CMSC 723: Computational Linguistics I

Assignment 5: Lexical Semantics with WordNet

Jimmy Lin (Instructor) and Melissa Egan (TA)

Due: **November 18, 2009**

Introduction

For this assignment, you will primarily be working with the `wordnet` distribution and reader that comes bundled with NLTK. Technically, wordnet is not really a corpus in the sense of the word that we have been using in the assignments so far. However, NLTK treats it like one for the sake of convenience. As discussed in class, wordnet is a collection of four separate taxonomies of synsets: one each for nouns, verbs, adjectives and adverbs. Listings 1, 2 and 3 show how these taxonomies can be explored in NLTK and cover all the functionality that you will need for this assignment. Please read these listings carefully before attempting the problems.

Installation

1. Make sure that you have installed the wordnet distribution. To do so, run the following code at the python interpreter:

   ```
   >>> import nltk
   >>> nltk.download()
   ```

   You should see a window come up called 'NLTK Downloader'. Select the 'Corpora' tab and scroll down to the 'wordnet' entry. If it is green and says "installed", it means you have the latest version installed. If it says "out of date" (in red) or "not installed", you need to install it. Make sure that the 'Download directory' field is set correctly and click the 'Download' button. This should download and unzip the wordnet corpus and the wordnet entry should now say installed (in green).

2. The wordnet corpus reader module was created as a replacement for an older wordnet interface that has been deprecated. However, we found that the new interface had a few bugs and lacked some functionality that you will need for this assignment. We have already submitted a

new version of the interface with the bugfixes and needed functionality to the NLTK project. However, we cannot wait for them to release it. So, we will provide you with instructions on how to replace your installed copy of the wordnet corpus reader module with the new version (which is available on the course home page for this assignment). **NOTE: You should NOT inspect the code in this module as some parts of it have been formulated into questions for this assignment.** Just copy it to the right place as explained. Run the following code:

```
>>> import nltk
>>> nltk.__file__
'/opt/lib/python/site-packages/nltk/__init__.pyc'
```

You will most likely get a different path. Let `$NLTKDIR` represent the path of the above file, i.e., for this example `$NLTKDIR` is the directory `/opt/lib/python/site-packages/nltk`. Now do the following:

- Rename the file `$NLTKDIR/corpus/reader/wordnet.py` to `$NLTKDIR/corpus/reader/wordnet.old.py`.
- Delete the file `$NLTKDIR/corpus/reader/wordnet.pyc`.
- Copy the provided `wordnet.py` file to `$NLTKDIR/corpus/reader`.

You'll need to restart Python after the you complete the above steps.

Problem 1: WordNet Topology (40 points)

(a) Plot a graph with the number of senses of each verb in the Verb taxonomy on the vertical axis and its polysemy rank[1] on the horizontal axis. What conclusion can you draw from this graph?

(b) A WordNet Noun taxonomy may be viewed as a directed acyclic graph where the relation between the nodes is one of hypernymy/hyponymy. Figure 1 shows the first three levels of such a simplified Noun taxonomy (not all the nodes and connections are shown). The only information provided to you is that there is a single node at the first (root) level of the taxonomy. This is the synset `entity.n.01` and its offset is 1740. There are three pieces of information required for each node:

[1]The polysemy rank for a verb is defined as its position in a list of all verbs sorted by number of senses, highest first.

Listing 1: Exploring wordnet in NLTK

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

# Find *all* the senses (synsets) for the noun bank
# (other parts of speech are: wn.VERB, wn.ADJ, wn.ADV)
>>> for synsets in wn.synsets('bank', pos=wn.NOUN):
        print synset
Synset('bank.n.01')
Synset('depository_financial_institution.n.01')
Synset('bank.n.03')
Synset('bank.n.04')
Synset('bank.n.05')
Synset('bank.n.06')
Synset('bank.n.07')
Synset('savings_bank.n.02')
Synset('bank.n.09')
Synset('bank.n.10')

# You can also get the second synset for the noun 'bank' directly
# Note that the function name is now singular and note the formatting
# of the argument which implies the taxonomy to be used and which
# sense is required.
>>> synset1 = wn.synset('bank.n.02')

# Similarly for the verb 'bank'
>>> synset2 = wn.synset('bank.v.02')

# Once we have a synset, we can inspect it ...
# The list of words for this synset
>>> print synset1.lemma_names
['depository_financial_institution', 'bank', \
                    'banking_concern', 'banking_company']

# Each synset has a unique offset (identifier)
>>> print synset1.offset
8420278
```

Listing 2: Exploring wordnet (contd.)

