CS 297 Report

Incorporating WordNet in an Information Retrieval System

by

Shailesh Padave

(shaileshpadave49@gmail.com)

Fall 2013

Advisor : Dr. Chris Pollett

Department of Computer Science

San Jose State University

TABLE OF CONTENTS

Introduction:

The idea of this project is to incorporate WordNet in an Information Retrieval System. The idea is to use WordNet output in a query rewriting algorithm for query expansion. WordNet is a large lexical database of English. Basically in WordNet, noun, verbs, adjectives, and adverbs are grouped together into sets of cognitive synonyms. Synonyms are also known as synset where each synset expresses distinct concept.[1] WordNet is developed at Princeton University. George A. Miller began this project in mid-1980's and now it is housed in Department of Computer Science.

In this semester, I worked on various experiments on WordNet. I tried to understand how WordNet works internally and learnt command line instructions for WordNet. This report provides a brief description of work done in CS297 Preparation for Master's writing project.

The work was divided into four deliverables. The first deliverable was to download WordNet on the local system and learn on which different kinds of queries WordNet supports. The second deliverable was to implement Cosine Similarity Ranking Algorithm. The goal of this deliverable was to extract synonyms from the given senses from WordNet. The third deliverable was to use the code for Part of speech tagging with the output of WordNet. The fourth deliverable was to implement an intersection ranking algorithm and cosine similarity ranking algorithm and find out suitable method for calculating similarity ranking. In some of the cases, the cosine similarity ranking gave unexpected result so the comparison between two methods helped us to get expected result. We now describe each of these deliverables in more details followed by the future work in CS298 and the conclusion.

2. Deliverables

Deliverable 1 :

Yioop is an open-source, distributed crawler and search engine written in PHP. It is designed to allow users to produce indexes of web-site. Yioop comes with its own extendable model-view-controller framework that you can use directly to create new sites that use Yioop search technology. This framework also comes with a GUI which makes it easy to localize strings and static pages. Yioop can be configured as either a general purpose search engine for the whole Web or it can be configured to provide search results for a set of URLs or domains. [3]

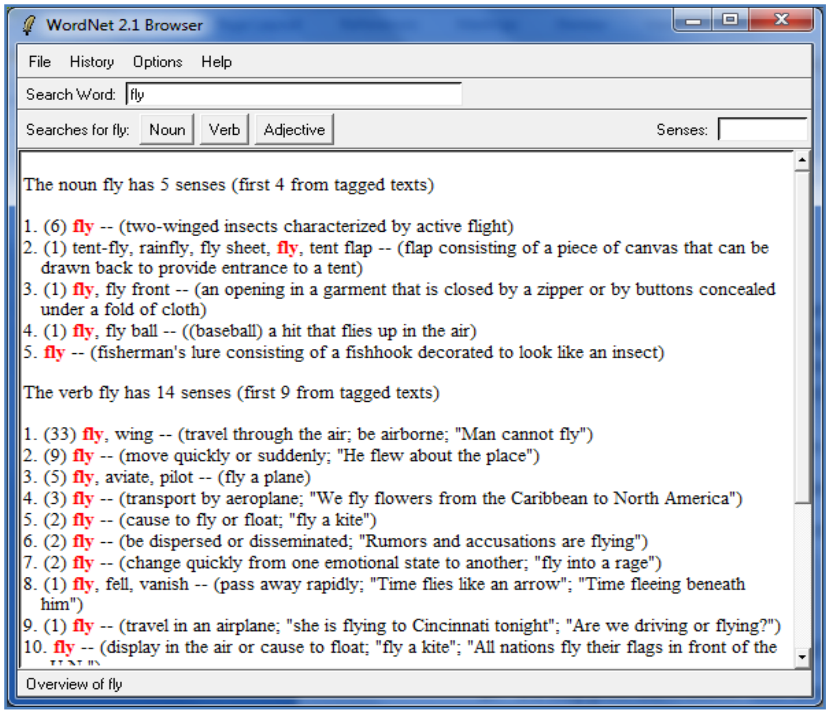
WordNet is a large lexical database of English. It groups English words into sets of synonyms called synset and provides short, general definition and records the various semantic relations between these synonyms. [2]

