NLP Lab Session
Week 10, April 1, 2010
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.

Labweek10examples.py

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

Open an IDLE window. Use the File-> Open to open the labweek10examples.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.

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.

```
>>> 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 name 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.) 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

```
>>> 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. One way to find semantic similarity is to find the hypernyms of two synsets.

```
>>>right = wn.synset('right_whale.n.01')
>>>orca = wn.synset('orca.n.01')
>>>minke = wn.synset('minke_whale.n.01')
>>>right.lowest_common_hypernyms(minke)
```

Of course, some words are more specific in meaning than others, the min_depth function tells how many edges there are between a word and the top of the hierarchy.

```
>>>right.min_depth()
>>>wn.synset('baleen_whale.n.01').min_depth()
>>>wn.synset('entity.n.01').min_depth()
```

Then we can calculate the similarity between two synsets by comparing the lengths of the paths between them. The score for path_similarity is between 0 (least similar) and 1(most similar). It is 1 for a synset with itself. The path_similarity function will return -1 if there is no path between the synsets.

```
>>>right.path_similarity(minke)
>>>right.path_similarity(orca)
```

The function hypernym_paths shows paths between the top of the hierarchy down to the synset. In this example, there is only one path between entity and the first sense of cat.

```
>>>cat1 = wn.synset('cat.n.01')
>>> pathscat=cat1.hypernym_paths()
>>> [synset.name for synset in pathscat[0]]
```

Other definitions of similarity are found in WordNet and are described here.

```
>>> help(wn)
```

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!).