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

Readings:

J&M (1st ed), ch. 8 (pp. 298–321)
or *J&M (2nd ed), ch. 5 (pp.11-42);*
NLTK Book:
ch. 3 on words (http://nltk.org/doc/en/words.html),
ch. 4 on tagging (http://nltk.org/doc/en/tag.html)

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 often precedes:

PoS	Example	Freq
verb: past participle	he had often gone ...	61
verb: base form	they often make ...	51
verb: simple past	they often saw ...	36
adjective	it is often dangerous to ...	30
...		

This can help in recognizing similarities and differences between words. Eg, do all adverbs pattern like often?

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.

Can help in speech synthesis and recognition. For example, say the following **out-loud**:

1. *Have you read 'The Wind in the Willows'?*
2. *The clock has stopped. Please wind it up.*
3. *The students tried to protest.*
4. *The students are pleased that their protest was successful.*

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

Category	Examples	CLAWS	Brown	Penn
Adjective	happy, bad	AJ0	JJ	JJ
Determiner	this, each	DT0	DT	DT
Noun singular	woman, book	NN1	NN	NN
Noun plural	women, books	NN2	NN	NN
Noun proper singular	London, Michael	NP0	NP	NNP
Noun proper plural	Finns, Hearts	NP0	NPS	NNPS
reflexive pro	itself, ourselves	PNX		
plural reflexive pro	ourselves, . . .		PPLS	
Verb past participle	given, found	VVN	VBN	VBN
Verb base form	give, make	VVB	VB	VB
Verb simple past	ate, gave	VVD	VBD	VBD

Tags and Tokens

In 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
```

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/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 ./.
```


Tags and Tokens

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

```
>>> from nltk.tag import tokenize, tag2tuple
>>> sent = """
... 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')
```

Tags and Tokens

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('a')[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'),
...]
```


Automatic POS Tagging

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

the	DT (determiner)		the	DT (determiner)	
can	MD (modal)	⇒	can	MD (modal)	X
	NN (sg noun)			NN (sg noun)	√
	VB (base verb)			VB (base verb)	X

For an unknown token, assign it a tag that is consistent with its context (eg, 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)

```
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 frequency can improve tagging accuracy by considering the PoS of the preceding word 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 tag of the previous word 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: `tag.bigram` class:

```
train() method
tag() method
```


Statistical Tagging – Bigram

```
>>> from nltk import tokenize, tag
>>> from nltk.corpus import brown
>>> train_sents = brown.tagged('b')
>>> 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:

$$P(NN|race) = .98 \quad P(VB|race) = .02$$

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.

```
>>> 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', '.']
>>>
>>> my_tagger = tag.Default('nn')
>>> list(my_tagger.tag(tokens))
[('John', 'nn'), (saw', 'nn'), ('3', 'nn'),
(Trans-Dnieprian, 'nn'), ('oryxes, 'nn'), ('.', 'nn')]
```


Unknown Words

Recall from Lecture 12, 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:

```
>>> patterns = [(r'^[0-9]+(.[0-9]+)?$', 'cd'), (r'.*', 'nn')]
>>> nn_cd_tagger = tag.Regexp(patterns)
>>> list(nn_cd_tagger.tag(tokens))
[('John', 'nn'), ('saw', 'nn'), ('3', 'cd'), ('Trans-Dnieprian',
'nn'), ('oryxes', 'nn'), ('.', 'nn')]
>>> patterns = [(r'^[0-9]+(.[0-9]+)?$', 'cd'), (r'.*s$', 'nns'),
(r'.*', 'nn')]
>>> nns_cd_tagger = tag.Regexp(patterns)
>>> list(nns_cd_tagger.tag(tokens))
[('John', 'nn'), ('saw', 'nn'), ('3', 'cd'),
('Trans-Dnieprian', 'nn'), ('oryxes', 'nns')]
```


Unknown Words

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

```
>>> from nltk_lite.corpora import brown
>>> from nltk_lite import tokenize, tag
>>> text = "John saw 3 Trans-Dnieprian oryxes"
>>> tokens = list(tokenize.whitespace(text))
>>> train_sents = brown.tagged('d')
>>>
>>> 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)]
```

Unknown Words

```
>>> patterns = [(r'^[0-9]+(.[0-9]+)?$', 'cd'), (r'.*s$', 'nns'),
(r'.*', 'nn')]
>>> nns_cd_tagger=tag.Regexp(patterns)
>>>
>>> unigram_tagger2=tag.Unigram(backoff=nns_cd_tagger)
>>> unigram_tagger2.train(train_sents)
>>> list(unigram_tagger2.tag(tokens))
[('John', 'np'), ('saw', 'vbd'), ('3', 'cd'),
('Trans-Dnieprian', 'nn'), ('oryxes', 'nns')]
```


Summary

- 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 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.