NLP Lab Session
Week 9, March 20, 2013
Using WordNet in NLTK

Getting Started
As usual, 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.

Labweek9wordnet.py

Open an IDLE window. Use the File-> Open to open the labweek9wordnet.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.

Starting NLTK

WordNet is imported from NLTK like other corpus readers and more details about using WordNet can be found in the NLTK book in section 2.5 of chapter 2.

For convenience in typing examples, we can shorten its name to 'wn'.

```
>>> import nltk
>>> from nltk.corpus import wordnet as wn
```

Synsets and lemmas

Although WordNet is usually used to investigate words, its unit of analysis is called a synset, representing one sense of the word. For an arbitrary word, i.e. dog, it may have different senses, and we can find its synsets. Note that **each synset is given an identifier** which includes **one** of the actual words in the synset, whether it is a noun, verb, adjective or adverb, and a number, which is relative to all the synsets listed for the particular actual word.

```
>>> wn.synsets('dog')
```

Once you have a synset, there are functions to find the information on that synset, and we will start with "lemma_names", "lemmas", "definitions" and "examples".

For the first synset 'dog.n.01', which means the first noun sense of 'dog', we can first find all of its words/lemma names. These are all the words that are synonyms of this sense of 'dog'.

```
>>> wn.synset('dog.n.01').lemma_names
```

Given a synset, find all its lemmas, where **a lemma is the pairing of a word with a synset**.

```
>>> wn.synset('dog.n.01').lemmas
```

Given a lemma, find its synset

```
>>> wn.lemma('dog.n.01.domestic_dog').synset
```

Given a word, find lemmas contained in all synsets it belongs to

```
>>> for synset in wn.synsets('dog'):
      print synset, ": ", synset.lemma_names
```

Given a word, find all lemmas involving the word. Note that these are the synsets of the word 'dog', but just also showing that 'dog' is one of the words in each of the synsets.

```
>>> wn.lemmas('dog')
```

Definitions and examples:

The other functions of synsets give the additional information of definitions and examples. Find definitions of the synset for the first sense of the word 'dog':

```
>>> wn.synset('dog.n.01').definition
```

Display an example use of the synset

```
>>> wn.synset('dog.n.01').examples
```

Or we can show all the synsets and their definitions:

```
>>> for synset in wn.synsets('dog'):
    print synset, ": ", synset.definition
```

Lexical relations

WordNet contains many relations between synsets. In particular, we quite often explore the hierarchy of WordNet synsets induced by the hypernym and hyponym relations. (These relations are sometimes called "is-a" because they represent abstract levels of what things are.) Take a look at the WordNet Hierarchy diagram in section 2.5 of the NLTK book.

Find hypernyms of a synset of 'dog':

```
>>> dog1 = wn.synset('dog.n.01')
 >>> dog1.hypernyms()
```

Find hyponyms

```
 >>> dog1.hyponyms()
```

We can find the most general hypernym as the root hypernym

```
>>>dog1.root_hypernyms()
```

There are other lexical relations, such as those about part/whole relations. The components of something are given by meronymy; NLTK has two functions for two types of meronymy, part_meronymy and substance_meronomy. It also has a function for things they are contained in, member_holonymy.

NLTK also has functions for antonymy, or the relation of being opposite in meaning. Antonymy is a relation that holds between lemmas, since words of the same synset may have different antonyms.

```
>>> good1 = wn.synset('good.a.01')
>>> wn.lemmas('good')
>>> good1.lemmas[0].antonyms()
```

Another type of lexical relation is the entailment of a verb (the meaning of one verb implies the other)

```
>>> wn.synset('walk.v.01').entailments()
```

There are more functions to use hypernyms to explore the WordNet hierarchy. In particular, we may want to use paths through the hierarchy in order to explore word similarity, finding words with similar meanings, or finding how close two words are in meaning.

Exercise:

1. Pick a word and show all the synsets of that word and their definitions.
2. Pick one synset of the word and show all of its hypernyms.
3. Show the hypernym path between the top of the hierarchy and the word.

Put the results of your three steps into the discussion for this week in the iLMS, along with any other interesting examples (as long as they are not too lengthy!).

Understanding treebank annotations and Stanford parser output:

In the last lab, we looked at Penn Treebank noun phrase chunks. The Penn Treebank annotations also contain fully parsed sentences.

Recall that these were hand annotated and can be used to make context free grammars. For help in understanding the parse tree node non-terminals (which Penn Treebank calls tags), there is an overview at http://bulba.sdsu.edu/jeanette/thesis/PennTags.html, and more detailed information at the Penn Treebank tagging guide, "Bracketing Guidelines for Treebank II Style Penn Treebank Project", which is a detailed annotation document.

```
>>> from nltk.corpus import treebank

>>> t0 = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> print t0
```

(Note that here, the NLTK **print** function puts out a more visual notation for a parse tree than observing the tree value directly.)

We'll also look at the next 2 or 3 sentences. It will be easier to cycle through the sentences if we use the default argument.

```
>>> t1 = treebank.parsed_sents()[1]
>>> print t1
```

etc.

In looking at these trees, we observe that there are embedded S tags of several kinds. For some examples, see the introduction to Chapter 8 of the NLTK book and the section on Clause Types on page 16 of the Treebank guidelines. (For simple definitions of grammatical concepts such as complement and complementizer, wikipedia is a helpful source.) For information on NULL and TRACE elements, see page 60 of the Treebank guidelines.

Now open the Stanford parser demo at http://nlp.stanford.edu:8080/parser/. Use your Penn Treebank sentence or make up a sentence of suitable length and complexity. The sentence should have approximately 8 – 12 words. Run it in the Stanford parser demo window, but if the parse looks too complex, choose a simpler or shorter sentence.

Now the parse will look something like this (ignoring Root) for the simple example of "Bob saw a man":

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

 nsubj(saw-2, Bob-1)
 root(ROOT-0, saw-2)
 det(man-4, a-3)
 dobj(saw-2, man-4)
```

where the numbers are the order of the words in the sentence.

In class, we will draw parse trees for these parses on the board.

Examples:
Arthur is the king.
Arthur rides the horse near the castle.