Tagging

Steven Bird Ewan Klein Edward Loper

University of Melbourne, AUSTRALIA

University of Edinburgh, UK

University of Pennsylvania, USA

May 16, 2007

Parts of speech

- How can we predict the bahaviour of a previously unseen word?
- Words can be divided into classes that behave similarly.
- Traditionally eight parts of speech: noun, verb, pronoun, preposition, adverb, conjunction, adjective and article.
- More recently larger sets have been used: eg Penn Treebank (45 tags), Susanne (353 tags).

Parts of Speech

What use are parts of speech?

They tell us a lot about a word (and the words near it).

- Tell us what words are likely to occur in the neighbourhood (eg adjectives often followed by nouns, personal pronouns often followed by verbs, possessive pronouns by nouns)
- Pronunciations can be dependent on part of speech, eg object, content, discount (useful for speech synthesis and speech recognition)
- Can help information retrieval and extraction (stemming, partial parsing)
- Useful component in many NLP systems

Closed and open classes

- Parts of speech may be categorised as *open* or *closed* classes
- Closed classes have a fixed membership of words (more or less), eg determiners, pronouns, prepositions
- Closed class words are usually *function words* — frequently occurring, grammatically important, often short (eg of,it,the,in)
- The major open classes are *nouns*, *verbs*, *adjectives* and *adverbs*

Closed classes in English

prepositions	on, under, over, to, with, by
determiners	the, a, an, some
pronouns	she, you, I, who
conjunctions	and, but, or, as, when, if
auxiliary verbs	can, may, are
particles	up, down, at, by
numerals	one, two, first, second

Open classes

nouns Proper nouns (Scotland, BBC), common nouns:

- count nouns (goat, glass)
- mass nouns (snow, pacifism)

verbs actions and processes (run, hope), also auxiliary verbs

adjectives properties and qualities (age, colour, value)

adverbs modify verbs, or verb phrases, or other adverbs: *Unfortunately* John walked *home extremely slowly yesterday*

The Penn Treebank tagset (1)

CC	Coord Conjuncn	*and,but,or*	NN	Noun, sing. or mass	*dog*
CD	Cardinal number	*one,two*	NNS	Noun, plural	*dogs*
DT	Determiner	*the,some*	NNP	Proper noun, sing.	*Edinburgh*
EX	Existential there	*there*	NNPS	Proper noun, plural	*Orkneys*
FW	Foreign Word	*mon dieu*	PDT	Predeterminer	*all, both*
IN	Preposition	*of,in,by*	POS	Possessive ending	*'s*
JJ	Adjective	*big*	PP	Personal pronoun	*I,you,she*
JJR	Adj., comparative	*bigger*	PP$	Possessive pronoun	*my,one's*
JJS	Adj., superlative	*biggest*	RB	Adverb	*quickly*
LS	List item marker	*1,One*	RBR	Adverb, comparative	*faster*
MD	Modal	*can,should*	RBS	Adverb, superlative	*fastest*

The Penn Treebank tagset (2)

RP	Particle	*up,off*	WP$	Possessive-Wh	*whose*
SYM	Symbol	*+,%,&*	WRB	Wh-adverb	*how,where*
TO	"to"	*to*	$	Dollar sign	*$*
UH	Interjection	*oh, oops*	#	Pound sign	*#*
VB	verb, base form	*eat*	"	Left quote	‘ , “
VBD	verb, past tense	*ate*	"	Right quote	’ , ”
VBG	verb, gerund	*eating*	(	Left paren	(
VBN	verb, past part	*eaten*	)	Right paren	)
VBP	Verb, non-3sg, pres	*eat*	,	Comma	,
VBZ	Verb, 3sg, pres	*eats*	.	Sent-final punct	. ! ?
WDT	Wh-determiner	*which,that*	:	Mid-sent punct.	: ; — ...
WP	Wh-pronoun	*what,who*			

Tagging

- Definition: POS Tagging is the assignment of a single part-of-speech tag to each word (and punctuation marker) in a corpus. For example:

  "/" The/DT guys/NNS that/WDT make/VBP traditional/JJ hardware/NN are/VBP really/RB being/VBG obsoleted/VBN by/IN microprocessor-based/JJ machines/NNS ,/, "/" said/VBD Mr./NNP Benton/NNP ./.
- Non-trivial: POS tagging must resolve ambiguities since the same word can have different tags in different contexts
- In the Brown corpus 11.5% of word types and 40% of word tokens are ambiguous
- In many cases one tag is much more likely for a given word than any other
- Limited scope: only supplying a tag for each word, no larger structures created (eg prepositional phrase attachment)

Information sources for tagging

What information can help decide the correct PoS tag for a word?

