NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF HUMANITIES AND SOCIAL SCIENCES

Acquiring Predominant Word Senses in Multiple Languages

Aiden Lim Si Hong

Matriculation No.: U1030022B

Supervisor: Associate Professor Francis Bond

Date: 21 April 2014

A Final Year Project submitted to the School of Humanities and Social Sciences, Nanyang Technological University in partial fulfilment of the requirements of the Degree of Bachelor of Arts in Linguistics and Multilingual Studies.

Acknowledgements

This project was almost certainly beyond the abilities of an undergraduate linguistics student with only a passing familiarity with computational linguistics to attempt alone. I would like to express my thanks to my supervisor Associate Professor Francis Bond for introducing me to this project, giving me so much help and guidance in developing this program and never once getting angry at all the silly questions I had. My gratitude to Dr. Wang Huizhen, who was always willing to discuss any computing problems I had and was always patient with any missteps I made. Finally, to all my friends, thank you for bearing with me when I started throwing out computer terms when discussing my fyp.

Contents

1 Introduction **1**

2 Overview of Word Sense Disambiguation and the Various Possible Approaches **3**

2.1 WSD Systems and the Various Approaches 4

2.1.1 The First Sense use as an Approach and Baseline 5

2.1.2 WSD System Design Approach 5

2.2 WSD Systems and Accuracy 8

2.3 Examples of WSD Systems 9

2.3.1 Heuristic-Rule Word Sense Disambiguation (HR-WSD) 9

2.3.2 CFILT 10

2.3.3 Clustering-By-Comittee (CBC) 11

3 Methodology **13**

3.1 Pre-processing the Data to form a Corpus 14

3.1.1 Discussion 15

3.2 Distributional Similarity Score 16

3.3 Semantic Similarity Score 19

3.3.1 Wu-Palmer Similarity 19

3.3.2 Leacock-Chodorow Similarity 20

3.3.3 SSS Results 20

4 Results **22**

4.1 Prevalence Scores 22

4.2 Examples 23

5 Evaluation and Discussion **26**

6 Conclusion **29**

Abstract

The accurate retrieval of the most appropriate sense of a word, in other words Word Sense Disambiguation, is a key process in machine learning and machine translation. Yet oftentimes, the reliabilty and accuracy of a system is a product of the input data which is itself dependent on financial and temporal restrictions. The resulting scarcity also affects the scope of WSD systems by limiting available data to specific languages. This paper proposes a Word Sense Disambiguation system that ignores these restrictions by being able to accept raw text input from any language and extract predominant word senses from it. This is done through a two-pronged approach of mathematically mapping the distribution of a target word and its neighboring words before calculating the semantic similarity of both. By using both the Distributional Similarity Score and Semantic Similarity Score, it becomes possible to mathematically calculate and propose a Predominant Word Sense without any language-specific input. This paper reports how the system was used on English and Mandarin Chinese Wikipedia raw text to extract Predominant Word Senses with a fair degree of accuracy for a successful Word Sense Disambiguation System

1 Introduction

Language production occurs on several different levels. To form even a simple sentence, one has to activate several layers of language components, the phonological, syntactical and semantic component layers, and cross-reference that with the users lexicon and grammatical constructions (Croft & Cruse, 2004, :247). To apply a different terminology to such a model, the interaction of the lexicon and semantic component can be argued to be a word-sense construction. When shifted to the field of computational linguistics and natural language processing, determining and defining the heuristic predominant sense of the word; in other words, Word-Sense Disambiguation (henceforth WSD), becomes one of the main areas of development. Accurate WSD allows for improvement in systems that focus on information retrieval, summarization and machine translation(Boyd-Graber et al., 2007). For programming purposes, a fundamentally different approach to WSD has to be adopted as compared to a human being since the interaction between the context and word are too complex for a computer to easily process without manual user intervention (McCarthy et al., 2004). In this paper, we attempt to utilize the McCarthy et al. (2007) method of acquiring the predominant word sense of a given word from raw text data and further extend it to be used on different languages.

To build a program to arrive at a predominant word-sense for a specific word, several processes have to be built and run to enable some degree of accuracy and reliability in postulating a sense. First, a suitable data set has to be selected to expose the program to enough data to build a statistical model for prediction. For this paper, we elected to use Wikipedia[1] as our raw text data set. The raw text data then had to be preprocessed to create a corpus to train our program. This corpus could then be used to run various mathematical processes to produce quantifiable variables that could be calculated to predict a word-sense. These various steps are explained in greater detail in the methodology and results sections later in this paper.

With this approach of training on language-specific data for a particular language while using mathematically based algorithms to build a statistical model, it is hoped that this program could be applied on a multi-language basis. McCarthy et al. (2007) worked only with English due to the wide array of

[1] URL: http://dumps.wikimedia.org/backup-index.html

training data and evaluation standards available. This paper and WSD system proposed hopes to adapt their theoretical framework and assumptions into an original WSD system that is not dependent on any language-specific input. Due to the timeframe this project has to be completed in, we decided to limit the current input data to English and Mandarin Chinese. However, since one of the intended goals was multi-language application, the processes of the WSD system were designed in such a way that was language-independent. With sufficient accuracy and reliability, it would then become possible for a single program to take in raw text data from any language, process that data and be able to carry out WSD for that language. Rather than having to write new programs and processes whenever working on a new language, a system such as proposed in this paper would have the flexibility to be adapted to any language desired thereby reducing financial and temporal costs in WSD-related endeavours. This paper will review the different approaches possible when building WSD systems, the approach we selected and its accompanying processes and the results of the system designed.

2 Overview of Word Sense Disambiguation and the Various Possible Approaches

'No man is an island' and in the same vein, no word can stand alone. Even simple one-word utterances have an underlying context that allows for interpretation and comprehension (Firth, 1957). Analyzing a single word is easily within a Natural Language Processing (NLP) programs' capabilities; however, parallel processing of the word-sense and context requires more complex systems. For example, a basic Part-of-Speech (POS) tagger when faced with the phrase 'The problem dogs the sailor' would result in the parse:

the_DT problem_NN / dogs_NNS / the_DT sailor_NN

Yet a human reader would draw from contextual prompts to identify 'dogs' as a verb in the sense of **bother** or **trouble**. Compared to that, an unsupervised machine-learning process would use the heuristic predominant sense of 'dog' as a noun in the sense of **canine, four-legged mammal**. Applied to large databases of text, the occurrence of specific, low frequency polysemy would be beyond most WSD systems to accurately parse without making use of the word context. To fully realize machine-learning and machine-translation, an accurate WSD system has to go beyond basic parsing and word-sense association.

Another factor for WSD is the domain sensitive-nature of senses. For example, Barclay et al. (1974) used *pianos are heavy* to illustrate how in one domain could be understood as **moving furniture** while in a music- specific domain as related to the **musical style of the piano**. At the same time, there is merit to the argument that a word-sense is user-generated to fit the subject on hand and establish the corresponding context, i.e. a Topic Model approach that predicts a common underlying theme in a collection of corpora (Boyd-Graber et al., 2007). Regardless, for the purposes of this paper, word and sense must be paired to utilize the wordlist of distribution-similar words and their respective semantic components. Take the phrase

Some party members stayed in the mainland[2]

as an example. If the target word was set as *party*, accurate WSD would involve first: identifying the target word as a **noun** instead of **verb**; second: associating *party_n* with the various possible senses and

[2]http://en.wikipedia.org/wiki/Kuomintang and part of the WikiCorpus data used in this paper

third: acquiring **political party** as the predominant word-sense. Our system can successfully carry out these processes; however, the critical third step can be achieved through various different methods which are worth considering.

2.1 WSD Systems and the Various Approaches

The critical step of pairing a target word with the corresponding predominant word-sense is the focus of much testing and development. In the following sections, this paper will present a rough overview of the different approaches to accomplishing that step and the attendent merits and deficiences. Broadly speaking the approaches can be grouped as dictionary-based (e.g. Oxford English Dictionary) or thesaurus-based e.g. WordNet (Fellbaum, 1998). Each approach can also make use of different types of data-sets. Manual data-sets are hand-annotated documents with markers that the WSD process can identify and work with. Automatic data-sets are documents that are stored and converted into a corpus which is given to the WSD system. The degree of operator input can also affect the accuracy and reliability of the results (McCarthy et al., 2007). Systems can be unsupervised (U) with users taking raw text and starting the WSD process or they can be supervised (S) with some form of user control on the process typically through annotated text.