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words; whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.[1]

The purpose of WordNet is twofold[2]:

1. To produce a combination of dictionary and thesaurus
2. To support automatic text analysis and artificial intelligence applications

Figure 1 shows screenshot for WordNet is as follows:

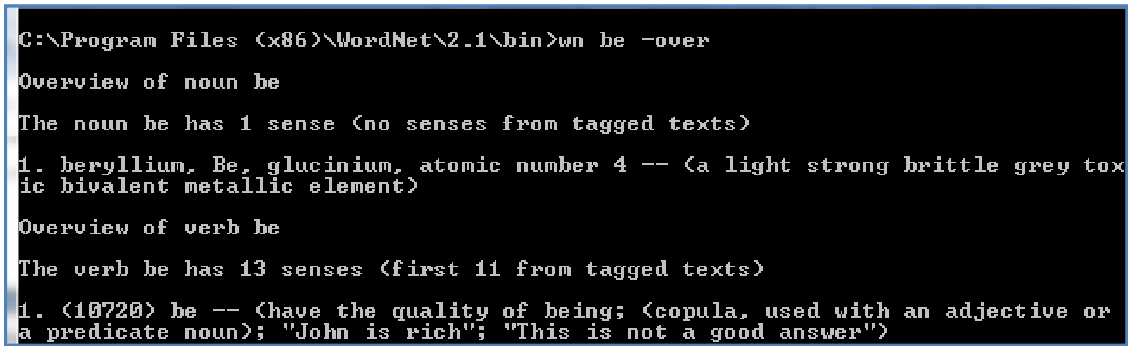

WordNet 2.1 Browser
File History Options Help
Search Word: fly
Searches for fly: Noun Verb Adjective
Senses:
Overview of fly

```
The noun fly has 5 senses (first 4 from tagged texts)

1. (6) fly -- (two-winged insects characterized by active flight)
2. (1) tent-fly, rainfly, fly sheet, fly, tent flap -- (flap consisting of a piece of canvas that can be
   drawn back to provide entrance to a tent)
3. (1) fly, fly front -- (an opening in a garment that is closed by a zipper or by buttons concealed
   under a fold of cloth)
4. (1) fly, fly ball -- ((baseball) a hit that flies up in the air)
5. fly -- (fisherman's lure consisting of a fishhook decorated to look like an insect)

The verb fly has 14 senses (first 9 from tagged texts)

1. (33) fly, wing -- (travel through the air; be airborne; "Man cannot fly")
2. (9) fly -- (move quickly or suddenly; "He flew about the place")
3. (5) fly, aviate, pilot -- (fly a plane)
4. (3) fly -- (transport by aeroplane; "We fly flowers from the Caribbean to North America")
5. (2) fly -- (cause to fly or float; "fly a kite")
6. (2) fly -- (be dispersed or disseminated; "Rumors and accusations are flying")
7. (2) fly -- (change quickly from one emotional state to another; "fly into a rage")
8. (1) fly, fell, vanish -- (pass away rapidly; "Time flies like an arrow"; "Time fleeing beneath
   him")
9. (1) fly -- (travel in an airplane; "she is flying to Cincinnati tonight"; "Are we driving or flying?")
10. fly -- (display in the air or cause to float; "fly a kite"; "All nations fly their flags in front of the
```

Figure 1

When WordNet searches for any word, the output will be given with the four types of word such as – noun, verb, adjective and adverb. We can get the output using the software as well as from command line arguments. There are different commands for different output types.

