Lecture 10:
Part of Speech Tagging with NLTK

Methods in Computational Linguistic II

Weka Demo

- Recap of the ARFF file format.
- Using the weka explorer.
  - Classification
  - Feature Inspection
  - Feature Selection
  - Filters

Lexical Categories

- What's a lexical category?

Why do we use Lexical Categories in CL?

- Generalization of hypotheses
- Explanatory Power
- Simplification of processing

Parts of Speech

- How are lexical categories defined?

- Probably the most common Lexical Category is Parts of Speech

Parts of Speech

- Function Words -- words without inherent semantic meaning that serve mainly a syntactic function
    - articles, pronouns, adpositions, aux verbs, interjections, particles, expletives, pro-sentences
    - **Fun Fact**: English words that start with voiced /dh/ are function words.
- Content Words -- words with semantic meaning

Simplified Part of Speech Set

Tag	Meaning	Examples
ADJ	adjective	new, good, high, special, big, local
ADV	adverb	really, already, still, early, now
CNJ	conjunction	and, or, but, if, while, although
DET	determiner	the, a, some, most, every, no
EX	existential	there, there's
FW	foreign word	dolce, ersatz, esprit, quo, maitre
MOD	modal verb	will, can, would, may, must, should
N	noun	year, home, costs, time, education
NP	proper noun	Alison, Africa, April, Washington
NUM	number	twenty-four, fourth, 1991, 14:24
PRO	pronoun	he, their, her, its, my, I, us
P	preposition	on, of, at, with, by, into, under
TO	the word *to*	to
UH	interjection	ah, bang, ha, whee, hmpf, oops
V	verb	is, has, get, do, make, see, run
VD	past tense	said, took, told, made, asked
VG	present participle	making, going, playing, working
VN	past participle	given, taken, begun, sung
WH	*wh* determiner	who, which, when, what, where, how

Penn Treebank Tag set

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Advantages and Disadvantages of Tagsets

- Why would you prefer one tag set over another?
- Are you limited to these tagsets?

Tagging in nltk

```
>>> text = nltk.word_tokenize("And now for something
completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'),
('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

- What is the input to nltk.pos_tag()?
- What is the output?

What if I forget what "JJ" means?

- NLTK includes documentation:
  - `nltk.help.upenn_tagset('RB')`
  - `nltk.help.brown_tagset('NN.*')`
  - Some corpora have specific documentation:
    - `nltk.corpus.???.readme()`

Tagging Homonyms

```
>>> text = nltk.word_tokenize("They refuse to permit us to
obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit',
'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse',
'NN'), ('permit', 'NN')]
```

- how does this work?

Data driven lexical classes

```
>>> text = nltk.Text(word.lower() for word in
nltk.corpus.brown.words())
>>> text.similar('woman')
Building word-context index...
man time day year car moment world family house country child boy
state job way war girl place room word
>>> text.similar('bought')
made said put done seen had found left given heard brought got been
was set told took in felt that
>>> text.similar('over')
in on to of and for with from at by that into as up out down through
is all about
>>> text.similar('the')
a his this their its her an that our any all one these my in your no
some other and
```

- how would you identify “similar” words?
- can we implement text.similar(w)?

Tagged corpora.

```
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ...]
>>> nltk.corpus.brown.tagged_words(simplify_tags=True)
[('The', 'DET'), ('Fulton', 'N'), ('County', 'N'), ...]


>>> print nltk.corpus.nps_chat.tagged_words()
[('now', 'RB'), ('im', 'PRP'), ('left', 'VBD'), ...]
>>> nltk.corpus.conll2000.tagged_words()
[('Confidence', 'NN'), ('in', 'IN'), ('the', 'DT'), ...]
>>> nltk.corpus.treebank.tagged_words()
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ...]
```

- simplify_tags=True guarantees compatible tags.
- There are also tagged corpora in non-English languages

Count Frequent Tags

```
>>> from nltk.corpus import brown
>>> brown_news_tagged =
brown.tagged_words(categories='news',
simplify_tags=True)
>>> tag_fd = nltk.FreqDist(tag for (word, tag) in
brown_news_tagged)
>>> tag_fd.keys()
['N', 'P', 'DET', 'NP', 'V', 'ADJ', ',', '.', 'CNJ',
'PRO', 'ADV', 'VD', ...]
```

- what's going on here?

Inspecting pairs of tags

```
>>> word_tag_pairs = nltk.bigrams(brown_news_tagged)
>>> list(nltk.FreqDist(a[1] for (a, b) in word_tag_pairs
if b[1] == 'N'))
['DET', 'ADJ', 'N', 'P', 'NP', 'NUM', 'V', 'PRO', 'CNJ',
'.', ',', 'VG', 'VN', ...]
```

- What can we tell with this?
- What does nltk.bigrams do?

How can you identify common verbs?

