NLTK tagging

L435/L555
Dept. of Linguistics, Indiana University
Fall 2016
We can use NLTK to perform a variety of NLP tasks

- Today, we will quickly cover the utilities for POS tagging

- Other modules include:
  - Classification
  - Parsing, Chunking, & Grammar Writing
  - Propositional Semantics & Logic

Goal: make you comfortable learning more on your own
As we saw, you can use `nltk.word_tokenize()` to break a sentence into tokens

- `nltk.sent_tokenize` breaks a text into sentences

```python
nltk.sent_tokenize("Imagine me and you. I do. \nI think about you day and night.")
```

['Imagine me and you.',
 'I do.',
 'I think about you day and night.']
Basic NLTK tagging

A very basic way to tag:

```python
>>> import nltk
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')]```
Representing tagged tokens

NLTK uses tuples to represent word, tag pairs:

```python
>>> tagged_token = nltk.tag.str2tuple('fly/NN')
>>> tagged_token
('fly', 'NN')

>>> sent = 'They/PRP refuse/VBP to/TO permit/VB us/PRP to/TO obtain/VB the/DT refuse/NN permit/NN'

>>> [nltk.tag.str2tuple(t) for t in sent.split()]
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```
NLTK has tagged corpora to work with


```python
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ...

>>> nltk.corpus.brown.tagged_words(simplify_tags=True)
[('The', 'DET'), ('Fulton', 'NP'), ('County', 'N'), ...
```
Corpus reading options

Ways to access information for tagged corpora:

- `.words()`
  [list of words]
- `.tagged_words()`
  [list of (word,tag) pairs]
- `.sents()`
  [list of list of words]
- `.tagged_sents()`
  [list of list of (word,tag) pairs]
- `.paras()`
  [list of list of list of words]
- `.tagged_paras()`
  [list of list of list of (word,tag) pairs]
Calculating corpus statistics

```python
>>> 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', 'DET', 'P', 'NP', 'V', 'ADJ', ',', '.', 'CNJ', ...]
>>> tag_fd['N']
22226
```
>>> wsj = nltk.corpus.treebank.tagged_words(simplify_tags=True)
>>> cfd1 = nltk.ConditionalFreqDist(wsj)
>>> cfd1['cut'].keys()
['V', 'VD', 'N', 'VN']
>>> cfd1['cut']['V']
12
Automatic POS tagging
Most Frequent Tag Tagger

```python
>>> 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)
[('I', 'NN'), ('do', 'NN'), ('not', 'NN'), ...

>>> brown_tagged_sents = brown.tagged_sents(categories='news')
>>> default_tagger.evaluate(brown_tagged_sents)
0.13089484257215028
```
Automatic POS tagging

Regular Expression Tagger

```python
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)
```

Note that the patterns are applied *in order*
Automatic POS tagging
Regular Expression Tagger (2)

```python
>>> brown_sents = brown.sents(categories='news')
>>> regexp_tagger.tag(brown_sents[3])
[('“”, ’NN’), ... (’such’, ’NN’), (’reports’, ’NNS’),
  ... (’considering’, ’VBG’), (’the’, ’NN’), ...]

>>> regexp_tagger.evaluate(brown_tagged_sents)
0.20326391789486245
```
Automatic POS tagging

Lookup Tagger

Idea: use the most frequent tag for every word

```python
>>> 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.tag(brown.sents(categories='news')[3])
[('', '', ''), ('Only', None), ('a', 'AT'), ...]
>>> baseline_tagger.evaluate(brown_tagged_sents)
0.45578495136941344
```
Unigram tagging

```python
>>> unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
>>> unigram_tagger.tag(brown_sents[2007])
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ... ]
>>> unigram_tagger.evaluate(brown_tagged_sents)
0.9349006503968017
```
N-gram tagging
Training & Testing Data

```python
>>> 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)
0.8110236220472441
```
N-gram tagging

Bigram tagging

```python
>>> bigram_tagger = nltk.BigramTagger(train_sents)
>>> bigram_tagger.tag(brown_sents[2007])
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ...]
>>> unseen_sent = brown_sents[4203]
>>> bigram_tagger.tag(unseen_sent)
[('The', 'AT'), ('population', 'NN'), ('of', 'IN'), ...]
>>> bigram_tagger.evaluate(test_sents)
0.10216286255357321
```
N-gram tagging

Combining taggers

Use the best information if you have it:

```python
>>> t0 = nltk.DefaultTagger('NN')
>>> t1 = nltk.UnigramTagger(train_sents, backoff=t0)
>>> t2 = nltk.BigramTagger(train_sents, backoff=t1)
>>> t2.evaluate(test_sents)
0.8447124489185687
```

Unknown words can (also) be handled via regular expressions and be better integrated into contextual information
Exercises

10. Train a unigram tagger and run it on some new text. Observe that some words are not assigned a tag. Why not?

11. Learn about the affix tagger (type help(nltk.AffixTagger)). Train an affix tagger and run it on some new text. Experiment with different settings for the affix length and the minimum word length. Discuss your findings.