NLP Homework 2
Due April 1, 2010

For this homework, choose one of the following tasks. If you choose task 1, you may choose to work in a group of two and just do more rounds of development and description of results. If you want to work in a group with more than two people, you must propose additional work to me. If you choose task 2, there will be group work for the programming design and teams will work well to carry out implementations with sub-tasks of programming, analysis and testing.

1. Chunk Parsing for Base Noun Phrases

What you will learn: More experience with regular expressions, syntactic knowledge of noun phrases, process of developing rules to improve accuracy of chunking.

For this task, you will do the analysis for a base noun phrase chunker for news style text. The noun chunker will use the NLTK functions to make a regular expression grammar based on POS tags, to make the chunker and to test it on text.

Recall that we had an simple example of a regular expression grammar:

```
NPchunkgrammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}       # determiner/possessive, adjectives, nouns
      {<NNP>+}                    # sequences of proper nouns
"""
```

To obtain this grammar, we may observer examples of noun phrases from the Wall Street Journal corpus:

another/DT sharp/JJ dive/NN
Pierre/NNP Vinken/NNP

For this assignment, we are going to extend this chunk grammar to do additional forms of noun phrases.

To explain the main idea of the assignment, first look at the first 50 sentences of the annotated Wall Street Journal and select any base noun phrases that will not match either of the above rules. Remember that base noun phrases are annotated with NP, but do not include any other noun phrases.

To look at the first sentence, we run: t = treebank.parsed_sents('wsj_0001.mrg')[0] and get:

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
```

```
  (VP
    (VB join)
    (NP (DT the) (NN board))
    (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
    (NP-TMP (NNP Nov.) (CD 29))))
 (. .))
```

To start with, we disregard the temporal noun phrase (NP-TMP) and observe that the other base NPs are:

(NP (NNP Pierre) (NNP Vinken))
(NP (CD 61) (NNS years))
(NP (DT the) (NN board))
(NP (DT a) (JJ nonexecutive) (NN director))

All these noun phrases match one of the two patterns except for the phrase "61 years". So we would collect that as a noun phrase that needs additional patterns.

Developing a Chunker

For our development, we are actually going to use the CoNLL corpus with the Wall Street Journal; it is the same corpus as Penn Treebank, but with a different format for chunk annotation. For this corpus, we can build regular expression chunk parsers with NLTK and then use its chunk scoring to evaluate how well our parser does so far. We can examine the errors and omissions of the chunking to determine additional regular expressions.

The use of the NLTK Regular Expression parser (RegexpParser) and the chunkscore function to develop a chunk parser are detailed in the python file called homework3examples.py. These examples will be discussed in lab.

For your homework assignment, continue the development of the chunk parser for 5-10 more rounds, improving the recall and precision, if possible. You may continue to use the first 5 files of the CoNLL corpus for your development and testing.

What to Hand In

Hand in a document that describes the final result of your chunk parser: the regular expression grammar and the results of chunkscore. Although you should NOT give every detail of your development process, you should write a brief discussion of any interesting aspects that occurred as you developed the chunk parser. These should include the following discussion questions:

Did you have difficulties with the ordering of the rules? For example, did an earlier rule capture an (incorrect) noun phrase that prevented a later rule from taking effect?

Did you introduce any rules that did not improve both precision and recall? For example, did you introduce a rule that gave greater recall, but reduced precision?

Were there any noun phrases that caused you particular problems in writing the rules? For example, you may observe chunked noun phrases where you believe that the human annotation for the gold standard is in error. Or you may write rules for phrases that are so rare that it hardly affects the score.

Submit your report to the iLMS system assignment dropbox at the end of the day on the due date.

2. Python Functions for Stanford Parser Output

What you will learn: more about Python programming, familiarity with parse structures for constituent trees and dependency grammars

For this task, you will write Python functions that take Stanford Parser output and search for various patterns of text. The design of the patterns will be to find useful word and phrase patterns for information extraction and classification tasks, such as sentiment or opinion analysis, that we may do for final projects.

For processing Stanford parser output, we want to assume that there is parser output from text in a file. From Python, we can read the file and use one or more functions to get lists of words or phrases from the text that satisfy particular patterns.

Getting Stanford parser output:
So far, we have identified two techniques for getting the line-oriented parser output in a file. The first uses the online demo (not the GUI from the downloaded parser) and puts output in a file by hand. The second uses the downloaded parser API. These are both described in the StanfordParser.doc.

Function list so far:.
1. Write a function that takes a list of POS tags and a list of words. It should return all the occurrences in the text of words that are on the list and were tagged with one of the tag. Example use case: identifying a list of words with an adjective tag that were on one of the affective word lists, for sentiment analysis.

2. Write a function that takes a list of subject words and a list of verb words and returns all the phrase pairs that consist of a noun phrase containing one of the subject words paired with a verb phrase that contains one of the verbs. Example use case: identifying opinion holders in opinion analysis.

3. Write a function that takes a list of subject words, a list of connecting words, and a list of object words, and returns the triples consisting of a noun phrase containing one of the subject words, one of the connecting words, and a noun phrase containing one of the object words. Example use case: identifying “entity relation entity” triples for information extraction relations.

I could give more specific Python function specifications, but I prefer to do that as a group exercise. There will be at least one scheduled group session in which we work out the details of writing the Python functions and give a more detailed specification for each one. This will be scheduled during the week after spring break. (Of course, you may also go ahead and work out specification details yourself.)

To complete the assignment, you or your team must design a set of test cases and test the programs. The programs should be well documented and suitable for use by other students in the class for their projects. You will hand in a short report with the overall description of your functions and the test cases and testing process, along with the commented python code file.

Submit your report and yor code to the iLMS system assignment dropbox at the end of the day on the due date.