```
>>> wsj =
nltk.corpus.treebank.tagged_words(simplify_tags=True)
>>> word_tag_fd = nltk.FreqDist(wsj)
>>> [word + "/" + tag for (word, tag) in word_tag_fd
if tag.startswith('V')]
['is/V', 'said/VD', 'was/VD', 'are/V', 'be/V', 'has/
V', 'have/V', 'says/V',
'were/VD', 'had/VD', 'been/VN', "'s/V", 'do/V', 'say/
V', 'make/V', 'did/VD',
'rose/VD', 'does/V', 'expected/VN', 'buy/V', 'take/V',
'get/V', 'sell/V',
'help/V', 'added/VD', 'including/VG', 'according/VG',
'made/VN', 'pay/V', ...]
```

- `nltk.help.upenn_brown_tagset('V.*')`

Using a ConditionalFreqDist

```
>>> cfd2 = nltk.ConditionalFreqDist((tag, word) for
(word, tag) in wsj)
>>> cfd2['VN'].keys()
['been', 'expected', 'made', 'compared', 'based',
'priced', 'used', 'sold',
'named', 'designed', 'held', 'fined', 'taken', 'paid',
'traded', 'said', ...]
```

- What are the most common instances of a particular tag?

Finding instances of word/tag pairs

```
>>> [w for w in cfd1.conditions() if 'VD' in cfd1[w] and
'VN' in cfd1[w]]
['Asked', 'accelerated', 'accepted', 'accused',
'acquired', 'added', 'adopted', ...]
>>> idx1 = wsj.index(('kicked', 'VD'))
>>> wsj[idx1-4:idx1+1]
[('While', 'P'), ('program', 'N'), ('trades', 'N'),
('swiftly', 'ADV'),
('kicked', 'VD')]
>>> idx2 = wsj.index(('kicked', 'VN'))
>>> wsj[idx2-4:idx2+1]
[('head', 'N'), ('of', 'P'), ('state', 'N'), ('has',
'V'), ('kicked', 'VN')]
```

Searching tagged corpora

```
from nltk.corpus import brown

def process(sentence):
    for (w1,t1), (w2,t2), (w3,t3) in nltk.trigrams(sentence):
        if (t1.startswith('V') and w2 == 'for' and t3.startswith('V')):
            print w1, w2, w3
```

```
>>> for tagged_sent in brown.tagged_sents():
...     process(tagged_sent)
```

Finding ambiguity

```
>>> brown_news_tagged = brown.tagged_words(categories='news',
simplify_tags=True)
>>> data = nltk.ConditionalFreqDist((word.lower(), tag)
...                                 for (word, tag) in
brown_news_tagged)
>>> for word in data.conditions():
...     if len(data[word]) > 3:
...         tags = data[word].keys()
...         print word, ' '.join(tags)
...
```

- Aside from last time
- d1.update(d2) includes all of d2 into d1
- possibly useful for combining feature sets

Constructing an automatic tagger

- Default taggers.
  - Assign the same tag to every entity.

```
>>> from nltk.corpus import brown
>>> brown_tagged_sents =
brown.tagged_sents(categories='news')
>>> brown_sents = brown.sents(categories='news')

>>> tags = [tag for (word, tag) in
brown.tagged_words(categories='news')]
>>> nltk.FreqDist(tags).max()

>>> raw = 'I do not like green eggs and ham, I do not like them
Sam I am!'
>>> tokens = nltk.word_tokenize(raw)
>>> default_tagger = nltk.DefaultTagger('NN')
>>> default_tagger.tag(tokens)

>>> default_tagger.evaluate(brown_tagged_sents)
```

Regular expression tagging

- Write your own rules.