As mentioned earlier, our system makes use of raw text Wikipedia documents to build a text-based WikiCorpus with minimal conversion work and no sense tagging. The corpus is then given to the machine to run the various designed processes, i.e. an unsupervised WSD system. Since accuracy in WSD is highly dependent on the level of 'granularity' or fineness of the data sets, the most accurate WSD systems are generally those that employ hand-tagged data sets to supervise machine learning (Stevenson & Wilks, 2001). The competing factor here is the availability of comprehensive training sets thereby greatly influencing the overall quality of such semi-automatic systems (Yarowsky & Florian, 2002). For example, SemCor is the largest available sense-tagged corpus with 220, 000 words taken from 103 passages of 2000 words from the Brown Corpus and the adapted text from the novel, *The Red Badge of Courage* (Landes et al., 1998). Approximately half of the corpus is open-class words which are linked to the WordNet senses.

2.1.1 The First Sense use as an Approach and Baseline

The First Sense (FS) refers to the first presented sense of a word in a dictionary or thesaurus. Accessing *party_n* from WordNet[3] gives a FS of **political party**, which is in agreement with the predominant sense of the *party_n* example given earlier. When applied to a WSD system that takes such an approach, one could argue that it is not so much disambiguation as a guess with some statistical support. Such a system is sometimes used as a baseline for comparison purposes to more advanced WSD systems. The FS baseline is heavily dependent on the use of dictionaries and thesauruses in its architecture. Mandarin Chinese would be based on the Chinese WordNet (ZHWN) while English uses the English WordNet (EWN) and Longman Dictionary of Contemporary English (LDOCE) (Procter, 1978). These data sets rank word-senses on their highest level of usage in their respective languages regardless of content. More precisely, WordNets of the various languages base their rankings on the frequency of word usage in their respective corpora (i.e. EWN frequency ranking is based on SemCor (Miller et al., 1993)), while dictionaries are ordered on the lexicographer's intuition (McCarthy et al., 2007).

2.1.2 WSD System Design Approach

Dictionary-based approaches utilize the encoded data in lexical resources, along with supervised machine training of large data sets to seek out possible usage patterns with a fair degree of reliability and accuracy. Taking the earlier *party* example, an accurate dictionary-based system would likely involve hand-tagging *party* with a marker to indicate **political party**. After tagging a data set, a WSD system would then be able to process and distinguish *party* against **political party** and **social event** with varying degress of accuracy and reliability.

Such dictionary-based systems and their calculated results are influenced by various factors. Firstly, the predominant word-sense is context-based with the leading word-sense varying on the context of the data set. Context, domain and genre, among other, are all required and used when determining the predominant word-sense (Magnini et al., 2002). However, most dictionaries only give one reading of a word with no contextual information and seemly at random selection (of course, this is a result of the domains the data set is drawn from). For example, the word 'mole' in the English WordNet (henceforth EWN)

[3]URL: http://wordnetweb.princeton.edu

has the chemistry-specific **gram molecule** while the Oxford English Dictionary provides **small burrowing creature**. With different domains and contexts producing inconsistent and varied frequency counts, dictionary rankings are not reliable indicators of predominant word-sense. Further, the word-senses are fairly rigid with little regard for changes in usage and terminology. For example, EWN provides the first sense of *pipe* as **tobacco pipe** when in modern times one would expect the second sense of **cylindrical tube** to be more accurate (McCarthy et al., 2004) barring various domain-specific usages of pipe. Therefore, a purely dictionary-based input system does not take in account the changing and domain-dependent nature of natural language usage and processes.

Another problem is deriving the predominant word-sense from hand-tagged sources (McCarthy et al., 2007) such as SemCor, which is also the primary sense derivation data set used to build WordNet. With the aforementioned resource and temporal limitations, hand-tagged data sets generally consist of a limited amount of words which makes for a relatively small training corpus. As Kintsch & Mangalath (2011) argues, it is almost impossible for lexical meanings to be accessed fully at any one point in time, notwithstanding that hand-tagged corpora are fairly static after completion with changes in semantic usage over time and culture not able to be incorporated. McCarthy et al. (2007) raises the point that word-sense ranking is based on the limited word frequency found in SemCor. From Figure 1, McCarthy et al. (2007) demonstrated that there are a large number of common usages words contained in WordNet without being present in SemCor, even excluding multi-word expressions.

	WordNet types		BNC types	
PoS	No.	%	No.	%
noun	43,781	81.9	360,535	97.5
verb	4,741	56.4	25,292	87.6
adjective	14,991	72.3	95,908	95.4
adverb	2,405	64.4	10,223	89.2

Figure 1: Quantity and percentage of words (excluding multi-word expressions) in WordNet and British national Corpus (BNC) without representation in SemCor (McCarthy et al., 2007, :558)

For example, the EWN has *cougar (noun)* used to describe **a large American feline**. However, se-

mantic expansion with recent social euphemistic usage adds on **middle-aged woman who enjoys an intimate relationship with a younger male**, a change not present in WordNet. Another example is *carnivorous* (adjective) which one would associate with **animal-eating or meat-eating** instead has a predominant sense of **audacious characteristic**. This seemingly illogical word-sense derivation is due to both senses having exactly equal frequency count in SemCor making it unable to rank the senses. Following from that, many senses with low or no frequency counts are unable to be effectively ranked giving a random sense rather than a true predominant sense.

Despite such problems, dictionary-based WSD systems prove useful when providing general first-sense heuristic disambiguation for words that have higher frequency, in other words common-use words that are less domain-sensitive which explains why the baseline still has a slightly over 1 in 2 chance to give the appropriate first word-sense. One way of utilizing dictionary-based systems is to determine a 'baseline-frequency' that uses the dictionary WSD for high frequency, commonly appearing words. When McCarthy et al. (2007) performed a sense review of polysemous words with greater than single occurrence sense frequency in the evaluation tests Senseval-2 and Senseval-3, it was discovered that the words with senses present in SemCor had a high accuracy rate of predominant sense derivation with Figure 2 summarizing the results.

	1 sense		> 1 sense		
PoS	%	FS = SC FS %	%	FS = SC FS %	Mean polysemy
noun	72.2	52.2	27.8	7.3	5.9
verb	45.6	25.1	54.4	16.9	12.7
adjective	62.9	40.5	37.1	10.3	4.8
adverb	64.7	50.0	35.3	17.6	4.7

Figure 2: Frequent sense analysis of polysemous words in Senseval-2 and -3 with frequency value greater than 1 (McCarthy et al., 2007, :558)

A significant portion have words of a different first-sense as compared to SemCor, a reflection of the limited domain coverage of SemCor. Further, the lower the frequency rate in SemCor, the lower the accuracy of the first-sense heuristic determination. In conclusion, dictionary-based WSD systems demonstrate greater accuracy for high frequency words, yet are inadequate in coverage of domains and

context sensitivity. Further, when considering the multilingual cross-language aim of this paper, utilizing a dictionary-based WSD system would be highly ineffective due to the lack of well-developed hand-tagged corpora like SemCor for most languages.

2.2 WSD Systems and Accuracy

The financial and temporal requirements to build systems such as SemCor or the Brown Corpus (Kilgarriff, 1998) have led many toward cost-effective automatic systems. However, when comparing automatic systems to systems based on hand-tagged data, there are still significant differences in reliability, particularly when using Semeval[4] evaluation benchmarks (Mihalcea & Edmonds, 2004). Semeval is a series of evaluation tests for computational-based semantic analysis systems. One of the tests, the 'all words' task, requires the tagging of each open-class word with its corresponding senses. As previously mentioned, systems based on manually annotated (i.e. hand-tagged) data sets performed much better on this test (Snyder & Palmer, 2004). Figure 3 (McCarthy et al., 2007, :557) shows the accuracy and recall results of the best supervised (S) and unsupervised (U) WSD systems against the benchmark of the task organizer's First Sense (FS) baseline.

