NLP Lab Session Week 7
October 12, 2017

WordNet

Getting Started

For this lab session download the examples: LabWeek7examples.WordNet.ipynb or .py and put it in your class folder for copy/pasting examples. Start your jupyter notebook 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

There is also an NLTK wordnet browser - try 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. (This may not behave correctly in all environments, but it is not required for our lab.)

Back in our python interpreter 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()
```

Summary so far:

- Given a *word (dog),* find the *synset ids: ([Synset('dog.n.01'), Synset('frump.n.01'), etc.])*
- Given a *synset id (‘dog.n.01’),* get the synset and find the synonyms (lemma names) of this sense *(['dog', 'domestic_dog', 'Canis_familiaris'])*
- Lemmas pair the synset id with the lemma names (synonyms).

Just using these concepts, there are other functions in NLTK that help you map from one kind of information about a word to some of the other kinds. 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. Since this prints all the synonym words for each sense, it gives a better idea of what each sense is:

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

There are other functions shown in the NLTK book (and in the API).

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())
```

And finally, putting it all together, for a word, we can show all the synonyms, definitions and examples:

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

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_meronomy. 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, we can open the wordnet browser:

http://wordnetweb.princeton.edu/perl/webwn

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. First, we look at the parts of a dog (noting that very few are given in wordnet).

```
>>> dog1.part_meronyms()
# what is this? check it out
print (wn.synset('flag.n.07').lemma_names(),wn.synset('flag.n.07').definition(),
wn.synset('flag.n.07').examples())
```

Then we look at things that a dog is a member of:

```
>>> 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. First, we get an adjective

```
>>> good1 = wn.synset('good.a.01')
>>> good1.lemma_names()
```
But the antonym function is only defined on lemmas, not on synsets.
```
>>> 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 hypernym_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)
```
Compare the two hypernym paths for the first sense of dog.
```
[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. (In the python interpreter, use the space bar to page through the help, and when you want to exit the Python help command, type ‘q’ or ‘quit’.) (In jupyter notebook, the help is given in a scrollable box.)

```
>>> 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. Typically, the information content (IC) of word senses is to combine knowledge of their hierarchical structure from an ontology like WordNet with statistics on their actual usage in text as derived from a large 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.

SentiWordNet, a sentiment lexicon in NLTK

So far, there appears to be only one sentiment lexicon in NLTK. [If there is time, we will review the lists of NLTK resources http://www.nltk.org/howto/corpus.html] The documentation for SentiWordNet is on this HowTo page: http://www.nltk.org/howto/sentiwordnet.html

In this sentiment lexicon, each word is judged to be made up of partly positive, negative and objective meaning, and 3 scores are given as to how much of each, where the scores must sum to 1.

```
>>> from nltk.corpus import sentiwordnet as swn
```

Note that there are not very many functions listed, but the trick is that the synsets in SentiWordNet are the same as in WordNet, so we can use the functions to find the synonyms, definitions and examples in wordnet.

```
>>> list(swn.senti_synsets('breakdown'))
>>> wn.synsets('breakdown')
```

Following the example in the HowTo page, we look at the third sense of the word breakdown:

```
>>> breakdown3 = swn.senti_synset('breakdown.n.03')
```

The print function for senti_synsets will give the positive and negative scores.

```
>>> print (breakdown3)
```

There are functions to access these two scores separately, and a third function to access the objective scores.

```
>>> breakdown3.pos_score()
>>> breakdown3.neg_score()
>>> breakdown3.obj_score()
```

Exercise:

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

1. Show all the synsets of that word, their lemma names, definitions, and examples.
2. Pick one synset of the word and show all of its direct hypernyms.
3. Show the hypernym paths between the top of the hierarchy and that word sense.

Use the online WordNet search:

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

5. Use SentiWordNet to get the senti_synset of the sense of the word that you picked in part 2. Show the positive, negative and objective sentiment scores for that word, if any.

Put the results of your five 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**.