NLP Lab Session
Week 14, December 4, 2013
More Semantics: WordNet similarity in NLTK and LDA Mallet demo
More on Final Projects: weka memory and loading Spam documents

Getting Started

For the final projects, the previous text processing program has been modified, and a new program that gets you started on creating a Spam detection classifier by showing how to read in the Spam/Ham files and create labeled documents in the same format as the movie review documents in NLTK.

WordNet Similarity

Word similarity is a way to compare how close two words are in their semantics. This has been used in NLP tasks like Word Sense Disambiguation and Topic Detection. In the NLTK, there are several similarity functions that use WordNet for their definition. Before describing those functions, we briefly review some of the functions of WordNet for synsets, definitions and hypernym/hyponym hierarchy.

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

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

Reviewing WordNet 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. These finctions include "lemma_names", "lemmas", "definitions" and "examples". For example, the function lemmas will show all the lemmas of a word. Note that here, we show the synsets of the word 'dog', but also showing that 'dog' is one of the words in each of the synsets.

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

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

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

Or we can show all the synsets and their definitions:

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

Lexical relations

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()
```

We can trace the paths of a word by visiting its hypernyms.

```
>>> paths=dog1.hypernym_paths()
>>> len(paths) #number of paths from the synset to the root concept "entity"
>>> [synset.name for synset in paths[0]] #list the first path
>>> [synset.name for synset in paths[1]] #list the second path
```

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.

```
>>>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. (Use the space bar to page through the help text. When you want to exit the Python help command, type 'q' or 'quit'.)

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

In particular, check path_similarity, lin_similarity, res_similarity (Resnick similarity) and wup_similarity (Wu and Palmer conceptual similarity).

Topic Modeling

One way to explore large quantities of text is to try to discover "topics", in the form of "recurring patterns of co-occurring words.". This idea has gotten a lot or press in the digital humanties, for example, in this Journal of Digital Humanities introduction:
http://journalofdigitalhumanities.org/2-1/dh-contribution-to-topic-modeling/

and in this tutorial article:
http://journalofdigitalhumanities.org/2-1/topic-modeling-a-basic-introduction-by-megan-r-brett/

and this blog on interpreting results:
http://miriamposner.com/blog/?p=1335

and in this Ted Underwood blog post using topic modeling to understand a journal archive.
http://tedunderwood.com/2012/12/14/what-can-topic-models-of-pmla-teach-us-about-

the-history-of-literary-scholarship/

Following the tutorial article by Megan Brett, we can download and install Mallet on our own computers. This doesn't work in the lab because we are not allowed to set environment variables.

To experiment with Mallet, I copied the Presidential Inaugural address corpus from the NLTK corpora directory to another directory where I wanted to work with topic modeling.

Inside the inaugural folder are individual .txt files for each document, from 1789-Washington.txt down to 2009-Obama.txt.

Mallet is run from the command line, so I get a terminal (Mac) or command prompt (PC) window and navigate to the mallet installation directory, where the mallet commands are stored in the bin subdirectory. First, we execute the import-dir command to prepare the data, where I am showing the Mac syntax for the command, and showing the path to the data directory. The command is shown on multiple lines here, but you would actually put it all on one line.

```
./bin/mallet import-dir
--input /Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/inaugural
--output /Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/president.mallet
--keep-sequence --remove-stopwords
```

This produces a single input file which I called president.mallet. Next, we give the command to find the topic model:

```
./bin/mallet train-topics
--input /Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/president.mallet
--num-topics 20 --optimize-interval 20
--output-state /Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/topic-state.gz
--output-topic-keys
/Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/president_keys.txt
--output-doc-topics
/Users/njmccrac1/AAAdocs/NLPfall2013/labs/mallet/president_compostion.txt
```

I specified the number of topics to be 20 and asked it to automatically optimize parameters. A number of outputs are generated, including the list of keywords for each topic (the topics are not named), and the composition of topics in each document. We can load the latter file into Excel for easier viewing. Note that we can experiment with a number of different topics, or there is an additional option to do that automatically.

While Mallet appears to be the most widely used package now for topic modeling, there is also the Stanford topic modeler with a GUI.

More Memory for Weka

Previously, we ran Weka in the lab by going to the weka directory and double-clicking on the jar file.

Instead, open a command prompt (or terminal window) and navigate to the weka folder that contains the jar file. At the prompt, we want to run the java command and give a larger memory size. For example, to ask for 2GB memory:

```
%  java -Xmx2024m -jar weka.jar
```

However, in the lab, you have to give the complete path to the java command:

```
% "C:\Program Files (x86)\Java\jre7\bin\java" -Xmx2024m -jar weka.jar
```

Loading SPAM/HAM files

Look at the program detectSPAM.py which should get you started to create a spam detection classifier by showing how to load the Spam and regular (ham) emails.