Other PoS tags Even though the PoS tags of other words may be uncertain too, we can use information that some tag sequences are more likely than others (eg *the/AT red/JJ drink/NN* vs *the/AT red/JJ drink/VBP*).
Using *only* information about the most likely PoS tag sequence does not result in an accurate tagger (about 77% correct)

The word identity Many words can gave multiple possible tags, but some are more likely than others (eg *fall/VBP* vs *fall/NN*)
Tagging each word with its most common tag results in a tagger with about 90% accuracy

Tagging in NLTK

The simplest possible tagger tags everything as a noun:

```
from nltk_lite import tokenize
text = 'There are 11 players in a football team'
text_tokens = list(tokenize.whitespace(text))
# ['There', 'are', '11', 'players', 'in', 'a', 'football', '

from nltk_lite import tag
mytagger = tag.Default('nn')
for t in mytagger.tag(text_tokens):
    print t
# ('There', 'NN')
# ('are', 'NN')
# ...
```

A regular expression tagger

We can use regular expressions to tag tokens based on regularities in the text, eg numerals:

```
default_pattern = (r'.*', 'NN')
cd_pattern = (r' ^[0-9]+(.[0-9]+)?$', 'CD')
patterns = [cd_pattern, default_pattern]
NN_CD_tagger = tag.Regexp(patterns)
re_tagged = list(NN_CD_tagger.tag(text_tokens))
# [('There', 'NN'), ('are', 'NN'), ('11', 'NN'), ('players
('in', 'NN'), ('a', 'NN'), ('football', 'NN'), ('team', 'N
```

A unigram tagger

The NLTK UnigramTagger class implements a tagging algorithm based on a table of unigram probabilities:

$$\text{tag}(w) = \arg\max_{t_i} P(t_i|w)$$

Training a UnigramTagger on the Penn Treebank:

```
from nltk_lite.corpora import treebank
from itertools import islice

# sentences 0-2999
train_sents = list(islice(treebank.tagged(), 3000))
# from sentence 3000 to the end
test_sents = list(islice(treebank.tagged(), 3000, None))

unigram_tagger = tag.Unigram()
unigram_tagger.train(train_sents)
```

Unigram tagging

```
>>> list(unigram_tagger.tag(tokenize.whitespace("Mr. Jones
the book on the shelf")))
[('Mr.', 'NNP'), ('Jones', 'NNP'), ('saw', 'VBD'), ('the',
('book', 'NN'), ('on', 'IN'), ('the', 'DT'), ('shelf', Non
```

The UnigramTagger assigns the default tag `None` to words that are not in the training data (eg *shelf*)
We can combine taggers to ensure every word is tagged:

```
>>> unigram_tagger = tag.Unigram(cutoff=0,backoff=NN_CD_ta
>>> unigram_tagger.train(train_sents)
>>> list(unigram_tagger.tag(tokenize.whitespace("Mr. Jones
the book on the shelf")))
[('Mr.', 'NNP'), ('Jones', 'NNP'), ('saw', 'VBD'), ('the',
('book', 'VB'), ('on', 'IN'), ('the', 'DT'), ('shelf', 'NN
```

Evaluating taggers

- Basic idea: compare the output of a tagger with a human-labelled *gold standard*
- Need to compare how well an automatic method does with the agreement between people
- The best automatic methods have an accuracy of about 96-97% when using the (small) Penn treebank tagset (but this is still an average of one error every couple of sentences...)
- Inter-annotator agreement is also only about 97%
- A good unigram baseline (with smoothing) can obtain 90-91%!

Evaluating taggers in NLTK

NLTK provides a function `tag.accuracy` to automate evaluation. It needs to be provided with a tagger, together with some text to be tagged and the gold standard tags.
We can make print more prettily:

```
def print_accuracy(tagger, data):
    print '%3.1f%%' % (100 * tag.accuracy(tagger, dat

>>> print_accuracy(NN_CD_tagger, test_sents)
15.0%
>>> print_accuracy(unigram_tagger, train_sents)
93.8%
>>> print_accuracy(unigram_tagger, test_sents)
82.8%
```

Error analysis

- The % correct score doesn't tell you everything — it is useful know what is misclassified as what
- *Confusion matrix*: A matrix (ntags x ntags) where the rows correspond to the correct tags and the columns correspond to the tagger output. Cell (i, j) gives the count of the number of times tag i was classified as tag j
- The leading diagonal elements correspond to correct classifications
- Off diagonal elements correspond to misclassifications
- Thus a confusion matrix gives information on the major problems facing a tagger (eg NNP vs. NN vs. JJ)
- See section 3 of the NLTK tutorial on Tagging

