

Tagging with NLTK

Linguistics 420
Statistical NLP
Spring 2007

Tagging with NLTK

NLTK has a tag module which allows us to access a variety of simple POS taggers within python

The documentation can be found at:

- <http://nltk.sourceforge.net/lite/doc/en/tag.html>

We'll mostly focus on the syntax here ...

2

Representation

NLTK represents words and tags by using tuples (pairs) and provides a function `tag2tuple` to convert word/tag to this representation

```
>>> from nltk_lite.tag import tag2tuple
>>> ttoken = 'fly/nn'
>>> tag2tuple(ttoken)
('fly', 'nn')
```

3

Accessing Corpora

NLTK comes with tagged corpora installed, e.g., the Brown corpus

```
>>> from nltk_lite.corpora import brown, extract
>>> extract(6, brown.tagged('a'))
[('The', 'at'), ('grand', 'jj'), ('jury', 'nn'),
 ('commented', 'vbd'), ('on', 'in'), ('a', 'at'),
 ('number', 'nn'), ('of', 'in'), ('other', 'ap'),
 ('topics', 'nns'), ('', ''), ... ('.', '.')]

```

Can iterate over the corpus by looping over `brown.tagged()`

4

Default Tagger

A very simple tagger assigns every word the same tag; this is NLTK's default tagger

```
>>> from nltk_lite import tokenize
>>> from nltk_lite import tag
>>> tokens = tokenize.whitespace('John saw 3 polar bears .')
>>> default_tagger = tag.Default('nn')
>>> list(default_tagger.tag(tokens))
[('John', 'nn'), ('saw', 'nn'), ('3', 'nn'), ('polar', 'nn'),
 ('bears', 'nn'), ('.', 'nn')]
```

We can test the accuracy like so (the Brown corpus is divided into different sections):

```
>>> tag.accuracy(default_tagger, brown.tagged('a'))
0.13089484257215028
```

5

Regular Expression Tagger

The regular expression tagger allows us to associate particular tags with certain REs

```
>>> patterns = [
...     (r'.*ing$', 'vbg'), # gerunds
...     (r'.*ed$', 'vbd'), # past tense verbs
...     (r'.*', 'nn')      # nouns (default)
... ]
>>> regexp_tagger = tag.Regexp(patterns)
>>> list(regexp_tagger.tag(brown.raw('a')))[3]
[('', 'nn'), ('Only', 'nn'), ('a', 'nn'), ('relative', 'nn'), ...,
 ('received', 'vbd'), ..., ('considering', 'vbg'), ... ]