```
# List of all direct hyponyms (descendant synsets) of this synset
>>> print synset1.hyponyms()
[Synset('agent_bank.n.02'), Synset('merchant_bank.n.01'),
Synset('home_loan_bank.n.01'), Synset('member_bank.n.01'),
Synset('commercial_bank.n.01'), Synset('acquirer.n.02'),
Synset('thrift_institution.n.01'), Synset('state_bank.n.01'),
Synset('federal_reserve_bank.n.01'), Synset('credit_union.n.01'),
Synset('lead_bank.n.01')]

# A lazy, breadth-first iterator over the subhierarchy defined by the
# hyponym relation and rooted at this synset. If second parameter
# is specified, it restricts the iterator only to that depth.
# So, to get only the direct hyponyms as above
>>> hyporel = lambda s: s.hyponyms()
>>> g = synset1.closure(hyporel, depth=1)

# Use next() method to get each element of the iterator
>>> print g.next()
Synset('agent_bank.n.02')
>>> print g.next()
Synset('merchant_bank.n.01')
# We can iterate until there's nothing left, and then
# we get an exception. Here's a loop showing how.
>>> while 1:
        try:
            print g.next()
        except StopIteration:
            break
Synset('home_loan_bank.n.01')
Synset('member_bank.n.01')
Synset('commercial_bank.n.01')
Synset('acquirer.n.02')
Synset('thrift_institution.n.01')
Synset('state_bank.n.01')
Synset('federal_reserve_bank.n.01')
Synset('credit_union.n.01')
Synset('lead_bank.n.01')
```

Listing 3: Exploring wordnet (contd.)

```
# Get the path(s) from this synset to the root, counting the distance of
# each node from the initial node on the way. A set of (synset, distance)
# tuples is returned. Note that a node may occur on several paths and
# hence may be present multiple times in the returned set with different
# distance values.
>>> dists = synset1.hypernym_distances()

# Sort by distance (which is the second field of each tuple)
>>> dists = sorted(list(dists), key=operator.itemgetter(1))

# Print out each node
# No nodes repeat in this case. To see an example of this,
# use hypernym_distances() on 'dog.n.01'
>>> for s, d in dists:
        print s,d
Synset('depository_financial_institution.n.01') 0
Synset('financial_institution.n.01') 1
Synset('institution.n.01') 2
Synset('organization.n.01') 3
Synset('social_group.n.01') 4
Synset('group.n.01') 5
Synset('abstraction.n.06') 6
Synset('entity.n.01') 7

# What is the length of the longest hypernym path from this synset
# to the root? There is also an equivalent min_depth() method
>>> print synset1.max_depth()
7

# Get a lazy iterator over all synsets for a given taxonomy, say nouns.
# Also remember that we can always convert a lazy iterator to a full
# list by using list()
>>> g = wn.all_synsets(pos=wn.NOUN)