N-gram taggers

- Basic idea: Choose the tag that maximises:

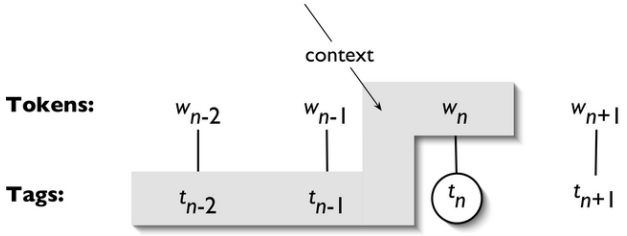
$$P(\text{word}|\text{tag}) \cdot P(\text{tag}|\text{previous n tags})$$

- For a bigram model the best tag at position i is:

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

  Assuming that you know the previous tag, t_{i-1}.
- Interpretation: choose the tag t_i that is most likely to generate word w_i given that the previous tag was t_{i-1}

N-gram taggers



Example (J+M, p304)

Secretariat/NNP is/VBZ expected/VBZ to/TO race/VB tomorrow/NN

People/NNS continue/VBP to/TO inquire/VB the/DT reason/NN for/IN the/DT race/NN for/IN outer/JJ space/NN

- "race" is a verb in the first, a noun in the second.
- Assume that race is the only untagged word, so we can assume the tags of the others.
- Probabilities of "race" being a verb, or race being a noun in the first example:

$$P(\text{race is } VB) = P(VB|TO)P(\text{race}|VB)$$
$$P(\text{race is } NN) = P(NN|TO)P(\text{race}|NN)$$

Example (continued)

$$P(NN|TO) = 0.021$$
$$P(VB|TO) = 0.34$$

$$P(\text{race}|NN) = 0.00041$$
$$P(\text{race}|VB) = 0.00003$$

$$\begin{aligned} P(\text{race is } VB) &= P(VB|TO)P(\text{race}|VB) \\ &= 0.34 \times 0.00003 = 0.00001 \\ P(\text{race is } NN) &= P(NN|TO)P(\text{race}|NN) \\ &= 0.021 \times 0.00041 = 0.000007 \end{aligned}$$

Simple bigram tagging in NLTK

```
>>> default_pattern = (r'.*', 'NN')
>>> cd_pattern = (r' ^[0-9]+(.[0-9]+)?$', 'CD')
>>> patterns = [cd_pattern, default_pattern]
>>> NN_CD_tagger = tag.Regexp(patterns)
>>> unigram_tagger = tag.Unigram(cutoff=0, backoff=NN_CD_t
>>> unigram_tagger.train(train_sents)
>>> bigram_tagger = tag.Bigram(backoff=unigram_tagger)
>>> bigram_tagger.train(train_sents)

>>> print_accuracy(bigram_tagger, train_sents)
95.6%
>>> print_accuracy(bigram_tagger, test_sents)
84.2%
```

Limitation of NLTK n-gram taggers

- Does not find the most likely sequence of tags, simply works left to right always assigning the most probable single tag (given the previous tag assignments)
- Does not cope with zero probability problem well (no smoothing or discounting)
- see module `nltk_lite.tag.hmm`

Brill Tagger

- Problem with n-gram taggers: size
- A rule-based system...
- ...but the rules are learned from a corpus
- Basic approach: start by applying general rules, then successively refine with additional rules that correct the mistakes (painting analogy)
- Learn the rules from a corpus, using a set of rule templates, eg:
  Change tag **a** to **b** when the following word is tagged **z**
- Choose the best rule each iteration

Brill Tagger: Example

Sentence	**Gold**	**Unigram**	**Replace `nn` with `vb` when the previous word is `to`**	**Replace `to` with `in` when the next tag is `nns`**
The	at	at		
President	nn-tl	nn-tl		
said	vbd	vbd		
he	pps	pps		
will	md	md		
ask	vb	vb		
Congress	np	np		
to	to	to		
increase	vb	*nn*	*vb*	
grants	nns	nns		
to	in	*to*	*to*	*in*
states	nns	nns		
for	in	in		
vocational	jj	jj		
rehabilitation	nn	nn		

Summary

- **Reading:** Jurafsky and Martin, chapter 8 (esp. sec 8.5); Manning and Schütze, chapter 10;
- Rule-based and statistical tagging
- HMMs and n-grams for statistical tagging
- Operation of a simple bigram tagger
- TnT — an accurate trigram-based tagger