```
>>> patterns = [
...     (r'.*ing$', 'VBG'),                 # gerunds
...     (r'.*ed$', 'VBD'),                  # simple past
...     (r'.*es$', 'VBZ'),                  # 3rd singular present
...     (r'.*ould$', 'MD'),                 # modals
...     (r'.*\'s$', 'NN$'),                 # possessive nouns
...     (r'.*s$', 'NNS'),                   # plural nouns
...     (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),    # cardinal numbers
...     (r'.*', 'NN')                       # nouns (default)
... ]

>>> regexp_tagger = nltk.RegexpTagger(patterns)
>>> regexp_tagger.tag(brown_sents[3])
[('``', 'NN'), ('Only', 'NN'), ('a', 'NN'), ('relative', 'NN'), ('handful',
'NN'),
('of', 'NN'), ('such', 'NN'), ('reports', 'NNS'), ('was', 'NNS'), ('received',
'VBD'),
("''", 'NN'), (',', 'NN'), ('the', 'NN'), ('jury', 'NN'), ('said', 'NN'),
(',', 'NN'),
('``', 'NN'), ('considering', 'VBG'), ('the', 'NN'), ('widespread',
'NN'), ...]
>>> regexp_tagger.evaluate(brown_tagged_sents)
```

Lookup tagging

- Look up the most common tag for a word

```
>>> fd = nltk.FreqDist(brown.words(categories='news'))
>>> cfd =
nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
>>> most_freq_words = fd.keys()[:100]
>>> likely_tags = dict((word, cfd[word].max()) for word in
most_freq_words)
>>> baseline_tagger = nltk.UnigramTagger(model=likely_tags)
>>> baseline_tagger.evaluate(brown_tagged_sents)
```

```
>>> baseline_tagger = nltk.UnigramTagger(model=likely_tags,
backoff=nltk.DefaultTagger('NN'))
```

How many words are enough?

```
def performance(cfd, wordlist):
    lt = dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model=lt,
backoff=nltk.DefaultTagger('NN'))
    return
baseline_tagger.evaluate(brown.tagged_sents(categories='news'))

def display():
    import pylab
    words_by_freq =
list(nltk.FreqDist(brown.words(categories='news')))
    cfd =
nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
    sizes = 2 ** pylab.arange(15)
    perfs = [performance(cfd, words_by_freq[:size]) for size in
sizes]
    pylab.plot(sizes, perfs, '-bo')
    pylab.title('Lookup Tagger Performance with Varying Model
Size')
    pylab.xlabel('Model Size')
    pylab.ylabel('Performance')
    pylab.show()
```

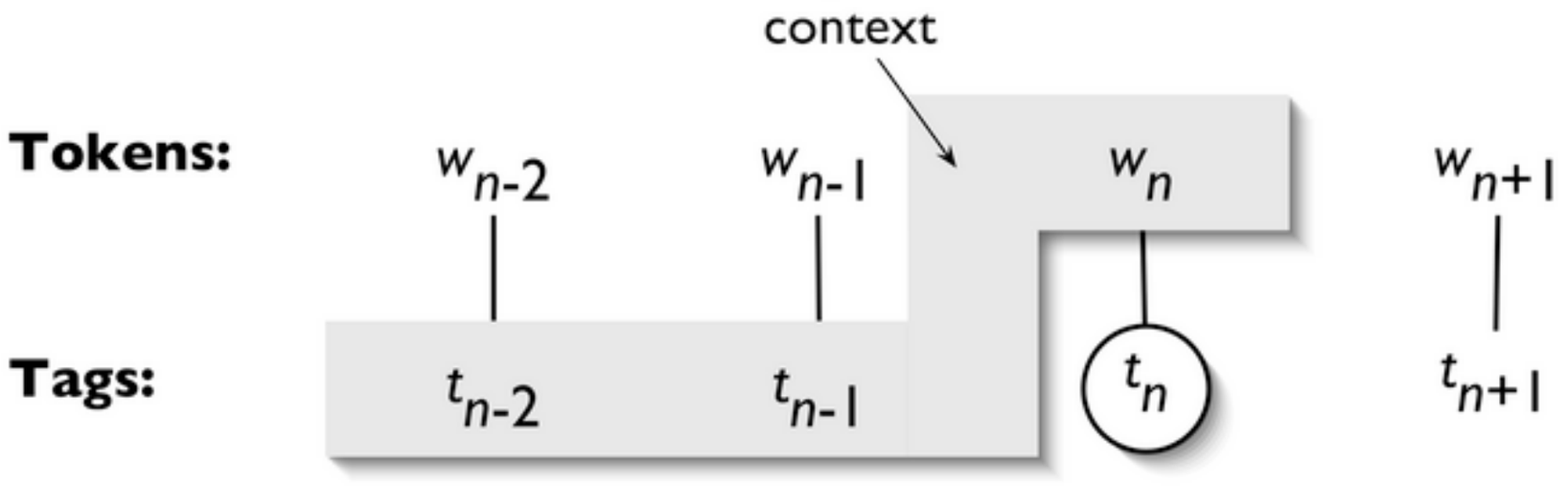
Why is this bad?

