NLP Lab Session
Week 14, April 24, 2013
Semantics: WordNet similarity in NLTK and LDA in Mallet

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.

Labweek14semantics.py

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
```

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

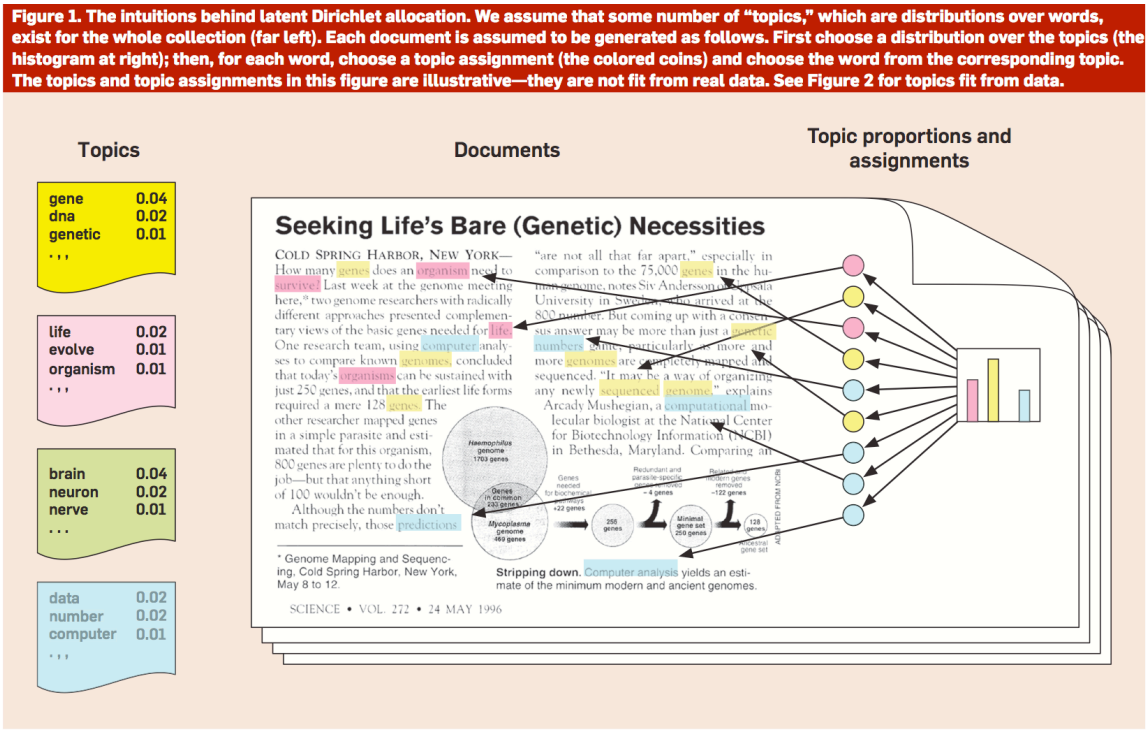
One way to explore large quantities of text is to try to discover "topics", in the form of "recurring patterns of co-occurring words.". This idea has gotten a lot or press in the digital humanties, for example, in this Journal of Digital Humanities introduction: http://journalofdigitalhumanities.org/2-1/dh-contribution-to-topic-modeling/

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/

In the original article by David Blei, Probabilistic Topic Models, he describes that each topic can be thought of as a distriution of topics, where each topic consists of a distribution of words related by co-ocurrence.



The Latent Dirichlet Allocation algorithm discovers these distributions from unstructured text. One popular implementation is Mallet, which I have downloaded and run from the command line. But more recently, Stanford has made available an on-line viewer of the results of Mallet, the Stanford Topic Modeling Toolbox: http://nlp.stanford.edu/software/tmt/tmt-0.4/.

Note that in order to get good results from Mallet, one needs to choose the number of topics to obtain and to tune some parameters that govern the running of the algorithm as in this description from Ted Underwood:

"MALLET also has a "hyperparameter optimization" option which Goldstone's 100-topic model above made use of. Before you run screaming, "hyperparameters" are just dials that control how much fuzziness is allowed in a topic's distribution across words (beta) or across documents (alpha). Allowing alpha to vary allows greater differentiation between the sizes of large topics (often with common words), and smaller (often more specialized) topics. (See "Why Priors Matter," Wallach, Mimno, and McCallum, 2009.) In any event, Goldstone's 100-topic model used hyperparameter optimization; Underwood's 150-topic model did not. A comparison with several other models suggests that the difference between symmetric and asymmetric (optimized) alpha parameters explains much of the difference between their structures when visualized as networks. "