NLP Lab Session Week 9
Week 9, March 25, 2010
Processing Parse Trees / Programming Lab

Resources:

NLTK book chapter on writing Python:
http://nltk.googlecode.com/svn/trunk/doc/book/ch04.html
Python tutorial from official web site: http://docs.python.org/release/2.5.2/tut/tut.html
NLTK API: http://nltk.googlecode.com/svn/trunk/doc/api/index.html
Online Stanford parser (one sentence only): http://nlp.stanford.edu:8080/parser/index.jsp

Start the Python Idle window and NLTK as usual (run the NLKT installer and copy nltk_data if necessary)

Assume that the Stanford Parser has been run and produced an output file. Read this file from the Python window.

```
>>>f = open('C:\AAAdocs\NLPspring2010\labs\LabExamplesWeek9\desert1.tree', 'r')
>>> ftext = f.read();
>>> len(ftext)
931
>>> ftext
```

Now we want to separate the sentences in the file and work on each sentence. I could do this with regular expressions, but I chose to use the find and rfind methods on the text string.

```
>>>ftext.find('(ROOT\n')
0
>>> ftext.find('(ROOT\n', 6)
435
>>> sent = ftext[6:435]
>>> sent

>>> sent.rfind(')')
420
>>> sent1 = sent[0:420]
>>> sent1
```

The NLTK has a class nltk.tree.Tree to represent parse trees and we can use their method ‘parse’ to create a tree out of this string, and other methods to process the tree. (Look at the API for nltk.tree.Tree.)

```
>>> dtree = nltk.tree.Tree.parse(sent1)

>>>type(dtree)
<class 'nltk.tree.Tree'>
```

```
>>> dtree.height()
13
```

We want to look for subtrees that satisfy particular conditions, and there is a method to find all the subtrees.

```
>>> dtree.subtrees()
<generator object at 0x01FE83F0>
>>> print dtree
```

What it returns is a generator, instead of a list of subtrees. One way to use a generator is to repeatedly use the .next() function to get each subtree.

```
>>> subtrees.next()
```

Or we can use the for statement on items in a generator, and it implicitly calls .next() to get each item t:

```
>>for t in subdtree:
        print t
```

Now the generator is empty, but we can always call .subtrees and get a new one

```
>>>count = 0
>>> for st in dtree.subtrees():
        print 'Tree number ', count
        count = count + 1
        print st
```

For anything of type nltk.tree.Tree, use __getitem__ function to get a list of children.

After doing subdtree.next(), I get to:

```
>>> st3 = subdtree.next()
>>> print st3
(S
  (NP (NNP Three) (NNP Calgarians))
  (VP
    (VBP have)
    (VP
      (VBN found)
      (NP
        (NP (DT a) (ADJP (RB rather) (JJ unusual)) (NN way))
        (PP
          (IN of)
          (S
```

```
        (VP
          (VBG leaving)
          (NP (NN snow) (CC and) (NN ice))
          (ADVP (RB behind))))))))
  (. .))
```

The length of the list tells us the number of children:

```
>>> len(st3)
3
```

There are three children

```
>>> st3.__getitem__(0)
Tree('NP', [Tree('NNP', ['Three']), Tree('NNP', ['Calgarians'])])
>>> st3.__getitem__(1)
……
>>> st3.__getitem__(2)
Tree('.', ['.'])
```

But trying to get another child will give an error.

The actual value of the top-level node (in this case the phrase tag) is given by the node attribute:

```
>>>st3.node
'S'
```

The flatten function can give a tree with just the leaves, and the leaves function gives just the list of leaves.

```
>>> st3.flatten()
Tree('S', ['Three', 'Calgarians', 'have', 'found', 'a', 'rather', 'unusual', 'way', 'of', 'leaving', 'snow',
'and', 'ice', 'behind', '.'])
```

The pos function gives a list with each leaf and its POS tag.

```
>>>st3.pos()
[('Three', 'NNP'), ('Calgarians', 'NNP'), ('have', 'VBP'), ('found', 'VBN'), ('a', 'DT'), ('rather', 'RB'),
('unusual', 'JJ'), ('way', 'NN'), ('of', 'IN'), ('leaving', 'VBG'), ('snow', 'NN'), ('and', 'CC'), ('ice',
'NN'), ('behind', 'RB'), ('.', '.')]
```

Suppose we want to write a function that can find pairs where we have a pattern in the tree with

```
(S
        (NP  ….
        (VP ….
```

And we want to return all the leaves of the noun phrase, e.g. “Three Calgarians”, and the beginning leaves of the VP phrase that have a verb POS: e.g. “have found”

Develop a function

def finSubjectVerb(tree)

which will take a tree representing one sentence and return a list of pairs that have the words from the NP and from the VP all the verb words that start the phrase

(Digress to talk about the function and how each part works to find the noun words and the verb words.)

After developing this function, what must be done to finish it?

- Run the Stanford parser on a text document.
- Write a driver program that reads the file, separates the sentences and runs the function on the parse tree generated from each sentence in the file
- Add comments
- Test on many sentences
- Decide if the function needs to be extended to handle more cases
  - (nested sentences – where the phrase tag can be S, SBAR, etc.)
  - (verbs can skip over adverbs or other insignificant words)
  - Other??
- Develop the function to do these cases
- Find several significant cases to show what the function can do for the report

What other functions can be written?

Preparing Parser Output Files

Sentence Segmentation

Before tokenizing the text into words, we need to segment it into sentences. NLTK facilitates this by including the Punkt sentence segmenter

```
>>> import pprint
>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = sent_tokenizer.tokenize(text)
>>> pprint.pprint(sents[0:10])
```

```
>>>sents = nltk.corpus.treebank_raw.sents()
>>>for sent in nltk.corpus.treebank_raw.sents():
```

Run the Stanford Parser as described in StanfordParserFiles.doc.