	All words		Lexical sample	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
Senseval-2 S	69.0	69.0	64.2	64.2
Senseval-2 S	63.6	63.6	63.8	63.8
Senseval-2 U	45.1	45.1	40.2	40.1
Senseval-2 U	36.0	36.0	58.1	31.9
FS baseline	57.0	57.0	47.6	47.6
Senseval-3 S	65.1	65.1	72.9	72.9
Senseval-3 S	65.1	64.2	72.6	72.6
Senseval-3 U	58.3	58.2	66.1	65.7
Senseval-3 U	55.7	54.6	56.3	56.3
FS baseline	61.5	61.5	55.2	55.2

Figure 3: Accuracy and Recall Results of top 2 supervised and unsupervised WSD systems measured against FS baseline in Senseval-2 and -3 (McCarthy et al., 2007, :557)

From Figure 3, for Seneval-2 and Seneval-3 in the 'all words' category, the unsupervised WSD sys-

[4]Semeval was formerly Senseval incorporating Senseval-1, Senseval-2 and Senseval-3

tems fell far short of even the FS baseline of 57%, though there were significant improvement in accuracy and recall from Seneval-2 to Seneval-3. The FS baseline itself is a fundamental selection of the most commonly used word-sense, notwithstanding domain and context. Yet the supervised systems perform only slightly better than the unsupervised, generally not exceeding 10 percentage points of the baseline. In Seneval-3, only 5 of 26 systems managed to beat the FS baseline (McCarthy et al., 2007). The best performer, based heavily on manually tagged SemCor and running complex algorithms produced a result of 6 points above the baseline. In essence, a large investment of resources with minimal gain in performance resulting in a low resource-result ratio for current WSD systems. As of recent years, current WSD systems seem to have reached a bottleneck point around the FS baseline without much significant progress (Snyder & Palmer, 2004) Contributing to that is the lack of training data and overly high granularity of word-senses. While theoretically allowing for greater accuracy, the main practical limitation of fine-grained word-senses is the overlapping of multiple senses, especially in non-domain limited use. As a case in point, manual annotators of Senseval-2 were in agreement only on 85% of all word senses (McCarthy et al., 2007).

2.3 Examples of WSD Systems

As mentioned, this paper proposes a system that follows McCarthy et al. (2007) design and theoretical framework, if not the actual tools used. However, that is only one system utilizing one particular approach. Other systems have taken different approaches which, though not applied in the current proposed WSD system, do have interesting design approaches which can be used to further improve the proposed system once it is operational.

2.3.1 Heuristic-Rule Word Sense Disambiguation (HR-WSD)

Shih (2010) developed a highly effective Heuristic-Rule Word Sense Disambiguation (HR-WSD) system tested on domain specific data in the Chinese WordNet[5] (CWN) . They utilized two heuristic-rules to characterize the domain of the text data based on specific features of the Chinese WordNet (Li et al., 1995). Since in CWN, the word-sense is ranked by the 'prototypicality' of the sense, the predominant word-sense is assumed to be the most representative of the senses of the target word. Hence, in domain-

[5]The Chinese WordNet referred to here is built by the Academia Sinica in Taiwan and is in Traditional Mandarin Chinese

specific data, words with a predominant domain-specific sense will automatically be characterized while words that have no such domain-specific senses are characterized on the default heuristic prototype sense of the CWN. The parameters to implement both heuristic rules are as follows:

for all senses s_d w **do**

if w has domain sense

choose domain sense s_d

else

choose prototype sense s_p

In essence, the first sense is obtained through a two-pronged approach that aimed to determine the most predominant word-sense in a specific domain with the general prototype sense as a contingency back-off method to mitigate the arbitrary sense ranking seen in dictionary-based WSD systems (Shih, 2010). While a through and considered approach, the dependence on the prototype-style ranking of the CWN makes it inapplicable for our proposed system since not all WordNets use such a ranking method.

The major strength of this method, itself based on the first-sense heuristic approach of McCarthy et al. (2007), is its capability to derive predominant word-senses regardless of domain, context or genre. This universality is also what allows us to expand beyond any one language resulting in a possible multilingual platform. Given that the context of a target word is incorporated into the system through mathematical means, the user does not require prior knowledge or proficiency in the target language to interpret the results. Moreover, systems using such an approach produce results comparable to supervised or hybrid semi-automatic WSD systems benchmarked against the Senseval-2 and -3 FS baseline. In the Semeval-2010 test with supervised and knowledge-based WSD systems, tests with complete texts within a specific domain across a multilingual testbed (Agirre et al., 2010), generally favored the supervised systems.

2.3.2 CFILT

The CFILT (Center for Indian Language Technology) WSD system was one of the best performing WSD systems in Semeval-2010. The CFILT system used a semi-automatic approach that focused on a Hopfield Networks[6]-driven domain-specific knowledge-based methodology (Khapra et al., 2010).

[6]Hopfield Networks is a recurrent type of artificial neural network designed to pattern the neuron and memory processes of the human brain. The network serves as a content-addressable network system with binary threshold nodes. For more

First, context-specific words were identified using background text data before analyzing them based on a hypnoymic graph derived from WordNet, thereby providing the most representative word-sense of a particular domain. Further, approximately a hundred words from the Semeval texts were manually annotated and disambiguated as contextual disambiguation 'seeds'. Such an approach takes the best of the supervised and unsupervised systems in terms of accuracy and reliability with its worth demonstrated by the high level of performance. While an interesting development, such a system would be unsuitable for our objective of developing a multi-language system since it would require user input for all the languages our proposed system would be applied to, an extremely time-consuming task. Nonetheless, for languages that our system would be more frequently applied to, this would still be one of the methods available to improve performance.

2.3.3 Clustering-By-Comittee (CBC)

One of the most similar WSD systems to predate the first-sense heuristic system (McCarthy et al., 2007) would be the Clustering-by-Committee (CBC) system implemented by Pantel & Lin (2002) although with differing objectives. In the CBC system, Pantel and Lin sought to identify the predominant word-sense by clustering word-senses in a constructed distributional thesaurus with a 10-word window. From there, they used an algorithm by Lin (1997) to weigh the distributional scores in a manner similar to the semantic processing our proposed system uses. The result was an overall accuracy rating of 61%. Essentially, the CBC system and our proposed one share the same theoretical framework, that is identifying distribution then adding semantic weighing to provide a consistent score to determine the predominant word-sense However, their methodological process is significantly different from ours in that they removed the target word's nearest neighbors once clustered to surface and identify potential predominant word-senses that had lower frequency due to less popular usage. On the other hand, our proposed systems' first-sense heuristic method associates the possible word-senses of a target word with a pre-determined wordlist mined from WordNet (of each repective language). We then multiply the resulting distributional similarity word-sense of the nearest-neighbor word-senses with semantic similarity weighting. The underlying assumption is that predominant word-senses will exhibit a higher frequency due to greater contextual usage, as documented in WordNet, as compared to the less frequently used

information, refer to J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences of the USA*, 79(8). 2554–2558, April 1982

senses (McCarthy et al., 2007).

Another deviation from the CBC system is it does not retain the distributional scores of each cluster since they are flushed from memory once a sense has been associated with the cluster. Hence, the cluster and target word only share a distributional relationship. McCarthy et al. (2007) pointed out the possibly that low frequency senses of a target word might be synonymous with the predominant word-sense, which are encapsulated in the same cluster, forming a pseudo-predominant sense cluster with future users unable to differentiate the true predominant word-sense. Furthermore, the 10-word window of the CBC system, while accounting for the calculation of the distributional relation of more words, would require significantly more powerful computational hardware; a hard ceiling for most WSD system operators. Hence, while the CBC system allows for more extensive coverage of less frequent word-senses, it might be unable to differentiate the specific nuances of over-lapping word-senses, which we believe our proposed system is able to accomplish. Nevertheless, their system allows for easier detection of new word-senses which can then be incorporated into existing WordNets of various languages. Through such, the coverage and relevance of WordNet is updated maintaining its lifespan and usefulness.