# Similarly you can get a lazy iterator over all verb words
>>> g = wn.all_lemma_names(pos=wn.VERB)
```

- W, the synset lemma names
- N, the number of direct descendants in the taxonomy rooted at the node.
- O, the offset

Complete this three-level subgraph for the Noun taxonomy by filling in W, N and O for each of its nodes, wherever missing. Count a node multiple times if it has multiple parents.

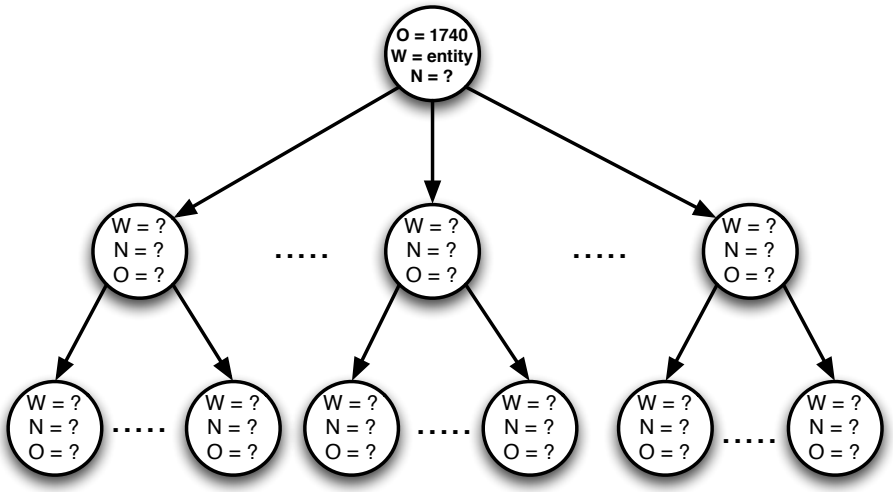

Figure 1: A subgraph of the Noun hyponymy taxonomy showing the first three levels (not all nodes are shown at all levels). The offset O for the root node is given. W refers to the synset words, O is the offset and N is the number of direct descendents rooted at the node.

(c) Compute the following statistics for the Verb, Adjective and Adverb taxonomies:

- The number of monosemous words.
- The number of ambiguous words.
- The total number of senses that are apportioned to the ambiguous words.
- Average polysemy including monosemous words.
- Average polysemy excluding monosemous words.

Hint: You may find a `FreqDist` very useful.

Problem 2: Lexical Similarity (40 points)

(a) Write a python program, using NLTK, to calculate the semantic similarity between a given pair of words as computed by the following two methods as explained in class:

- Edge-distance similarity
- Wu-Palmer similarity

(b) For each of the following word-choice problems, find the word that is closest in meaning to the given word by employing (a) the edge-distance similarity measure (b) the Wu-Palmer similarity measure. For each problem, show the similarity value assigned to each of the candidates by each of the methods.

nurse:

- *caregiver*
- professional
- surgeon

pachyderm:

- eutherian
- *elephant*
- fish

The actual right answers in both problems are italicized in both cases. Which method of measuring lexical similarity gets the right answer for both the problems? Do you think this method would generally work better than the other? If so, why?

Notes:

1. **IMPORTANT**: These similarity methods are already implemented in the wordnet NLTK module. However, the point of the assignment is to have you implement it yourself so that you can gain a better understanding of the methods. You may *not* refer to or use the NLTK implementations of these methods in any way. We will be inspecting your implementations for any obvious signs of plagiarism in accordance with the university honor code.

2. The similarity between a pair of words is defined as the maximum pairwise similarity over *all* senses of both words.

3. For Wu-Palmer similarity, recall that we have to compute the Lowest Common Subsumer (LCS) where "lowest" refers to that subsumer that's deepest in the hierarchy. An obvious way to do this is to find the subsumer that has the *longest* path to the root. However, since it is likely that there are multiple paths to the root from any node in the hierarchy and we would not want to choose artifically longer paths, we must select the shortest of such paths. So, in short, the LCS is that subsumer that has the longest shortest path to the root. This can be easily implemented by separating out the two operations, e.g., (longest (shortest path to root)).

4. When computing the length of a path from the root for Wu-Palmer, you must count *both* the start and the end node. Keep this in mind when using the `max_depth()` method.

Problem 3: Lexical Semantics (20 points)

(a) Here are two senses of the verb (not the noun) *break*:

- **Sense**: break an object into pieces.
  Example: Edgar broke the vase.
- **Sense**: break a bone.
  Example: Mildred broke her wrist.

Write down, in English and *without* using WordNet or NLTK, 5 or more additional senses. Try to do this without a dictionary if you can, but if you're not a native speaker of English, use a dictionary if absolutely necessary.

(b) Use NLTK to look up the verb senses for *break*. Which WordNet senses do your senses from part (a) match, if any? (One of your senses might match more than one WordNet sense, of course.) For example,
Sense 1 from your list might match WordNet senses 2,3,4,5.

(c) Do any of your senses group naturally into a class with common elements of meaning? How would you group them? (Use a hierarchy if that makes more sense.) Hint: You should examine your list of 5 or more senses in the context of the WordNet structure and determine whether there is a way to group these 5 to 10 senses into a smaller number of "equivalence classes".