```
>>> from nltk.corpus import brown
>>> brown_tagged_sents = brown.tagged_sents(categories='news')
>>> brown_sents = brown.sents(categories='news')
>>> unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
>>> unigram_tagger.tag(brown_sents[2007])
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments',
'NNS'),
('are', 'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'),
('type', 'NN'),
(',', ','), ('being', 'BEG'), ('on', 'IN'), ('the', 'AT'),
('ground', 'NN'),
('floor', 'NN'), ('so', 'QL'), ('that', 'CS'), ('entrance', 'NN'),
('is', 'BEZ'),
('direct', 'JJ'), ('.', '.')]
>>> unigram_tagger.evaluate(brown_tagged_sents)
```

Better....

```
>>> size = int(len(brown_tagged_sents) * 0.9)
>>> size
4160
>>> train_sents = brown_tagged_sents[:size]
>>> test_sents = brown_tagged_sents[size:]
>>> unigram_tagger = nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
```

N-gram tagging




```
>>> size = int(len(brown_tagged_sents) * 0.9)
>>> size
4160
>>> train_sents = brown_tagged_sents[:size]
>>> test_sents = brown_tagged_sents[size:]
>>> bigram_tagger =
nltk.BigramTagger(train_sents)
>>> bigram_tagger.evaluate(test_sents)
```

Putting it all together

```
>>> t0 = nltk.DefaultTagger('NN')
>>> t1 = nltk.UnigramTagger(train_sents, backoff=t0)
>>> t2 = nltk.BigramTagger(train_sents, backoff=t1)
>>> t2.evaluate(test_sents)
```

- Backoff allows for robustness
- cutoff = n
- TrigramTagger(train_sents)
- NgramTagger(n, train_sents)

Tagging

- There is nothing special about POS tags in this demo.
- Any annotation can be trained and tagged with this NgramTagger class
- Chunking, Sentence boundaries, Named Entities

Brill Tagging

- N-gram modeling is based on the markov assumption
- Probabilistic Approach.
- An earlier approach is error-driven called the Brill Tagger

Brill Tagging

- Combines Rule-based and Stochastic Training
  - Rules specify tags
  - Errors in a corpus dictate the best rules
- Input:
  - Tagged corpus
  - Dictionary with most frequent tags

- Idea: Strip tags from the corpus and learn them from applying rules
  - 1. initialize with most probably tag for each word (unigram tagging)
  - 2. Change tags according to rewrite rules
    - "if w-1 is a determiner and w is a verb, then change tag to noun"
  - 3. Compare to gold standard
  - 4. Iterate

- Rules are created via templates of the form if w-1 is an X and w is a Y then change tag to Z, e.g.
  - Find the rule that works for most tags
  - Iterate on newly tagged corpus until a threshold is reached
  - Return an ordered set of rules
- Rules can introduce errors that are corrected later

Templates

The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One of the three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The preceding (following) word is tagged **z** and the word two before (after) is tagged **w**.

#	Change tags From	To	Condition	Example
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Sample

- Labels every word with its most-likely tag
  E.g. *race* occurences in the Brown corpus:

  P(NN|race) = .98
  P(VB|race)= .02
  is/VBZ expected/VBN to/TO race/NN tomorrow/NN

- Then TBL applies the following rule

"Change NN to VB when previous tag is TO"
... *is/VBZ expected/VBN to/TO race/NN tomorrow/NN*
becomes
... *is/VBZ expected/VBN to/TO race/VB tomorrow/NN*

Condensed algorithm

Step 1: Label every word with most likely tag (from dictionary)
Step 2: Check every possible transformation & select one which most improves tag accuracy (cf Gold)
Step 3: Re-tag corpus applying this rule, and add rule to end of rule set
Repeat 2-3 until some stopping criterion is reached, e.g., X% correct with respect to training corpus
RESULT: Ordered set of transformation rules to use on new data tagged only with most likely POS tags

- Issues:
  - No guarantees that the stopping criterion is reached
  - Rules are learned in order -- highly dependent
  - Rules may (and often do) interact
- Upside:
  - Rules are easy to understand.
- Demo: nltk.tag.brill.demo()