3 Methodology

As stated earlier, our methodology aims to determine the heuristic predominant word-sense from interpreting the prevalence ranking of a given target word. The process is a two-pronged approach utilizing the Distributional Similarity Score (DSS) and the Semantic Similarity Score (SSS). The computational platform was Python with the programs written in the Interactive Development Environment (IDLE) interface with several functions of the Natural Language Toolkit (NLTK) being used. The various programs and related data processing took place over a period of 3 months with the final data set for English consisting of 12,499,099 words in 18931 files with a seperate data set for Mandarin Chinese. From the data set, the distribution and frequency of each word relative to all other words in the data set is calculated with the distribution allowing us to determine the 'Nearest Neighbors' (NN) of a target word. With the NN of a target word, it becomes possible to calculate the semantic 'closeness' of the two through the use of a distributional thesaurus, in this case WordNet. This assumption comes from Harris (1968) who postulates that words that occur in the same context would have related meanings. By comparing the various levels of semantic relatedness, it becomes possible to arrive at a predominant word-sense thus completing the WSD system.

The pre-processing of the data was done on the desktop terminal (4-core 4GB RAM) in the HSS Computational Linguistics Lab (Room 03-43) while all subsequent processes were run on the NLP server (128-core 196GB RAM), a necessary move since the data could be pre-processed in parts while any further processes had to be run on the corpus as a whole. Working on the server also allowed for easier sharing of data. The overall runtime of a complete cycle, from pre-processing the data to extracting predominant word-senses, assuming using a single-core machine, would be an estimated 25 days. Subsequent sections will detail the runtime of each individual process, as well as the means used to redure runtime.

Each section will discuss the underlying concepts and assumptions of our WSD system before going into more detailed explanations of the various processes used. Owing to the distribution of the programming work, since the scope and timeframe of this project is beyond the capabilities of a student working alone, some subsections will have more specific details of the work done. The results of the various

processes will also be presented along with a simple evaluation of the final output of our system against the FS baseline of WordNet.

3.1 Pre-processing the Data to form a Corpus

Before being able to calculate the DSS and SSS and thereby arrive at the predominant word-sense, the process must begin with data collection. A semantic repository was extracted from WordNet for the respective languages used in this project. WordNet was chosen for its easy availability and gold standards of evaluation and reliability (Cotton et al., 2001). The raw text data set was mined from the English Wikipedia text dump[7] with the Mandarin Chinese data coming from the Chinese version of the WikiDump (zhwiki). Wikipedia was chosen as the source over other corpora due to four factors, namely: relatively formal structured prose, wide coverage of various domains, multilingual availability and easy access to a huge data resource. Formal prose would mean more structured data as compared to informal text with a lower incidence of descriptive superlatives. Being a web-based unofficial user-created encyclopedia, Wikipedia would not be constrained to any single domain. Corpora often suffer from domain-specificity, such as SemCor, and using Wikipedia alleviates this to some degree by drawing from a wide range of domains. Since the Wikipedia pages of most languages are freely available and easily downloaded, running our system on a new language is a trival task as far as data collection is concerned. With these factors in mind, a higher level of reliability and coverage of raw textual data was obtainable allowing for greater representation and accuracy. The raw text data was first preprocessed through the use of the NLTK Tokenizer function[8] with each word-token being allocated a Part-of-Speech (POS) tag. This is an unsupervised automatic process that was fairly simple to write taking less than three days, including small-scale testing. Taking the earlier example:

Some party members stayed in the mainland

Tokenization would break the sentence up into its component words

some _ party _ members _ stayed _ in _ the _ mainland

which then allows for Part-of-Speech tagging[9]. The final output would be in the form

[7]URL: http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

[8]URL: http://www.nltk.org/howto/tokenize.html

[9]URL: http://nltk.googlecode.com/svn/trunk/doc/book/ch05.html#using-a-tagger

Some NNP *some*

party NN *party*

members NNS *member*

stayed VBD *stay*

in IN *in*

the DT *the*

mainland *mainland* NN[10]

As can be expected, converting approximately 13 million words to such a format is a time and memory-intensive process. After proper tokenization and POS tagging, the output used slightly under 16GB of storage with the overall process taking 5 days to run. The runtime could have been much longer; however, by dividing the raw tokenized output (that is, without part of speech annotation yet) among the four cores of the quad-core machine available, it was possible to effectively reduce the runtime required to 25% of the original. Though fairly time-consuming, processing so much raw text provides us with a large data set with a layer of morphological analysis to work with. The preprocessing of the Mandarin Chinese data after downloading was not handled by the writer. Dr. Wang Huizhen, a visiting scholar from Northeastern University, China was responsible for the process of tokenizing and POS tagging the downloaded Chinese Wikipedia[11] and the subsequent DSS calculation. The Chinese DSS data was then shared with the writer for SSS calculation. However, it should be emphasized that the algorithms and calculations used for DSS were identical for both the English and Mandarin Chinese data-sets. The division of labor was intended to allow for concurrent processing of the corpus data of the two languages in the interests of maintaining the timeframe of the project.

3.1.1 Discussion

One of the objectives of this project was to develop a WSD system that was largely independent of language-specificity. The one area this project falls short of that objective is in pre-processing the language data. Ultimately, the tools available and the nature of the language itself necessitate individual handling of each set of language data. For example, in the English data set, dividing the data on the sentence level (a.k.a. sentence splitting) would have improved accuracy but was not strictly necessary.

[10] AC_wiki_54_tokens.txt

[11] URL: zh.wikipedia.org

However, for languages like Mandarin Chinese and Japanese, it is a far more important step. Another area of difference would be the handling of multi-word expressions. For English again, aside from multi-word phrases like idiomatic expressions, operating on the word level still allows for a high degree of accuracy. Yet languages like Mandarin Chinese and Japanese which also encode meaning at the phrase level demand a different style of processing (Kando et al., 1998).

Another language-specificity issue is the processing tools available for tokenization and POS tagging. The tools contained in NLTK are mostly limited to English usage. Processing tools for non-English languages do exist but as seperate programs such as Stanford[12] for Mandarin Chinese and Mecab[13] for Japanese. Creating a single program incorporating all the various language processing tools available would be highly impractical. Rather than building a single massive combined program, it seems more prudent to include the various programs tools and call upon them only when dealing with the associated language. Once properly tokenized and tagged, any language data-set can be given to our program thanks to the mathematical functions and algorithms used in subsequent processes.

3.2 Distributional Similarity Score

Calculating the Distributional Similarity Score(DSS) is the first of the two-prong approach adopted. DSS is a means of factoring in contextual input in a mathematically calculatable fashion. While still inferior to a human-reader pulling contextual cues from the target text, it could be argued to be superior to the First Sense approach discussed earlier and drastically reduces the user-input requirement making it possible for automatic raw text WSD. Our DSS calculation process involves two steps, reading and clustering the tagged data into word-pairs then running it through a mathematical k-Nearest-Neighbors (k-NN) algorithm to arrive at a DSS for the word-pair. Using our *party* example, the process would cluster a variable number of words around the target *party* word. We designated a 3-word non-retroactive window for this run-through of the program which means the DSS process would consider the following first and second words. This is a variable rather than a fixed value hence future implementations can use a 2-, 4-, 5-, 7- etc. window length with the limitation being the time taken to process a larger window. An example of the operation of the 3-word window would be:

[12]URL: http://nlp.stanford.edu/projects/chinese-nlp.shtml

[13]URL: http://mecab.googlecode.com/svn/trunk/mecab/doc/index.html

Some party members stayed in the mainland

party members

party stayed

with the word-pairs constructed being [party_n - members_n] and [party_n - stayed_v]. If the Token_PoS format was not desired, the process allows for any one of Token, Lemma or Lemma_PoS, an intentional design to allow for working with multi-language data-sets. To lemmatize the tokens, the NLTK Lemmatizer[14] would be used before activating the window. The results in this paper were calculated using the Lemma_PoS and Token_PoS functions under the assumption that these two settings would be the most accurate as compared to non-PoS tagged data.

