

## NLP Lab Session Week 8

### October 21, 2015

## WordNet

### Getting Started

For this lab session download the examples: LabWeek7examples.txt and put it in your class folder for copy/pasting examples. Start your Python interpreter or IDLE session.

### WordNet in 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. Remember that you can browse WordNet on-line at <http://wordnetweb.princeton.edu/perl/webwn> or you can use the NLTK wordnet browser by opening a command prompt (or terminal) window, typing “python” to get a separate python environment. Then type “import nltk” and “nltk.app.wordnet()”, and nltk should open your default browser to a wordnet browse page. (In the lab, this will work if this is the right version of Python that has nltk installed.)

Back in our IDLE window, for convenience in typing examples, we can shorten its name to ‘wn’.

```
>>> 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.

While using the wordnet functions in the following section, it is useful to search for the word ‘dog’ in the on-line WordNet at <http://wordnetweb.princeton.edu/perl/webwn>

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

**Once you have a synset id, there are functions to find the information on that synset, and we will look at “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, Figure 2.11, in section 2.5 WordNet 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_meronymy`. It also has a function for things they are contained in, `member_holonymy`.

For quickly seeing which of these relations are defined for any word, if we have a separate Python shell, we can open the NLTK wordnet explorer app, which opens in the default browser.

```
>>> nltk.app.wordnet()
```

We observe that the first sense of the word dog has Part Meronyms and Member Holonyms defined. In NLTK, we can also use functions to get those relations.

```
>>> dog1.part_meronyms()
# check out the definition of this part meronym of dog
>>> wn.synset('flag.n.07').definition
>>> dog1.member_holonyms()
```

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. It is more likely to find antonyms defined for adjectives than nouns.

```
>>> good1 = wn.synset('good.a.01')
>>> good1.lemmas
>>> 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()
```

NLTK lets us use the hypernym relations to explore the hierarchy of words in WordNet. We can use `hyponym_paths` to find all the paths from the first sense of `dog` to the root, and list the synset names of all the entities along those two paths.

```
dog1.hypernyms()
paths=dog1.hypernym_paths()
len(paths)
[synset.name for synset in paths[0]]
[synset.name for synset in paths[1]]
```

## Word Similarity

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. Start by looking up three different types of whales and looking at their hypernym paths to the top of the hierarchy.

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

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

For any synset of a word, the function `lowest_common_hypernyms` will find the least ancestor of it with another word.

```
>>> 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 the log of the path length and 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)
```

Or we might want to compare these with words that are not very similar:

```
>>> tortoise = wn.synset('tortoise.n.01')
>>> novel = wn.synset('novel.n.01')
```

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

<http://www.nltk.org/howto/wordnet.html>

Or you can access the help text for WordNet. (When you want to exit the Python help command, type 'q' or 'quit'.)

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

We can try the Leacock-Chodorow Similarity, which uses the path lengths, but also uses how deep the least common ancestor is in the hierarchy.

```
>>> right.lch_similarity(orca)
>>> right.lch_similarity(tortoise)
>>> right.lch_similarity(novel)
```

Other word similarity measures add something about the “information content” of the word in a particular corpus, which is based on the relative frequency of the word in the corpus. Resnik similarity is one of those measures.

```
>>> from nltk.corpus import wordnet_ic
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
```

```
>>> right.res_similarity(orca)
>>> right.res_similarity(tortoise)
>>> right.res_similarity(novel)
```

Note that the similarity measures are not computed on the same scale, i.e. it is not the actual similarity score that matters, but how that score compares to other scores computed by the same measure.

### **Exercise:**

Pick a word and show the following things from the nltk functions:

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

Use the NLTK browser:

4. See what other relations are defined for your word and show one or more of them.

Put the results of your four steps into the discussion for this week in Blackboard, along with any other interesting examples (as long as they are not too lengthy!). Please put your **word in the title of your post**.