

Part of Speech Tagging

Informatics 2A: Lecture 12

Bonnie Webber

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

22 October 2009

- Corpus Annotation
 - Tags and Tokens
- 1 Automatic POS Tagging
 - Default Tagging
 - Rule-based Tagging
 - Statistical Tagging
 - Transformation-based Tagging
 - Unknown Words

Readings:

J&M (2nd ed) Ch. 5 (Sec 5.2–5.7);

NLTK Book: Chapter 5, Categorizing and Tagging Words

Benefits of Part of Speech Tagging

o 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**.

o 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

- 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
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

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/MNS 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/MNS 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
```

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_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 **to**

*He wants to/TO go.
He went to/IN the store.*

Rank 7 **that**

*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.

Rule-based Tagging

Basic idea:

- Assign each token all its possible tags.
- 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

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

Problem: Cannot eliminate all POS ambiguity.

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	VTN	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 = $152470/1000000 \approx 15\%$.)

Statistical Tagging: Unigram

A **Unigram tagger** knows the most frequent tag for each word in its training corpus.

- Take a manually tagged corpus for training.
- For each word type, record the frequency of each tag it has been assigned.
- Given a new text, label each word with its most frequent tag from the training corpus.
- Words in the new text not found in the training corpus (**unknown words**) get no tag.

Statistical Tagging: Unigram

```
>>> from nltk.corpus import brown
>>> from nltk import UnigramTagger
>>> mysteries = brown.fileids(['mystery'])
>>> uni_tagger = UnigramTagger(brown.tagged_sents(mysteries))
>>> t1 = 'Who said the jury took 49 days to reach a verdict'.split()
>>> uni_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')]
>>> 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?

Statistical 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 .

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

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:

- Label each word with its most frequent tag from a training corpus (i.e., unigram tagging)
- 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".
- 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

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.

```
>>> 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.

Unknown Words

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:

```
>>> from nltk import RegexpTagger
>>> patterns1 = [(r'[0-9]+(?:[0-9]+)?$', 'cd'), (r'.*', 'nn')]
>>> p1_tagger = RegexpTagger(patterns)
>>> p1_tagger.tag(tokens)
[('John', 'nn'), ('saw', 'nn'), ('49', 'cd'), ('Siberian', 'nn'), ('oryxes', 'nn')]
>>> patterns2 = [(r'[0-9]+(?:[0-9]+)?$', 'cd'), (r'.*s$', 'nns'), (r'.*', 'nn')]
>>> p2_tagger = RegexpTagger(patterns2)
>>> p2_tagger.tag(tokens)
[('John', 'nn'), ('saw', 'nn'), ('49', 'cd'), ('Siberian', 'nn'), ('oryxes', 'nns')]
```

Unknown Words

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.

```
>>> 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')]
```

This is our tagger for known words.

Unknown Words

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

```
>>> patterns2 = [(r'[0-9]+(?:[0-9]+)?$', 'cd'), (r'.*s$', 'nns'), (r'.*', 'nn')]
>>> p2_tagger=RegexpTagger(patterns)
>>> mysteries = brown.fileids(['mystery'])
>>> ub=UnigramTagger(brown.tagged_sents(mysteries), backoff=p2_tagger)
>>> ub.tag(tokens)
[('John', 'nn'), ('saw', 'VBD'), ('49', 'cd'), ('Siberian', 'nn'), ('oryxes', 'nns')]
```

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