After building word-pairs and counting the frequency that a specific pair appeared in the data-set, the process then extracted word-pairs with a higher appearance frequency. The setting used in this instance was frequency of 10 and above with the assumption being that anything below that was likely to be a result of coincidental word arrangement rather than a meaningful pairing. Again, this setting is a variable and can be adjusted for any desired frequency from 1 upwards. The process then takes these word-pairs and associated frequencies to build a relational matrix between the first and second words of a word pair.

	member_n	stay_v
party_n	*x*	*y*

Table 1: Example of Distributional Matrix

Table 1 is a very simple representation of how the two word-pair examples given would be plotted with *x* and *y* referring to the frequency that that particular word combination occured throughout the data-set. From this matrix, the second step of k-NN algorithm application and calculation can be applied.

k-NN is an algorithm designed to calculate the distance between two words against the overall word distribution of the data-set. The implementation we used is part of the scipy package, a Python-compatible mathematics addon. We chose it for the ease of integration with our existing Python-based

[14]URL: http://nltk.googlecode.com/svn/trunk/doc/book/ch03.html#normalizing-text

```
party_n party_n 0.000000e+00 island_n 5.341234e-03 force_n 5.581005e-03
government_n 6.418186e-03 country_n 6.609601e-03 event_n 6.847817e-03 office_n
6.851687e-03 book_n 6.963410e-03 program_n 7.084079e-03 community_n
7.126122e-03 power_n 7.129760e-03 character_n 7.134257e-03 point_n 7.146939e-03
student_n 7.245137e-03 land_n 7.302530e-03 election_n 7.315752e-03 army_n
7.409826e-03 club_n 7.653517e-03 division_n 7.688447e-03 record_n 7.709049e-03
```

Figure 4: Sample Final DSS Calculation Output

code which allowed the intermediate output from the first step to be easily transferred to the second step of k-NN calculation. The k-NN algorithm produces a DSS proportional to the distance between the two words, that is the closer the two words, the smaller the DSS. The k-NN process will then rank the Nearest Neighbors to the target word. While the ranking process has an upper limit of the total size of the data set given, we decided to limit our calculations to only include the nearest 20 words. It was felt that any larger would result in diminishing returns, that is the difference between the *n*-th and *n+1*-th DSS scores would be insignificant and so on. A limit of twenty would retain most of the significant context words while still being an acceptable data-set to process for the next process in the program. The Mandarin Chinese data also underwent the same k-NN calculation process with the results appearing much the same as in Figure 4.

Figure 4 is a very small sample of the output of the k-NN process with *party_n* being the target word and the closest 20 words it. As shown, the output is a mix of logical and questionable distribution environments. For example, *party* and *government* seem compatible while *island* and *character* seem to be more random in nature. However, it should be emphasized here that DSS is a calculation of word distribution. It is a proximity-based calculation without any semantic analysis or input. Hence, the next process is specifically designed to take this distribution output and add semantic information to it.

Overall, the DSS process took approximately 10 days to write. The resulting process was seperated into two sub-processes with Step-1 being carried out by the first sub-process and Step-2 the second. This was done to allow for easier trouble shooting particularly early in Step-1 where the control of the window reading and the counting of word-pair frequencies was still being fine-tuned. Step-2 was relatively easier

to implement since it was just a question of formatting the matrix from Step-1 into a form that could be understood by the k-NN calculation process. Implementing Step-1, that is sub-process 1, required approximately 9 hours with Step-2 taking a further 20 hours to produce the final output as seen in Figure 4. This cycle was repeated for all the different word-type inputs of Token, Lemma, Token_PoS and Lemma_PoS for a total of approximately 5 days of processing time. The smaller size of the Mandarin Chinese data and the fact that in Mandarin Chinese the lemma and token form are practically the same translated into a shorter overall runtime of approximately 2 days per cycle for a final total of 4 days. The sub-processes can be found in the appendix under MatGen.py and dss_calc.py at the end of this paper.

3.3 Semantic Similarity Score

The Semantic Similarity Score (SSS) is the second part of our approach to WSD. As mentioned earlier, DSS is simply a measure of proximity to the target word. By applying the SSS process, it becomes possible to utilize the DSS output as a pseudo-context and introduce semantic analysis rather than simply taking the heuristic first sense. The concept of SSS calculation is to take the target word and the Nearest Neighbors produced from the DSS calculation output and measure the semantic similarity between the two words. The semantic word-sense are referenced from WordNet with the similarity calculated through the Python NLTK-WordNet packages'[15]Wu-Palmer Similarity Measure and Leacock-Chodorow Similarity Measure for two different sets of results. Regardless of the similarity measure used, the calculated SSS presents us with the degree of similarity between a sense of the target word with a sense of the neighboring word.

3.3.1 Wu-Palmer Similarity

The Wu-Palmer Similarity Measure is obtained by analyzing the concepts in an ontology against the taxonomic links. This is done through the tree-structure within the Wordnet of a given language, i.e. English Wordnet. First, the 'depth' between two concepts (the level in the tree-structure where the two concepts first intersect at a common node) is measured. This is balanced against the 'depth' of the Least Common Subsumer (LCS), in other words the 'depth' of the first common node to the root sense (Wu & Palmer, 1994). The formula is

[15]URL: http://www.nltk.org/howto/wordnet.html#similarity

$$Sim(S1, S2) = \frac{2 * Depth(LCS)}{Depth(S1) + Depth(S2)}, Depth(LCS) > 0$$

with the resulting similarity score having a value between 0 and 1.

3.3.2 Leacock-Chodorow Similarity

The Leacock-Chodorow Similarity Measure is essentially another approach to calculating the semantic similarity between the senses of two words via the architecture of WordNet. However, instead of considering the nearest common node and taxonomic depth of the LCS as in Wu-Palmer, the Leacock-Chodorow directly measure the distance of the path between two senses. The formula used is

$$Sim(S1, S2) = max[-\log \frac{N(path)}{2D}]$$

where *N(path)* is the number of nodes in path *p* from S1 to S2 with *D* being the maximum 'depth' of taxonomy (Leacock & Chodorow, 1998).

3.3.3 SSS Results

Since both similarity measures take different approaches to calculating semantic similarity, it was felt that using both would allow for better evaluation of the effectiveness of our WSD systems as compared to only using one measure. However, the processes that used either measure were completely identical except in the one line of code that called the similarity measure function. It should be stressed that great care was taken to ensure that calculations of subsequent processes did not mix the outputs of both measures. Data-sets calculated with Wu-Palmer Similarity would not be considered with data-sets from Leacock-Chodorow or vice versa. The results and accuracy of both measures will be further discussed in the Evaluation section.

For the Chinese data, the operational concept and calculation implementation was identical to the English data. It differed only in the retrieval of a word-sense for the target word. Due to the structure of the Chinese WordNet[16] used here, directly mapping the sense to the word is not possible unlike the English WordNet. Hence, an extra intermediate step was required to first map the target word to the WordNet

[16]URL: http://compling.hss.ntu.edu.sg/cow/

Identification Number and from there to the word-sense. While this step could be included in the process handling the English data, it would only serve to unnecessarily lengthen the runtime by adding a redundant section of code. However, implementing it for the Chinese data also allows for the inclusion of new languages that have similar structural peculiarities. All that is required for a new language to be processed would be the WordNet of that language. For a clearer sense of the difference, please refer to sss_calc.py and ssstest.py in the appendices.

Running SSS offers some advantages for WSD. As shown earlier, DSS calculations can produce some fairly curious word relations. By implementing similarity measures that are based on the links between two word-senses, any word relations that have no semantic value would be uncalculable and thus discardable. This tidies up a the data and leaves us with senses that have the possibility to be the predominant word-sense. At the same time, the mapping of word to sense through WordNet, while simple to implement with a fair degree of accuracy, does somewhat limit the final similarity output by restricting words to entries present in WordNet. This fact will be further addressed in the Discussion section.

The Python-integrated nature of both similarity measures greatly increased the ease of writing the SSS calculation process. The main body of the code was completed within a day with the variation for non-English languages a simple matter of including a few extra lines of code. Overall, the select nature of SSS calculation, regardless of measure used, further reduced the data-set from the previous DSS calculation. This resulted in a comparatively shorter runtime of less than an hour for one cycle, for a grand total of 4 hours over both data types with each similarity measure.

4 Results

Following the methodology of McCarthy et al. (2007), with the DSS and SSS data it becomes possible to determine the prevalence score of each word-sense and by ranking the senses thereby determine the predominant sense. Since the actual tools used to calculate these scores differ from McCarthy et al. (2007) in that they used Jiang-Conrath and Lesk Similarity Measures to our Wu-Palmer and Leacock-Chodorow, it would be meaningless to do a number-to-number comparison of both sets results. What can be done, since the conceptual framework of both methodologies is the same, is compare how the order of prevalent senses are different in both sets of results. Before that, a brief explanation of the calculations used to determine the prevalence scores.

