

# CIS 430/530 Fall 2011 HW 4

Instructor: Ani Nenkova

TA: Alexander Shoulson

Released: November 10, 2011

Due: 11:59PM November 20, 2011

## Overview

This assignment will familiarize you with the use of WordNet for analyzing the ambiguity, polysemy, and specificity of individual words. This assignment greatly benefits from breaking each assigned function down into a series of helper functions, as many components are reused throughout the assignment.

**All sample output is for example purposes only and does not reflect actual results.**

## Deliverables (Total 105 Points)

You will be asked to submit the code for the functions you have implemented as well as a brief writeup of your observations and results and a visualization. Write ample comments in your code. In addition, this assignment requires you to also submit a graph you have made with the visualization toolkit.

**Make sure that your function names and the order of their parameters exactly match the `func(param1, param2, ...)` signatures specified in the assignment.**

## Submitting your work

The code for your assignment should be placed in a single file called `hw4_code_yourpennkey.py` where `yourpennkey` is your Penn Key. The writeup should be in **plain text format** and named as `hw4_writeup_yourpennkey.txt`. Since my (Alex) pennkey is “shoulson”, I would submit the following files:

```
hw4_code_shoulson.py, hw4_writeup_shoulson.txt
```

**Write your and your partner’s name and pennkey at the top of both your code and writeup.**

To electronically submit homework, if you are not already working on Eniac, you need to place the file containing your solution on your SEAS account storage. One way to do this would be to use an SFTP client such as FileZilla or WinSCP.

Then connect via ssh to `seas.upenn.edu` and use the `turnin` command to submit your files for grading:

```
turnin -c cis530 -p hw4 hw4_code_yourpennkey.py hw4_writeup_yourpennkey.txt
```

This should print out a confirmation message. If you are prompted for a section name, type `ALL`. You can run `turnin` multiple times before the deadline. Each time you run `turnin`, it overwrites your previous submission for that assignment and changes your submission time (homework resubmitted after the deadline will appear late). You can check that the homework was submitted successfully:

```
turnin -c cis530 -v
```

This will show you the list of file(s) you have submitted.

## Code Guidelines

You can use in-built NLTK and matplotlib modules whenever possible unless specified otherwise. However, only import the modules that you are actually using. For example:

```
from nltk import * # Bad!
from nltk import FreqDist, ConditionalFreqDist # Good!
```

You may write any extra helper functions that you think are necessary, but all the functions defined in this document should be present. Please add descriptive comments to all functions that you write. Do not do any preprocessing/filtering of data unless explicitly mentioned. For example, the words of a set of documents, refers to all tokens, even non-alphabetic ones.

**Do not leave any code statements in the body of your file**, with the exception of global variables (if you choose to use any). Place any test or execution code in your `if __name__ == '__main__':` block. The rest of your code should be in the functions you write for this assignment, or in helper functions.

## Data

We will be using the same data from Homework 3: sets of articles and summaries. There are five article collections, each containing ten news articles each discussing the same topic or event (50 articles total). These articles were part of the 2004 DUC summarization task, which is described in more detail at <http://duc.nist.gov/duc2004/>. We are also giving you four summaries for each article collection (20 summaries total). Each of these summaries was human-written and comprises information from all ten articles in the collection it summarizes.

On eniac, The article collections can be found at `/home1/c/cis530/data-hw3/articles`  
Similarly, the summaries can be found at `/home1/c/cis530/data-hw3/summaries`

The article collections are titled `d30006t`, `d30008t`, `d30037t`, `d30047t`, and `d31013t`.

## Readings

Reading chapter 2 the NLTK book, particularly the section on WordNet will be helpful for this assignment. Also read the functions and variables in the following NLTK modules:

```
http://nltk.googlecode.com/svn/trunk/doc/api/nltk.corpus.reader.wordnet-module.html
http://nltk.googlecode.com/svn/trunk/doc/api/nltk.corpus.reader.wordnet.Synset-class.html
http://nltk.googlecode.com/svn/trunk/doc/howto/wordnet.html
```

## 1 Polysemy and Specificity (30 Points)

For this problem, you should write functions that return the most polysemous and the most specific words from a generated TopicWords file (as you created in Homework 3).

## 1.1 Polysemous Words (15 Points)

Write two functions, `get_most_polysemous` and `get_least_polysemous`. The function `get_most_polysemous(n, word_list, part_of_speech)` returns the `n` most polysemous words from a given list of words. For example, if part of speech is 'noun', then return the `n` most polysemous nouns. Other possible values for part of speech are 'verb', 'adjective' and 'adverb'. Similarly, `get_least_polysemous(n, word_list, part_of_speech)` returns the `n` least polysemous words in the word list. Only return words that have at least one synset for the given part of speech.

```
>>> word_list = load_collection_words('/path/to/collection/directory')
>>> word_list
['The', 'Maryland', 'Transit', 'Administration', 'released', ...]
>>> get_most_polysemous(5, word_list, 'noun')
['opponent', 'part', 'agent', 'movement', 'animal']
```

## 1.2 Specific Words (15 Points)

Now, you will find which are the most specific *nouns* in this category. We will try a simple way of computing specific words. We will look at how far down the hypernym hierarchy a given noun is. If the distance of a noun to the root of the hypernym tree (entity) is large, then this noun is more specific compared to one that is closer to the root. Implement this functionality in `get_most_specific(n, word_list)`, which returns the most `n` specific *nouns* in a given word list. Similarly, `get_least_specific(n, word_list)` returns the least specific `n` words.

