NLP Lab Session Week 4
September 22, 2011

Tokenization and (Starting) POS Tagging

We will start with some preliminaries to next week's work with the NLTK POS taggers and end with a look at tokenization.

Reading Tagged Corpora

The NLTK corpus readers have additional methods (aka functions) that can give the additional tag information from reading a tagged corpus. Both the Brown corpus and the Penn Treebank corpus have text in which each token has been tagged with a POS tag. (These were manually assigned by annotaters.)

The tagged_sents function gives a list of sentences, each sentence is a list of (word, tag) tuples. (Python tuples are a 2-element list that you can't change the order of the elements.) We'll first look at the Brown corpus, which is described in Chapter 2 of the NLTK book.

```
import nltk
from nltk.corpus import brown
brown.tagged_sents()[:2]
```

The tagged_words function just gives a list of all the (word, tag) tuples, ignoring the sentence structure.

```
brown.tagged_words()[:50]
```

The Brown corpus is organized into different types of text, which can be selected by the categories argument, and it also allows you to map the tags to a simplified tag set, described in table 5.1 in the NLTK book.

```
brown_news_tagged = brown.tagged_words(categories='news', simplify_tags=True)
brown_news_tagged[:50]
```

Other tagged corpora also come with the tagged_words method:

```
nltk.corpus.nps_chat.tagged_words()[:50]
```

The Penn Treebank has the tagged_words and tagged_sents methods as well.

```
from nltk.corpus import treebank
treebank.tagged_words()[:50]
len(treebank.tagged_words())
```

In our previous labs, we used the Frequency Distributions of the NLTK to map words to numbers (the frequencies), and we noted that this is just a Python dictionary. NLTK also has a structure called the Conditional Frequency Distribution which is a nested dictionary. It allows you to map a (word, tag) pair to a frequency. NLTK calls the first set of keys the "conditions", which index a regular frequency distribution with keys and values.

Here is a function definition that finds the most frequent words that are tagged with any of the noun tags, those prefixed with NN.

```
# find the most frequent words in Penn Treebank that have one of the noun tags
def findtags(tag_prefix, tagged_text):
        cfd = nltk.ConditionalFreqDist((tag, word) for (word, tag) in tagged_text
                                        if tag.startswith(tag_prefix))
        return dict((tag, cfd[tag].keys()[:20]) for tag in cfd.conditions())

tagdict = findtags('NN', treebank.tagged_words())
for tag in sorted(tagdict):
        print tag, tagdict[tag]
```

In the next lab, we'll use these tagged corpora to train a POS tagger and test its accuracy.

Regular Expression Tokenizer
(From section 3.7 in the NLTK book http://www.nltk.org/book)

So far, we have depended on the NLTK wordpunct tokenizer for our tokenization. Not only does the NLTK have other tokenizers, but we can custom-build our own tokenizer if we wish. Here is a regular expression that will match any sequence of word characters, or anything that starts with a non-space character optionally followed by some word characters. We can use the regular expression function findall to apply this pattern to text:

```
import re
p = re.compile('\w+|\S\w*')
emmatext[:50]
p.findall,emmatext[:50])
```

This tokenizer is simple enough that just two regular expression patterns can be used to identify text to be put into a token.

But NLTK has built a tokenizing function that helps you write tokenizers by giving it the compiled pattern. Regular expressions can also be written down in the "verbose" version that allows the alternatives to be on different lines with comments.

```
pattern = r''' (?x)     # set flag to allow verbose regexps
      ([A-Z]\.)+            # abbreviations, e.g. U.S.A
      | \w+(-\w+)*          # words with internal hyphens
      | \$?\d+(\.\d+)?%?    # currency and percentages, $12.40, 50%
      | \.\.\.              # ellipsis
      | [][.,;"'?():-_']    # separate special character tokens
```

```
    '''
```

As far as I can tell, the nltk function regexp_tokenize applies these regular expressions to text by applying each regular expression in order to get anything that matches as a token. So note that it is important that the expression to separate special characters as individual tokens comes last in the list, so that other expressions, such as the words with internal hyphens, can first get longer tokens that involve individual characters.

We can test this function on Emma text and also on some text with additional interesting tokens.

nltk.regexp_tokenize(emmatext[:300], pattern)

specialtext = 'That U.S.A. poster-print costs $12.40 and there are 1,259,000 copies at 50% off.'

nltk.regexp_tokenize(specialtext, pattern)

Note that if there are any characters not matched by one of the regular expression patterns, then it is omitted as a token in the result.

Next, we'll try to make a regular expression tokenizer appropriate for tweet text. Some of the patterns in this tokenizer are taken from tweetmotif, a Python regular expression tokenizer written for tweets by Brendan O'Connor (http://tweetmotif.com/about). Here is a tokenizer:

```
tweetPattern = r''' (?x)          # set flag to allow verbose regexps
      (https?://|www)\S+          # simple URLs
      | (:-\)|;-\))               # small list of emoticons
      | &(amp|lt|gt|quot);        # XML or HTML entity
      | \#\w+                     # hashtags
      | @\w+                      # mentions
      | \d+:\d+                   # timelike pattern
      | \d+\.\d+                  # number with a decimal
      | (\d+,)+?\d{3}(?=([^,]|$))   # number with a comma
      | ([A-Z]\.)+                # simple abbreviations
      | (--+)                     # multiple dashes
      | \w+(-\w+)*                # words with internal hyphens or apostrophes
      | ['\".?!,:;]+              # special characters
      '''
```

We need some examples to work with.

tweet1 = "@natalieohayre I agree #hc09 needs reform- but not by crooked politicians who r clueless about healthcare! #tcot #fishy NO GOV'T TAKEOVER!"

tweet2 = "To Sen. Roland Burris: Affordable, quality health insurance can't wait http://bit.ly/j63je #hc09 #IL #60660"

tweet3 = "RT @karoli: RT @Seriou: .@whitehouse I will stand w/ Obama on #healthcare, I trust him. #p2 #tlot"

(Sometimes there are problems copying and pasting text over a line and with different type fonts for quotes.)

Try the regexp tokenizer on these tweets, for example:

nltk.regexp_tokenize(tweet1,tweetPattern)

Exercise

Choose one of the following, i.e. work with either the regular pattern or the tweet pattern in the tokenizer. You may work in groups.

Run the regexp tokenizer with the regular pattern on the sentence "Mr. Black and Mrs. Brown attended the lecture by Dr. Gray, but Gov. White wasn't there."

- Design and add a line to the pattern of this tokenizer so that titles like "Mr." are tokenized as having the dot inside the token. Test and add some other titles to your list of titles.
- Design and add the pattern of this tokenizer so that words with a single apostrophe, such as "wasn't" are taken as a single token.

Run the regexp tokenizer with the tweet pattern on the three example tweets.

- Design and add a line to the pattern of this tokenizer so that titles like "Sen." and "Rep." are tokenized as having the dot inside the token. Test and add some other titles to your list of titles.
- Design and add to the pattern of this tokenizer so that words with a single apostrophe, such as "can't" are taken as a single token.
- Design and add to the pattern of this tokenizer so that the abbreviation "w/" is taken as a single token.

Choose at least one of your tokenizer solutions and post your revised pattern to the Discussion in the iLMS for Week 4 with a short example text that demonstrates its effect. Mention any examples that you think of that need additional regular expressions to be tokenized.