4.1 Prevalence Scores

McCarthy et al. (2007) used a simple yet effective algorithm that made use of all the information from DSS and SSS to calculate the prevalence score of a particular sense. The algorithm (McCarthy et al., 2007, :565) was:

$$Prevalence\ Score(w, s_i) = \sum_{n_j \in N_w} dss(w, n_j) \times \frac{sss(s_i, n_j)}{\sum_{s_{i'} \in senses(w)} sss(s_{i'}, n_j)}$$

where,

$$sss(s_i, n_j) = \max_{s_x \in senses(n_j)} sss'(s_i, s_x)$$

The application and explanation would be more obvious from a proper example. Taking our earlier example of *party*, the DSS of the neighboring words is calculated and used to determine the SSS and both pieces of information used in the algorithm to calculate a set of prevalence scores, all as shown in Figure 5 on Page 23.

Figure 5 only displays the 5 nearest neighbors and their corresponding 4 highest senses, a reduction for the sake of presentation. The process that calculates the prevalence score naturally does not eliminate senses though it does rank them. In brief, for target word *party*, the DSS of the Neighbor *island* (5.23) is multiplied by the value of the semantic 'closeness' between *island* and **Political Group** (0.182)

Neighbours of *party* (DSS)

Senses	*island (5.23)*	*force (5.19)*	*government (5.05)*	*country (5.02)*	*event (4.98)*
Political Group	*0.182*	*0.833*	*0.769*	*0.714*	*0.400*
Group of people	*0.167*	*0.667*	*0.571*	*0.533*	*0.363*
Company	*0.182*	*0.727*	*0.615*	*0.571*	*0.400*
Occasion	*0.167*	*0.533*	*0.571*	*0.333*	*0.727*
Total	*0.698*	*2.760*	*2.526*	*2.151*	*1.89*

$$= 5.23 \times \frac{0.182}{0.698} + 5.19 \times \frac{0.833}{2.760} + 5.05 \times \frac{0.769}{2.526} + 5.02 \times \frac{0.714}{2.151} + 4.98 \times \frac{0.400}{1.890}$$

$$= 1.364 + 1.566 + 1.537 + 1.666 + 1.05$$

$$= 7.183$$

prevalence score (**political group**) = **7.183**
prevalence score (**group of people**) = 5.846
prevalence score (**company**) = 6.342
prevalence score (**occasion**) = 6.080

Figure 5: Calculation and Ranking of Senses for *Party_n*

divided by the sum of all (**Political Group, Group of People, Company etc.**) sense similarity measure scores with *island* (0.698). This process is repeated for all links between the Neighbors (*island, force, government etc.*) and **Political Group**. This series of calculations are the actual implementation of the algorithm for prevalence scores as used to calculate the prevalence score for **political party**. The prevalence scores for the other senses are also calculated in the same way. Comparing the scores, **political party** is the highest at 7.183 making it the predominant sense for *party* as calculated by our WSD system.

4.2 Examples

Since the calculations are purely mathematical, the algorithm works regardless of language. The Mandarin Chinese DSS and SSS data-sets were given to the same process with similar results. An

example from the Mandarin Chinese data can be found on Page 24 for 年_n, meaning *year* with the Predominant Sense being **calender year**.

Neighbours of 年 year (DSS)

Senses	深夜 *night (2.12)*	股 *stock (2.04)*	頻道 *channel (2.05)*	碼 *number (2.08)*	台幣 *Taiwan dollars (2.02)*
365 days	*0.235*	*0.125*	*0.117*	*0.429*	*0.400*
Time take for planet revolution	*0.250*	*0.133*	*0.125*	*0.462*	*0.428*
Calendar year	*0.250*	*0.133*	*0.712*	*0.462*	*0.428*
Total	*0.735*	*0.391*	*0.954*	*1.353*	*1.256*

Figure 6: Calculation of Senses for 年

Another example from the English data-set would be the target word *car* as shown in Figure 7

Neighbours of *car* (DSS)

Senses	*engine (5.86)*	*hill (5.81)*	*bridge (5.76)*	*aircraft (5.75)*	*facility (5.72)*
Automobile	*0.869*	*0.571*	*0.600*	*0.727*	*0.588*
Railcar	*0.762*	*0.632*	*0.667*	*0.800*	*0.667*
Gondola	*0.526*	*0.706*	*0.706*	*0.500*	*0.625*
Elevator car	*0.526*	*0.706*	*0.706*	*0.500*	*0.625*
Total	*2.683*	*2.615*	*2.679*	*2.527*	*2.505*

prevalence score (**automobile**) = **7..845**
prevalence score (**railcar**) = 7.452
prevalence score (**gondola**) = 6.801
prevalence score (**elevator car**) = 6.801

Figure 7: Calculation and Ranking of Senses for *Car_n*

with the Predominant Sense of **automobile**.

Neighbours of *star* (DSS)					
Senses	*movement (6.08)*	*ground (5.99)*	*base (5.97)*	*night (5.94)*	*scene (5.92)*
Celestial star	*0.471*	*0.571*	*0.533*	*0.167*	*0.500*
Champion	*0.421*	*0.000*	*0.571*	*0.182*	*0.444*
Star (Visible from Earth)	*0.471*	*0.571*	*0.533*	*0.167*	*0.500*
Lead	*0.381*	*0.444*	*0.421*	*0.154*	*0.400*
Total	*1.744*	*1.586*	*2.058*	*0.67*	*1.844*

Prevalent senses	**McCarthy et al (2007)**	**Our paper (Prevalence score)**
1	Celebrity	Celestial star (8.432)
2	Celestial body	Champion (6.163)
3	Shape	Star (Visible from Earth) (8.432)
4	Zodiac	Lead (6.877)

Figure 8: Calculation and Ranking Comparison of Senses for *Star_n*

Figure 8 is a comparison of the results from McCarthy et al. (2007) and our own system. From the example, there are some significant differences. Our Predominant Sense of **Celestial Star** is McCarthy et al. (2007) second highest sense while their third and fourth sense are not even considered in our data-set. We argue that this is largely a function of domain-specificity. As a formal prose pseudo-encyclopedia, the number of entries that use *star* in the context of **celestial body** in Wikipedia would be much higher than McCarthy et al. (2007) data-set of SemCor. and vice versa for **celebrity**. If the data-set was switched from Wikipedia to another corpus, it is highly possible that our system would produce results identical to McCarthy et al. (2007).

5 Evaluation and Discussion

From the previous section, it is fairly apparent that our WSD system is fully capable of running through a complete cycle, from data input to determining the Predominant Sense. Since ability is assured, the question now is one of effectiveness, that is, whether the results of our system are accurate and can be used. To determine this, we used a simple comparison of our Predominant Word-Sense of a target word against the highest frequency sense of that word in WordNet as the baseline. We did this for the predominant word-senses produced from both Wu-Palmer and Leacock-Chodorow Similarity values. The results are shown in Figure 9.

Settings	No. of Words	Accuracy (%)	Independent Senses (Word-Class Conflict)	Multiple Senses
Lemma-PoS *wup*	37171	74.085	9072 (32)	2164
Lemma-PoS *lch*	36237	74.994	8564 (0)	1989
Token-PoS *wup*	34096	72.640	8827 (34)	2044
Token-PoS *lch*	33263	73.709	8262 (0)	1838

Figure 9: Evaluation Against WordNet

Since the data-set was limited to target and neighbor words that appear in WordNet, the overall length of the data-set was much reduced to less than 40,000 words. The smaller sample size is what makes the accuracy rate seem higher than it actually is, if one were to compare to Figure 3 on Page 8. Still, an accuracy of above 70% for both data types of Lemma-PoS and Token-PoS and both similarity measures is no mean feat. Also, bear in mind that the accuracy rate is in comparison to the most frequent sense of the target word in WordNet. This by no means guarantees that the most frequent sense is the correct one, especially if the difference in frequencies of WordNet Sense-1 and Sense-2 are negligible.