### NOTE:

1. This problem asks you to compute specific words using a simple heuristic on the hypernym tree. You have studied a more sophisticated method in the lecture where the hierarchy was combined with word probabilities from a corpus. Here, we will make some simplification and only use WordNet to compute the specific words.
2. Sometimes, there will be multiple paths up the hypernym tree. In this case, choose the shortest path as the path on which you compute the specificity score. See the NLTK book pg. 70 for an example of multiple paths.
3. If a word does not have a hypernym, it should be ignored from the calculation. You do not need to compute specificity for such words. Take care when you implement this. A hypernym path returned by NLTK functions contains the synset for the given word together with other hypernyms. So if the path length is one, it would be the case that there are no hypernyms and so you should neglect such paths.

```
>>> word_list = load_collection_words('/path/to/collection/directory')
>>> get_most_specific(5, word_list)
['slaying', 'jail', 'dentist', 'flounder', 'human']
```

## 2 Word Similarity (40 Points)

- a) (10 Points) You have learned how to compute the similarity between two words using the shortest path between them through the hypernym tree. This similarity method is implemented in NLTK as the function `path.similarity` in the `Synset` class. Write a function `get_similarity(word1, word2)` that computes the word similarity between two *nouns*. Each word can have multiple synsets to which it belongs. Pick the synset for each of the two words that yields the **maximum** similarity.

```
>>> get_similarity('operation', 'find')
0.111111111111
```

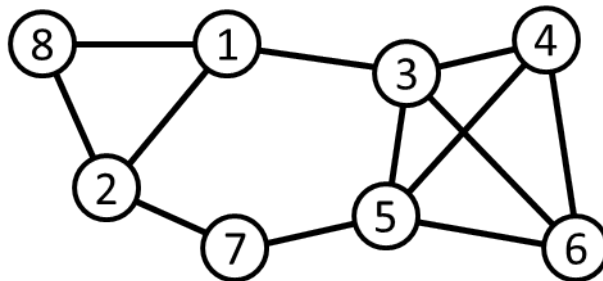
- b) (5 Points) Write a function `get_all_pairs_similarity(word_list)` which, given a list of noun strings, computes a similarity value between every pair of nouns in the list. You can assume that only words that are nouns will be provided in the `nounlist` argument. (Although words with other parts of speech may appear in the list, you can assume that all words given are nouns.) The return value is a list of strings; each string has the format `('word1', 'word2', similarity_value)`.

```
>>> get_all_pairs_similarity(['settlement', 'camp', 'base', 'country'])
[('settlement', 'camp', 0.25), ('settlement', 'base', 0.2), ('settlement', 'country',
0.3), ('camp', 'base', 0.16), ('camp', 'country', 0.2), ('base', 'country', 0.25)]
```

- c) (5 Points) Write a function `filter_pairs_similarity(pair_list, minimum)` that takes a list of word similarities, formatted as `('word1', 'word2', similarity_value)`, and returns only those that meet the minimum similarity given, in the same format.

```
>>> pairs = get_all_pairs_similarity(['settlement', 'camp', 'base', 'country'])
>>> filter_pairs_similarity(pairs, 0.25)
[('settlement', 'camp', 0.25), ('settlement', 'country', 0.3), ('base', 'country',
0.25)]
```

- d) (20 Points) Write a function `get_similar_groups(word_list, minimum)`. This function will take a `word_list`, compute the pairwise similarity, filter that list to remove any similarities below the `minimum`, and return a list of maximal cliques (with at least 3 vertices) within the resulting graph. A clique is expressed as a list of the included vertices. For instance, the following graph has cliques `[1, 2, 8]` and `[3, 4, 5, 6]`. Use the Bron-Kerbosch algorithm ([http://en.wikipedia.org/wiki/Bron-Kerbosch\\_algorithm](http://en.wikipedia.org/wiki/Bron-Kerbosch_algorithm)) for finding maximal cliques.



```
>>> word_list = load_collection_words('/path/to/collection/directory')
>>> get_similar_groups(words, 0.25)
[['front', 'conspiracy', 'interest'], ['front', 'home', 'side'], ['attacks',
'operation', 'performing'], ['operation', 'performing', 'find'], ['operation',
'performing', 'interest'], ['demonstrators', 'protester', 'protesters'],
['witness', 'anti', 'protester', 'protesters', 'opponents']]
```

### 3 Writeup: Analysis and Discussion (35 Points)

#### 3.1 Polysemous and Specific Topic Words (20 Points)

For this question, we will be analyzing the document collection `d31013t`, for which you should have generated a `TopicWords .ts` file in Homework 3. Be sure to read the 10 articles and familiarize yourself with their contents and the topic of the collection.

- a) List the top 10 polysemous *topic* nouns, verbs, adjectives and adverbs.
- b) For how many of these polysemous words do you think the first sense in WordNet (or the most frequent) could be the one intended for the topic of these 10 articles? Do you think having these polysemous words pose any problems while interpreting them with respect to this topic?
- c) What are the 10 least polysemous *topic* nouns, verbs, adjectives, adverbs?
- d) List the 10 most specific words and 10 least specific topic words.
- e) Do you agree with their specificity ratings?
- f) Compute the 20 most specific and least specific words for the *entire* collection (not just topic words). How do these lists compare with those of just the topic words?

### 3.2 Word Similarity (15 Points)

- a) Use the function `compute_similar_groups` for each of the five collections' *topic words* and include the output cliques in your writeup. Use a `minimum` value of 0.25.
- b) What do you notice about these similarity groupings? Are there any surprising cliques? Do you notice any trends?
- c) Which are the most and least interesting word groupings?
- d) Are there any surprising or uninformative groupings? Pick one such grouping, examine the synsets for each word, and try to justify the similarities within that group.