Currently we are looking for the output same as shown in Fig. 2. For this, we used the following command for input word 'be'



```
C:\Program Files (x86)\WordNet\2.1\bin>wn be -over

Overview of noun be

The noun be has 1 sense (no senses from tagged texts)

1. beryllium, Be, glucinium, atomic number 4 -- (a light strong brittle grey tox
ic bivalent metallic element)

Overview of verb be

The verb be has 13 senses (first 11 from tagged texts)

1. (10720) be -- (have the quality of being; (copula, used with an adjective or
a predicate noun); "John is rich"; "This is not a good answer")
```

Figure 2

All the information we got is extracted from database. The database information for WordNet is as shown in below table:

Types of word	**Files**
Adjective	data.adj , index.adj
Adverb	data.adv , index.adv
Noun	data.noun , index.noun
Verb	data.verb , index.verb

Now we will see how the data is stored in data.verb file.

00048819 29 v 01 habit 0 002 @ 00047662 v 0000 + 03479089 n 0101 01 + 09 00 | put a habit on

- *synset_offset* – Current byte offset in the file (8 digit)
- *lex_filenum – (2 digit)* lexicographer file name containing the synset
- *ss_type – n,v,a,s,r*
- *w_cnt – number of words in synset (2 digit HEX)*
- *word – Actual search word*
- *gloss – represented as vertical bar followed by text string. May contain 1 or more examples*

Information about data.verb

castilian n 1 1 @ 1 0 06979859

- *lemma – lower case word*
- *pos – n v a r*
- *synset_cnt – number of synset in that lemma*
- *p_cnt – number of pointers*

Deliverable 2:

This deliverable involved cosine similarity ranking algorithm. The cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. [4]

In cosine similarity ranking, we maintain query vector and document vector. All the unique words from document vectors are collected together. The size of query vector and document vector is equal to the size of dictionary. We can measure closeness between the two vectors by taking their dot product.[6]

To calculate θ ,we have the following equation:

$$\cos\theta = \frac{\sum_{i-1}^{V} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{|V|} x^2}\sqrt{\sum_{i=1}^{|V|} y^2}}$$

when $\theta = 0°, \cos\theta = 1$ and two vectors are collinear and if $\theta = 90°, \cos\theta = 0$ and two vectors are orthogonal.

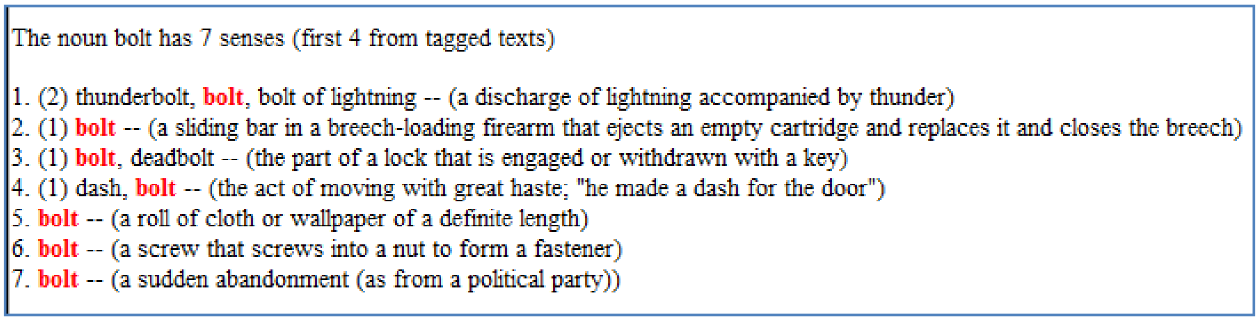
Consider we have a query vector $\vec{q}$ and document vector $\vec{d}$, and similarity is defined as the cosine of the angle between them[6]. i.e.

$$sim(\vec{d}, \vec{q}) = \frac{\vec{d}}{|\vec{d}|} \cdot \frac{\vec{q}}{|\vec{q}|}$$

With help of this similarity ranking algorithm, we will find out the synonyms for the given word from query according to the part of speech. Basically, WordNet output contains the synonym for the word along with the word usage in that sentence. Now to figure out which is the proper word from the given senses we used cosine similarity ranking algorithm.