Figure 10 shows some examples where our predominant sense disagrees with WordNet. While some are clearly different, as in *abundance_n* where WordNet offers a general sense of the word as compared to our physics-specific sense, others are less distinct. The other two predominant word-sense examples here of *relax_v* and *boiling_n* are not so much different senses as different ‘degrees’ of the same sense,

Word (PoS)	Predominant sense in Wordnet (Sense location serial)	Our predicted predominant sense (Sense location serial)
Relax (V)	*become less tense, loosen up (00026378)*	*make less severe or strict (02607993)*
Abundance (N)	*copiousness, teemingness (05122340)*	*(physics) ratio of the number of atoms of specific isotope present against the total number of isotopes (13842451)*
Boiling (N)	*the application of heat (13461952)*	*cooking in a liquid that has been brought to a boil (00248659)*

Figure 10: Sample of Conflicting Word-Senses

particularly in the case of *boiling_n*. Earlier in this paper, the 'granularity' of senses and the effect it has on WSD was raised and this seems to be a prime example.

Unfortunately, evaluation of the Chinese data and predominant word-senses was not possible. As mentioned in the SSS section, the nature of the structure of the Chinese WordNet does not facilitate the direct access of word to sense. Therefore, despite being able to reliably output the predominant word-senses, comparing them to the highest frequency senses of WordNet is not possible. I would suggest that due to the multi-word nature of Mandarin Chinese, the greater specificness would allow for a smaller pool of possible predominant senses hence making it easier to determine the predominant sense. However, this is pure speculation on my part which would require some effort to verify or disprove, specifically in designing a way to access the Chinese WordNet.

Upon detailed examination of the data, it was found that there were some target words that had multiple, equally valid senses which is reflected in the Multiple Senses column of Figure 9. Our system handles such situations by adopting a first-priority approach. The first sense for a target word is taken to be the predominant sense with any subsequent senses of equal prevalence score being discarded. Multiple senses are usually an indicator of senses that are on the same level in the tree-structure of WordNet, hence the similarity measure calculates identical SSS results for all of them. A good example is *ping_v* where senses 1 and 3 both had equal prevalence scores. Accessing the senses in WordNet[17] show them to be **1:**

[17]URL: http://wordnetweb.princeton.edu/perl/webwn?s=ping&sub=Search+WordNet&o2=&o0=1&o8=1&o1=1&o7=

hit with a pinging noise and **3: make a short high-pitched sound** both of which seem similar enough to occupy adjacent nodes in WordNet. Solving such conflicts would seem to be beyond an automatic unsupervised system since it is merely measuring the distance in WordNet tree-structure rather than truly comprehending the senses as a human reader might. To solve this, I would suggest allowing for some measure of user input, where in the event of multiple word-senses the user is provided with the details and is prompted to make a decision. That way, when subjected to the same evaluation test used earlier, the user-selected predominant sense has a higher likelihood of being correct since a user would choose the most familar sense as the predominant sense. Alternatively, the system could be instructed that in the event of multiple senses, the sense that has the highest frequency in WordNet should be selected and presented as the predominant sense, again an argument of greater usage and thus higher frequency.

&o5=&o9=&o6=&o3=&o4=&h=

6 Conclusion

The objective of this project was to design a WSD system that could accept raw text data from any language and process the data to determine the Predominant Word Sense. I would argue that that goal has been achieved with a fair degree of accuracy. Although pre-processing the data is still on an individual language basis, subsequent processes to calculate the prevalence scores and arrive at a predominant sense have been able to be applied evenly to both English and Mandarin Chinese with minimal changes when switching languages.

Not to say that there exists no room for improvement. From the design perspective, there are still many changes that can be made to either increase the ease of use or improve accuracy. As mentioned, pre-processing the data still requires individual langauge-specific handling. A 'master program' of sorts could be designed and implemented to take any language data, accept user input to which specific language the data is from and activate the appropriate software processes to tokenize and tag the data. An ambitious program would go a step further and integrate some level of automatic language recognition such that a user is just required to input the data-set and the program would handle the rest.

As mentioned in the Evaluation and Discussion Section, words with multiple predominant senses could be handled differently which could raise the accuracy level of the program. The programs' dependence on WordNet, while greatly simplifying similarity measure calculations, does affect preformance and evaluation, particularly for non-English data. The introduction of other thesaurus or dictionaries could allow for more options when calculating similarity. However, to keep to the original intent of a multi-language program, such databases for each language would have to be individually integrated, an extremely intensive task for little overall gain. The advantage, and selection, of WordNet was precisely for its multi-language capabilities and finding a similar database in not a trivial undertaking.

In conclusion, I would say that this project was successful in developing a WSD system that could work with language data from any language and produce Predominant Word Sense output of a fair degree of accuracy. Further development of the system would help to improve performance and/or accuracy and the many variables in the process allow for even more exhaustive testing and evaluation. Different

window lengths can be tested to find an optimal length for the best distributional scores while still maintaining system processing speeds. The number of Nearest Neighbors considered can also be adjusted to increase the distribution of senses which would improve semantic similarity distinction. Simply experimenting with different values for these variables could very well be another project aimed at determining the optimal settings for the highest degree of accuracy. Additional languages can be introduced to the system to determine how widely it can be used. The availability of the Japanese and *Bahasa* WordNets allowing with pre-processing tools make these languages the next logical extension of the program. What this paper presents is the core of a system that has the potential for widespread application and further refinement.

References

Agirre, E., de Lacalle, O. L., Fellbaum, C., Hsieh, S., Tesconi, M., Monachini, M., ... Segers, R. (2010). SemEval-2010 task 17: All-words Word Sense Disambiguation on a Specific Domain. In *Proceedings of the 5th International Workship on Semantic Evaluations (SemEval-2010).* Uppsala, Sweden.

Barclay, J. R., Bransford, J. D., Franks, J. J., McCarrell, N. S., & Nitsch, K. (1974). Comprehesion and Semantic Flexibility. *Journal of Verbal Learning and Verbal Behavior*, *13*, 471-481.

Boyd-Graber, J., Blei, D., & Zhu, X. (2007). A Topic Model for Word Sense Disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (p. 1024-1033). Prague.

Cotton, S., Edmonds, P., Kilgarriff, A., & Palmer, M. (2001). *Senseval-2.* Retrieved from `http://www.sle.sharp.co.uk/senseval2`

Croft, W., & Cruse, D. (2004). *Cognitive Linguistics*. Cambridge University Press.

Fellbaum, C. (Ed.). (1998). *Wordnet: An Electronic Lexical Database*. MIT Press, Cambridge MA.

Firth, J. R. (1957). *Studies in Linguistics Analysis*. Basil Blackwell Oxford, England.

Harris, Z. S. (1968). *Mathematical Structures of Languages*. Wiley, New York, NY.

Kando, N., Kageura, K., Yoshioka, M., & Oyama, K. (1998). Phrase Processing Methods for Japanese Text Retrieval. *ACM SIGIR Forum*, *32*, 23-28.

Khapra, M. M., Sohoney, S., Kulkarni, A., & Bhattacharyya, P. (2010). All Words Domain Adapted WSD: Finding a Middle Ground between Supervision and Unsupervision. In *Proceedings of the 5th International Workship on Semantic Evaluations (SemEval-2010).* Uppsala, Sweden.

Kilgarriff, A. (1998). Gold Standard Datasets for Evaluating Word Sense Disambiguation Programs. *Computer Speech and Language*, *12*, 453-472.

Kintsch, W., & Mangalath, P. (2011). The Construction of Meaning. *Topics in Cognitive Science*, *3*, 346-370.

Landes, S., Leacook, C., & Tengi, R. I. (1998). *Building Semantic Concordances*. Cambridge MA: MIT Press.

Leacock, C., & Chodorow, M. (1998). Combining Local Context with WordNet Similarity for Word Sense Identification. In *Christiane Fellbaum (editor) WordNet: A Lexical Reference System and its Application.* MIT Press, Cambridge MA.

Li, X., Szpakowicz, S., & Matwin, S. (1995). A Wordnet-based Algorithm for Word Sense disambiguation. In *The 14th Internation Joint Conference on Artificial Intelligence.*

Lin, D. (1997). Using Syntactic Dependency as Local Context to resolve Word Sense Ambiguity. In *Proceedings of ACL-97* (p. 64-71). Madrid, Spain.

