NLP Lab Session Week 7
March 4, 2010
Parsing in NLTK

Installing NLTK Toolkit and the Stanford Parser

Reinstall nltk-2.0b7.win32.msi and Copy and Paste nltk_data from H:\ nltk_data to C:\ nltk_data

This week we will also install the Stanford Parser. Go to the URL:

http://nlp.stanford.edu/software/lex-parser.shtml

Scroll down to the download section and click on the link:

Download Stanford Parser Version 1.6.2

Save the zip file on your H: drive if you have approximately 140 MB available; otherwise put it on your C: drive (and you'll have to download it again when you need it). Unzip the parser.

Getting Started

As we did in the last lab session, we will work together through a series of small examples using the IDLE window that will be described in this lab document. However, for purposes of using cut-and-paste to put examples into IDLE, the examples can also be found in a python file on the iLMS system, under Resources.

Labweek7parsing.py

Open an IDLE window. Use the File-> Open to open the labweek7parsing.py file. This should start another IDLE window with the program in it. **Each example line(s)** can be cut-and-paste to the IDLE window to try it out.

Running parsing demos

As always, we start by importing from nltk all of the programs

```
import nltk
```

The first parsing demo shows the recursive descent parser, which is a top-down, back-tracking parser. The second shows the shift-reduce parser, which is a bottom-up parser and needs guidance as to what operation (shift or reduce) to apply at some steps. The third shows a chart parser in top-down strategy (1); it also has strategies for bottom-up, bottom-up left corner and stepping.

```
nltk.app.rdparser()
nltk.app.srparser()
nltk.parse.chart.demo(1, should_print_times=False, trace=1)
```

Running Parsers

In NLTK, the parsers that are provided all need a grammar to operate. The parse_cfg function is given to take a normal string representation of a CFG grammar and convert it to a form that the parsers can use. Here is an example:

```
>>> grammar = nltk.parse_cfg("""
 S -> NP VP
 VP -> V NP | V NP PP
 PP -> P NP
 V -> "saw" | "ate" | "walked"
 NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
 Det -> "a" | "an" | "the" | "my"
 N -> "man" | "dog" | "cat" | "telescope" | "park"
 P -> "in" | "on" | "by" | "with"
 """)
```

First, we define a recursive descent parser from this grammar and then test it on a short sentence. The recursive descent parser is further described in the NLTK book in section 8.5.

```
>>> rd_parser = nltk.RecursiveDescentParser(grammar)
>>> sent = "Mary saw Bob".split()
>>> for tree in rd_parser.nbest_parse(sent):
        print tree
```

Note that this parser returns n (all?) the parse trees, so we can try this out on a syntactically ambiguous sentence.

```
>>> sent2 = "John saw the man in the park with a telescope".split()
>>> for tree in rd_parser.nbest_parse(sent2):
        print tree
```

If you try other sentences, don't put the punctuation at the end because we didn't include any punctuation in the grammar.

We can add words to our grammar in order to parse other sentences.

```
groucho_grammar = nltk.parse_cfg("""
 S -> NP VP
 VP -> V NP | V NP PP
 PP -> P NP
 V -> "saw" | "ate" | "walked" | "shot"
 NP -> "John" | "Mary" | "Bob" | "I" | Det N | Det N PP
 Det -> "a" | "an" | "the" | "my"
 N -> "man" | "dog" | "cat" | "telescope" | "park" | "elephant" | "pajamas"
 P -> "in" | "on" | "by" | "with"
 """)
```

Next we make a shift-reduce parser from the groucho grammar and test it on a simple sentence. The shift-reduce parser is also further described in section 8.5 of the NLTK book.

```
sr_parse = nltk.ShiftReduceParser(groucho_grammar)
sent3 = 'Mary saw a dog'.split()
print sr_parse.parse(sent3)
```

Next we test it on a more complicated sentence, but it doesn't find a parse tree because its automatic selection of shift-reduce operators is not sophisticated enough.

```
sent4 = "I shot an elephant in my pajamas".split()
print sr_parse.parse(sent4)
```

If you like, try making a recursive descent parser with the groucho grammar and observe the trees. (Note that we were careful not to include rewrite rules such as VP -> VP PP, because that would involve an infinite recursion in the recursive descent parser.)

NLTK also has a dependency parser for projective sentences. For that, we need to make a dependency grammar that shows the dependency relation between words. Note that this is an unlabeled dependency grammar.

```
groucho_dep_grammar = nltk.parse_dependency_grammar("""
  'shot' -> 'I' | 'elephant' | 'in'
  'elephant' -> 'an' | 'in'
  'in' -> 'pajamas'
  'pajamas' -> 'my'
  """)
```

Now we can make a dependency parser and test it.

```
pdp = nltk.ProjectiveDependencyParser(groucho_dep_grammar)
trees = pdp.parse(sent4)
for tree in trees:
  print tree
```

Annotated syntax trees

From Penn Treebank, we can view the syntax trees of the sentences. Recall that these were hand annotated and can be used to make context free grammars.

```
t = treebank.parsed_sents('wsj_0001.mrg')[0]
print t
```

NLTK also has some methods to help extract grammars from Penn Treebank.

Stanford Parser

Go to where you unzipped the Stanford parser, go into the folder and double-click on the lexparser-gui.bat icon. This will run a GUI with a parser demo.

In the GUI window, click Load Parser, Browse, go to the parser folder and select englishPCFG.ser.gz. It will take a couple of minutes to load the parser and it will give a message in the lower pane.

To test the parser, we can type a simple sentence in the upper pane and select it (it should be yellow). Click on Parse and the parse tree should appear in the lower pane.

We can also load text files. In the GUI window, click Load File, Browse, and navigate to and select testsent.txt in the top directory of the parser distribution folder or navigate to any other text file that you want to load. The first sentence should be selected, or you can use Prev and Next to select other sentences. Then you can click Parse to see each sentence.

Later we will run the parser from the command line so that we get files of parser output. To see the type of output that is generated, look at the example at the bottom of the Stanford parser web page (from mumbai.txt).

Exercise in understanding parser output:

Choose a sentence from any text file that you have, or make up a sentence. The sentence should have approximately 12 – 20 words. Run it in the Stanford parser demo window.

Now convert the resulting parse tree to line-oriented parse format, such as the Penn Treebank parsed sentences. For example, when I parse "Bob saw a man.", I get the tree that can be represented as (ignoring Root):

```
(S
  (NP
      (NNP Bob))
  (VP
      (VBD saw)
      (NP (DT a) (NN man)))
  (. .)
)
```

Try to make your example follow the conventions for lines and indentation as best you can understand from the Treebank examples.

Post your example in the Discussion group for lab today.

(If you want to check your example, you can put it in a file and run the Stanford parser from the command line to get this output. But please try to do it by hand first! The goal is to get understanding of that output and how it represents a parse tree.)