e.g. : Consider we have query as – *running bolt*

If we try to search synonyms for word 'bolt' we will get output as shown in Fig. 3:



The noun bolt has 7 senses (first 4 from tagged texts)

1. (2) thunderbolt, **bolt**, bolt of lightning -- (a discharge of lightning accompanied by thunder)
2. (1) **bolt** -- (a sliding bar in a breech-loading firearm that ejects an empty cartridge and replaces it and closes the breech)
3. (1) **bolt**, deadbolt -- (the part of a lock that is engaged or withdrawn with a key)
4. (1) dash, **bolt** -- (the act of moving with great haste; "he made a dash for the door")
5. **bolt** -- (a roll of cloth or wallpaper of a definite length)
6. **bolt** -- (a screw that screws into a nut to form a fastener)
7. **bolt** -- (a sudden abandonment (as from a political party))

Figure 3

So now we have to find out cosine similarity for each given sentence from seven senses of noun bolt with given query as 'running bolt'.

Deliverable 3:

Part-of-speech tagging also known as grammatical tagging or word-category disambiguation, is a process of marking up a word in a text(corpus) according to this a particular part of speech. This based on both its definition as well as context i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. [5] Tags are the grammatical parts of speech that the words fall into, the traditional noun, verb, adjective and adverb.

Our code reads the query provided by user and it tags each word with its particular part of speech. This is useful to figure out to from which part of WordNet output we should pick up the senses. This we reduce the time to find synonyms for given query.

In absence of part-of-speech tagging, we have to find out cosine similarity ranking with each sentence from WordNet output. Part-of-speech tagging code will help us simplify the logic and time to get output.

The tagger was trained by analyzing a corpus and noting the frequencies of the different tags for a given word. As words were tagged they were assigned to the most frequent tag for the word if it was in the corpus, or tagged as a noun if not. Then a series of transformations were applied, which changed the tag depending on various conditions. The results were compared to known correct tags, and the rules that added the most accuracy retained.[7]

e.g. : Sam is running behind the dog.

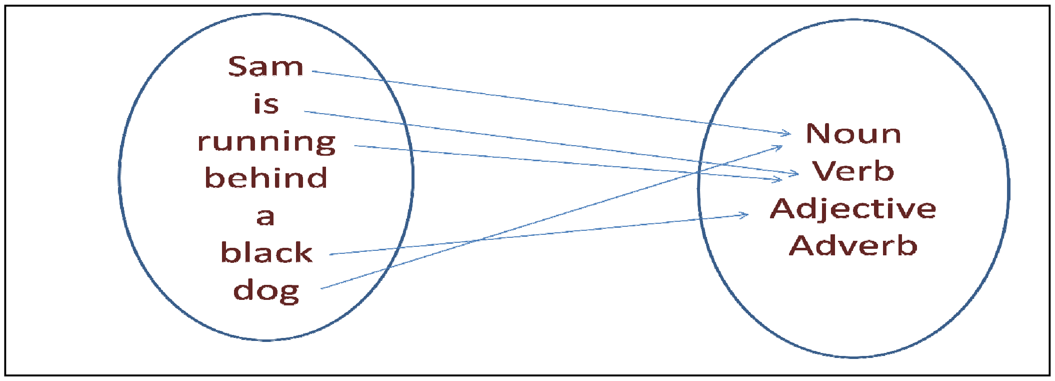


Figure 4

Output will be as follows:

Sam_NN is_VB running_VB behind a dog_NN.

Deliverable 4:

Intersection method is another way to find out the similarity between the sentences. For cosine similarity ranking, in some cases, we may get IDF as 0 which leads the total score zero. So another experiment was carried out with intersection ranking algorithm. We are using an intersection function to calculate the intersection between two sentences.

In the intersection method, two sentences have a good intersection then they holds the same information and a sentence has a good intersection with many other means that sentence holds some information from each one of them. We may say that it is key sentence in our text.