```

6

<div data-bbox="173 254 589 281" data-label="Section-Header"> <h3>Obtaining parts of corpora for training</h3> </div> <div data-bbox="27 348 725 371" data-label="Text"> <p>If we want to train taggers, we need to obtain (slice) a certain portion of the corpus</p> </div> <div data-bbox="27 392 555 462" data-label="Text"> <pre>>>> from nltk_lite.corpora import brown >>> from itertools import islice >>> train_sents = list(islice(brown.tagged(), 500))</pre> </div> <div data-bbox="27 483 341 504" data-label="Text"> <p>This gives us sentences 1 through 499</p> </div> <div data-bbox="732 703 740 718" data-label="Text"> <p>7</p> </div>	<div data-bbox="1105 254 1281 281" data-label="Section-Header"> <h3>Unigram Tagger</h3> </div> <div data-bbox="837 348 1425 371" data-label="Text"> <p>So, we can train a unigram tagger (i.e., most likely tag for each word):</p> </div> <div data-bbox="837 392 1221 438" data-label="Text"> <pre>>>> unigram_tagger = tag.Unigram() >>> unigram_tagger.train(train_sents)</pre> </div> <div data-bbox="837 459 1034 480" data-label="Text"> <p>And then we can run it:</p> </div> <div data-bbox="837 501 1494 619" data-label="Text"> <pre>>>> text = "John saw the book on the table" >>> tokens = list(tokenize.whitespace(text)) >>> list(unigram_tagger.tag(tokens)) [('John', 'np'), ('saw', 'vbd'), ('the', 'at'), ('book', None), ('on', 'in'), ('the', 'at'), ('table', None)]</pre> </div> <div data-bbox="837 640 1245 661" data-label="Text"> <p>Note that unknown words are given the tag None</p> </div> <div data-bbox="1544 703 1552 718" data-label="Text"> <p>8</p> </div>
<div data-bbox="230 829 534 886" data-label="Section-Header"> <h3>N'th Order Markov Taggers Bigrams</h3> </div> <div data-bbox="27 953 428 974" data-label="Text"> <p>We can declare different n-gram taggers to train</p> </div> <div data-bbox="27 997 680 1113" data-label="Text"> <pre>>>> bigram_tagger = tag.Bigram() >>> bigram_tagger.train(brown.tagged(['a','b'])) >>> list(bigram_tagger.tag(tokens)) [('John', 'np'), ('saw', 'vbd'), ('the', 'at'), ('book', 'nn'), ('on', 'in'), ('the', 'at'), ('table', None)]</pre> </div> <div data-bbox="27 1134 534 1155" data-label="Text"> <p>Here, we just use bigrams and use two portions of the corpus</p> </div> <div data-bbox="732 1274 740 1289" data-label="Text"> <p>9</p> </div>	<div data-bbox="1042 829 1346 886" data-label="Section-Header"> <h3>N'th Order Markov Taggers Trigrams</h3> </div> <div data-bbox="837 953 1482 1071" data-label="Text"> <pre>>>> trigram_tagger = tag.Trigram() >>> trigram_tagger.train(brown.tagged('a')) >>> list(trigram_tagger.tag(tokens)) [('John', None), ('saw', None), ('the', None), ('book', None), ('on', None), ('the', None), ('table', None)]</pre> </div> <div data-bbox="837 1092 1240 1113" data-label="Text"> <p>We learn an important lesson from this example:</p> </div> <div data-bbox="837 1146 1414 1169" data-label="List-Group"> <ul style="list-style-type: none"> • NLTK is not very good at dealing with unknown tokens/sequences </div> <div data-bbox="1544 1274 1552 1289" data-label="Text"> <p>10</p> </div>
<div data-bbox="277 1402 487 1430" data-label="Section-Header"> <h3>Combining Taggers</h3> </div> <div data-bbox="27 1467 704 1491" data-label="Text"> <p>Because we need to have good lexical statistics, we can back off to other taggers:</p> </div> <div data-bbox="27 1512 680 1841" data-label="Text"> <pre>>>> t0 = tag.Default('nn') >>> t1 = tag.Unigram(backoff=t0) >>> t2 = tag.Bigram(backoff=t1) >>> t1.train(brown.tagged('a')) >>> t2.train(brown.tagged('a')) >>> list(t0.tag(tokens)) [('John', 'nn'), ('saw', 'nn'), ('the', 'nn'), ('book', 'nn'), ('on', 'nn'), ('the', 'nn'), ('table', 'nn')] >>> list(t1.tag(tokens)) [('John', 'np'), ('saw', 'vbd'), ('the', 'at'), ('book', 'nn'), ('on', 'in'), ('the', 'at'), ('table', 'nn')] >>> list(t2.tag(tokens)) [('John', 'np'), ('saw', 'vbd'), ('the', 'at'), ('book', 'nn'), ('on', 'in'), ('the', 'at'), ('table', 'nn')]</pre> </div> <div data-bbox="725 1848 740 1862" data-label="Text"> <p>11</p> </div>	<div data-bbox="1136 1402 1248 1430" data-label="Section-Header"> <h3>Evaluation</h3> </div> <div data-bbox="837 1497 1450 1709" data-label="Text"> <pre>>>> accuracy0 = tag.accuracy(t0, brown.tagged('b')) >>> accuracy1 = tag.accuracy(t1, brown.tagged('b')) >>> accuracy2 = tag.accuracy(t2, brown.tagged('b')) >>> print 'Default Accuracy = %4.1f%%' % (100 * accuracy0) Default Accuracy = 12.5% >>> print 'Unigram Accuracy = %4.1f%%' % (100 * accuracy1) Unigram Accuracy = 80.2% >>> print 'Bigram Accuracy = %4.1f%%' % (100 * accuracy2) Bigram Accuracy = 78.3%</pre> </div> <div data-bbox="1544 1848 1552 1862" data-label="Text"> <p>12</p> </div>

Evaluation

Building a Confusion Matrix

Aside from just seeing accuracies, a good analysis would consider which tags are commonly mistakenly tagged for others

Exercise: (in groups)

- Build a **confusion matrix** for tagging section 'b' of the corpus
- First, decide what you need, and then decide what the syntax should be