adverbs pattern like \texttt{often}?
Part of Speech Tagging
Automatic POS Tagging

Benefits
Corpus Annotation
Tags and Tokens

Corpus Annotation

**Annotation:** adds information that is not explicit in a corpus, increases its usefulness (often application-specific).

**PoS annotation scheme** consists of a tag set and annotation guidelines.

**Tag set:** an inventory of labels for marking up a text corpus

**Annotation guidelines** tell annotators (domain experts) how tag set is to be applied; ensure consistency across different annotators.

**Example: part of speech tag sets**
1. CLAWS tag (used for BNC); 62 tags;
2. Brown tag (used for Brown corpus); 87 tags;
3. Penn tag set (used for the Penn Treebank); 45 tags.

**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>Determiner</td>
<td>this, each</td>
<td>DT0</td>
<td>DT</td>
<td>DT</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>NP1</td>
<td>NPS</td>
<td>NNPS</td>
</tr>
<tr>
<td>reflexive pro</td>
<td>it, 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>

In NLTK, a token and its associated POS tag are represented using a Python tuple:

```python
>>> tok = ('fly', 'nn')
>>> print tok[0]
fly
>>> print tok[1]
nn
In files, tagged tokens are usually given in the form text/tag:

```
Our/PRP\$ enemies/NNS are/VBP innovative/JJ and/CC resourceful/JJ
/, and/CC so/RE are/VB ve/PRP ././. They/PRP never/RE 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 ./ ./.
```

The NLTK function tag2tuple maps text/tag pairs into Python tuples; tokenize provides simple tokenization:

```python
>>> from nltk.tag import tokenize, tag2tuple
data:
"""
... John/nn saw/vb the/at book/nn on/in the/at table/nn ./end
... He/nn sighed/vb ./end
"""

>>> for t in tokenize.whitespace(sent):
...    print tag2tuple(t),
...    ('John', 'nn') ('saw', 'vb') ('the', 'at') ('book', 'nn')
...    ('on', 'in') ('the', 'at') ('table', 'nn') ('.', 'end')
...    ('He', 'nn') ('sighed', 'vb') ('.', 'end')
```
Part of Speech Tagging
Automatic POS Tagging

Benefits
Corpus Annotation
Tags and Tokens

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

```python
>>> from nltk.corpus import brown
>>> print brown.tagged('a')
[('The', 'at'), ('Fulton', 'np-tl'), ('County', 'nn-tl'),
('Grand', 'jj-tl'), ('Jury', 'nn-tl'), ('said', 'vbd'),
('Friday', 'nr'), ('an', 'at'), ('investigation', 'nn'),
('in', 'in'), ('Atlanta’s', 'np$'), ('recent', 'jj'),
...]
```

POS tagging a large corpus by hand is a lot of work. Automatic taggers assign the correct word class label to each token in a text. Automatic tagging is difficult because of part-of-speech ambiguity.

**Example**

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

**But** the most frequent words have more than one POS tag, so more than 40% of the tokens are ambiguous.

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

For an unknown token, assign it a tag that is consistent with its context (e.g., the most frequent tag).

Statistical Tagging

Basic idea: Assign each token its most common tag:

1. Take a manually tagged corpus.
2. For each word type, record the frequency of each tag it has been assigned.
3. Label each word in a new text with its most frequent tag from the tagged corpus.

This approach uses unigram frequency, i.e., the frequency of word-tag pairs for individual words (no context).

NLTK: `tag.unigram` class (see chapter 4 of NLTK book)

```python
train() method
tag() method
```
Statistical Tagging – Unigram

```
>>> from nltk import tokenize, tag
>>> from nltk.corpus import brown
>>> train_sents = brown.tagged('b')
>>> unigram_tagger = tag.Unigram()
>>> unigram_tagger.train(train_sents)
>>> text = "the human race is expected to race tomorrow"
>>> tokens = list(tokenize.whitespace(text))
>>> list(unigram_tagger.tag(tokens))
[('the', 'at'), ('human', 'jj'), ('race', 'nn'), ('is', 'bez'),
 ('expected', 'vbn'), ('to', 'to'), ('race', 'nn'), ('tomorrow', 'None')]
```

Do you see any problem here with unigram tagging?

Statistical Tagging – Bigram

```
>>> from nltk import tokenize, tag
>>> from nltk.corpus import brown
>>> bigram_tagger = tag.Bigram()
>>> bigram_tagger.train(train_sents)
>>> text = "the human race is expected to race tomorrow"
>>> tokens = list(tokenize.whitespace(text))
>>> list(bigram_tagger.tag(tokens))
[('the', None), ('human', None), ('race', None), ('is', None),
 ('expected', None), ('to', 'in-hl'), ('race', None), ('tomorrow', None)]
```

Problem: Need more data to train on before one can reap benefit from bigram context.

Transformation-based Tagging

```
Basic idea: combine features of both 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.
```
Transformation-based Tagging

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:

the/DT human/NN race/NN is/VBZ expected/VBN to/TO race/NN tomorrow/NN

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

the/DT human/NN race/NN is/VBZ expected/VBN to/TO race/VB tomorrow/NN

Unknown Words

In every new text to be tagged, there will be words that don’t appear in the training corpus. What to do?

Most really new words are nouns: guess “noun” whenever word is unknown.

```python
>>> from nltk_lite import tokenize, tag
>>> text = "John saw 3 Trans-Dnieprian oryxes."
>>> tokens = list(tokenize.whitespace(text))
>>> print tokens
['John', 'saw', '3', 'Trans-Dnieprian', 'oryxes', ' .']
```

```python
>>> unigram_tagger1=tag.Unigram()
>>> unigram_tagger1.train(train_sents)
>>> list(unigram_tagger1.tag(tokens))
[('John', 'np-tl'), ('saw', 'vbd'), ('3', 'cd'), ('Trans-Dnieprian', None), ('oryxes', None)]
```

Use one of these unknown word taggers as a backoff strategy for (for example) a unigram tagger for known words:

```python
>>> from nltk_lite.corpora import brown
>>> from nltk_lite import tokenize, tag
>>> text = "John saw 3 Trans-Dnieprian oryxes."
>>> tokens = list(tokenize.whitespace(text))
>>> unigram_tagger1.train(train_sents)
>>> unigram_tagger1.tag(tokens)
[('John', 'np-tl'), ('saw', 'vbd'), ('3', 'cd'), ('Trans-Dnieprian', None), ('oryxes', None)]
```
A number of POS tag sets exist for English (e.g. Brown, CLAWS, Penn).

Automatic POS tagging is difficult because many highly frequent words are POS ambiguous.

Rule-based tagging assigns a word all possible tags and 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.