In this method, we will find out the tokens from given query and sentence. This method will try to find out the similar tokens from the both sentences. More similar vectors, more will be the score.

We found that this is useful method than cosine similarity ranking. As in cosine similarity ranking, we use TF (term frequency) and IDF (Inverted Document Frequency). In our code, we used to have two or more sentences in an array, and when try to find out the similarity between one sentence with another sentences from array, it will calculate TF and IDF. If a word occurs one time in an array, then according to IDF, log(1) is zero. In cosine similarity ranking technique, we take product of TF and IDF, which will eventually become zero. Since, a word occurred once on the sentence then we should get some similarity with the sentence from document vector and query sentence from query vector. On the other hand, intersection method will provide similarity if a word from query sentence occurred once in document sentence. This situation will repeat many times in our implementation, so we found intersection method is more useful than cosine similarity ranking technique.

Conclusion:

WordNet has no direct API which supports PHP. In this semester, we studied WordNet and we found some command line arguments which gave us the very useful output for our project. We extracted the required data from output.

WordNet extracts all the information from database. In database, it contains two files each for noun, verb, adjective and adverb with extension as .data and .index. So WordNet refers totally eight files for generating output. Whenever we get the output from WordNet, for each type, WordNet creates senses. We are going to use sentence or sentences provided for each sense to figure out perfect synonym for word given in query. We figured out that this is efficient way to expand the query using WordNet.

This project gave me good understanding of WordNet's internal structure and how it works. The command line arguments are more beneficial to extract the data from generated output. I learned PHP language very well in this semester. The part-of-speech tagging helped us to reduce our work to find similarity with each sentence from output. We used cosine similarity ranking and intersection method to find out similarity between two sentences.

In next semester, I will work to integrate my developed code into Yioop!. I will get the summary from Yioop! for any website. I will use the part-of-speech tagging on each sentence. Afterwards, each sentence from the summary will be tagged with verb, noun, adjective or adverb as per its tagging in given sentence. After this process, stemmer will stem the given input. I have to hide that tagging provided by part-of-speech tagging from stemmer.

Once the stemming process is done, WordNet will work on each sentence from summary. We will pick up each sentence from summary. WordNet will work on each word in summary. First we will see part of speech tag attached to it. Then for that word we'll request WordNet to get the output. According to the part of speech tagging, we will extract only those senses from the output of WordNet which are matched with tagging provided by part-of-speech tagging. Then we will find out similarity between sentence/sentences given by WordNet in example for each sense with sentence provided in the summary. The synonyms with the highest scored sentence will be selected and those words will be used to expand the query.

We figured out the query expansion technique using WordNet . After using the synonyms provided by WordNet, Yioop! will calculate the page rank with the new modified query. Yioop! will compare it with result provided by original query result and results provided by new expanded query result. The highest ranked pages will be showed first.

This concludes the description of the deliverables of my 298 project.

References:

[1] *WordNet: A lexical database for English* Retrieved on Dec 1, 2013, from website: http://wordnet.princeton.edu/

[2] *Wikipedia for WordNet* Retrieved on Dec 1, 2013, from website: http://en.wikipedia.org/wiki/WordNet

[3] *Yioop Documentation from SeekQuery* Retrieved on Dec 1, 2013, from website: https://seekquarry.com/?c=main&p=documentation#overview

[4] *Wikipedia for Cosine Similarity Ranking* Retrieved on Dec 1, 2013, from website: http://en.wikipedia.org/wiki/Cosine_similarity

[5] *Wikipedia for Part of Speech Tagging* Retrieved on Dec 1, 2013, from website: http://en.wikipedia.org/wiki/Part-of-speech_tagging

[6] *Cosine Similarity* Retrieved on Dec 10, 2013, from website: http://www.cs.sjsu.edu/faculty/pollett/267.2.12f/Lec12092012.html#(9)

[7] *Part of speech tagging* Retrieved on Dec 10, 2013, from website: http://phpir.com/part-of-speech-tagging