Magnini, B., Strapparava, C., Pezzuli, G., & Gliozzo, A. (2002). The Role of Domain Information in Word Sense Disambiguation. *Natural Language Engineering*, *8*, 359-373.

McCarthy, D., Koeling, R., Weeds, J., & Carroll, J. (2004). Finding Predominant Word Senses in Untagged Text. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics* (p. 280-287). Barcelona, Spain.

McCarthy, D., Koeling, R., Weeds, J., & Carroll, J. (2007). Unsupervised Acquisition of Predominant Word Senses. *Association for Computational Linguistics*, *33*(4).

Mihalcea, R., & Edmonds, P. (2004).
In R. Mihalcea & P. Edmonds (Eds.), *Proceedings Senseval-3 3rd International Workshop on Evaluating Word Sense disambiguation Systems.* Barcelona, Spain.

Miller, G. A., Leacock, C., Tengi, R., & Bunker, R. T. (1993). A Semantic Concordance. In *Proceedings of the ARPA Workshop on Human Language technology* (p. 303-308). San Francisco, CA.

Pantel, P., & Lin, D. (2002). Discovering Word Senses from Text. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (p. 613-619). Edmonton, Alberta, Canada.

Procter, P. (Ed.). (1978). *Longman Dictionary of Contemporary English*. Harlow, UK: Longman Group Ltd.

Shih, M.-H. (2010). System Description for All-Words Word Sense Disambiguation on a Specific Domain at SemEval2010. In *SemEval'10 Proceedings of the 5th internation Workship on Semantic Evaluation* (p. 433-435).

Snyder, B., & Palmer, M. (2004). The English All-Words Task. In *Proceedings of the ACL Senseval-3 Workshop* (p. 41-43). Barcelona, Spain.

Stevenson, M., & Wilks, Y. (2001). The Interaction of Knowledge Sources for Word Sense Disambiguation. *Computational Linguistics*, *27*, 321-350.

Wu, Z., & Palmer, M. (1994). Verb Semantics and Lexical Selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics* (p. 133-139). New Mexico State University, New Mexico.

Yarowsky, D., & Florian, R. (2002). Evaluating Sense Disambiguation Performance Across Diverse Parameter Spaces. *Natural Language Engineering*, *8*, 293-310.

Appendix 1 MatGen.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
import os
import codecs
import nltk
import re
import sys
import gc
from nltk import *
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
import itertools
from sets import Set
from collections import defaultdict as dd

###Specifying Input/Output###

engwnet = codecs.open('wn-data-eng.tab', 'r', encoding='utf-8')

wnet = set()
keynum=dict()
xkey=list()
ykey=list()
xyknown=set()
xykey=list()
fcount = 0
xyv = set()
threshold = 9 #Significant frequency level
lemmatizer = WordNetLemmatizer()
gc.isenabled()

def pos2wn (pos, lang, lemma=''):
    ### FIXME: check and document --- Change POS for VN?
    ### する　できる　なさる　いたす　致す
    if lang == 'jpn':
        if pos in [u'名詞-形容動詞語幹', u"形容詞-自立", u"連体詞"] \
                and not lemma in [u"この", u"その", u"あの"]:
            return 'a'
        elif pos == u"名詞-サ変接続":
             return 'n'
        elif pos in [u"名詞-サ変接続",  u"名詞-ナイ形容詞語幹",
                     u"名詞-一般", u"名詞-副詞可能",
                     u"名詞-接尾-一般", u"名詞-形容動詞語幹",
                     u"名詞-数",  u"記号-アルファベット"]:
            return 'n'
        elif pos in [u"動詞-自立", u"動詞-サ変接続"]:
            return 'v'
        elif pos in [u"副詞-一般", u"副詞-助詞類接続"]:
            return 'r'
        else:
            return 'x'
    elif lang=='eng':
        if  pos == 'VAX':  #local tag for auxiliaries
            return 'x'
        elif pos.startswith('N'):
```

```
            return 'n'
        elif pos.startswith('V'):
            return('v')
        elif pos.startswith('J'):
            return('a')
        elif pos.startswith('R'):
            return('r')
        else:
            return 'x'
    elif lang=='cmn':
        if pos in "NN NN2 CD DT PN PN2 LC M M2 NR NT".split() and \
                not lemma in u"这 这个 这些 那 那个 那些 什么 时候".split():
            return 'n'
        elif pos in "VV VV2 VC VE".split() and not lemma in [u"可以", u"能
"]:
            return 'v'
        elif pos in "JJ JJ2 OD VA VA2".split():
            return 'a'
        elif pos in "AD AD2 ETC ON".split():
            return 'r'
        else:
            return 'x'
    else:
        return 'u'

def featbuild(word, idno):
      typ = ['tokens', 'tokenspos', 'lemmas', 'lemmaspos']
      typn = typ[idno]
      fname = '%s/%sindex' % (typn, lang)
      index = codecs.open(fname, 'w', encoding='utf-8')
      fname = '%s/%sindex_num' % (typn, lang)
      index_num = codecs.open(fname, 'w', encoding='utf-8')
      fname = '%s/%smatrix.mtx' % (typn, lang)
      matrix = open(fname, 'w')
      fname = '%s/%swmatrix.mtx' % (typn, lang)
      wmatrix = codecs.open(fname, 'w', encoding='utf-8')
      sys.stderr.write('Working with Type: -%s- for Matrix Construction
and DSS Calculation\n' % typn)
#     y = len(word)
      ### Building Window and Freq Count  ###

      ## Initializing Wordnet for w1

      sys.stderr.write('Files Read - %d words total used.\nStarting
Matrix Construction\n' % len(word))
      sys.stderr.write('Count how many x, y and total, non-zeros\n')
      ### count how many x, y and total, non-zeros
      ### Building Word Index
      xkeys=set()
      ykeys=set()
        freqall = 0
      for w1 in word:
          if w1 in wnet:
            for w2 in word[w1]:
                if word[w1][w2] > threshold:
                  freqall +=1
                  if w1 not in xkeys:
                        xkeys.add(w1)
                      xkey.append(w1)
```

```
                if w2 not in ykeys:
                      ykeys.add(w2)
                xyv.add((w1, w2))

      xykey = xkey + list(ykeys.difference(xkeys))
      #xkeysf = open('xkeys_list', 'w')
      #xkeysf.write('\n'.join(xkeys))
      #ykeysf = open('ykeys_list', 'w')
      #ykeysf.write('\n'.join(ykeys))
      #xykeysf = open('xykeys_list', 'w')
      #xykeysf.write('\n'.join(xykey))

      sys.stderr.write('Counted how many x, y, nonzeros: %d, %d, %d
(above threshold %d)\n' % (len(xkeys), len(xykey), len(xyv), threshold))

      sys.stderr.write('Beginning Keylist Construction\n')

      for i,k in enumerate(xykey):
          ### add 1 to get a 1 indexed list
          index.write("%s\n" % k)
          index_num.write("%s\t%d\n" % (k, i+1))
          keynum[k]= i+1
      #    print k, keylist[k]

      sys.stderr.write('Index Built - %d keylist words\n' % len(keynum))

      # Write the header
      matrix.write("%%MatrixMarket matrix coordinate real general\n")
      matrix.write("%d %d %d\n" % (len(xkeys), len(xykey), len(xyv)))

      sys.stderr.write("Writing Matrix with (x, y, nonzero) (%d, %d,
%d)\n" % \
                           ((len(xkeys), len(xykey), len(xyv))))

      ### count how many x, y and total, non-zeros
      ### write the non-zero values

      for w1 in word:
          if w1 in wnet:
            for w2 in word[w1]:
                if word[w1][w2] > threshold:
                  wmatraw = "'%s, '%s', '%d'" % ((w1), (w2), word[w1][w2])
                  wmatrix.write(wmatraw + "\n")
                  freq = word[w1][w2]
                     matraw = "%d %d %e" % (keynum[w1], keynum[w2],
freq/freqall)
          #            print matraw
                  matrix.write(matraw + "\n")


      sys.stderr.write('Matrix Constructed\n')
      matrix.close()

sys.stderr.write('Starting\n')
feats_l = []
lemin = raw_input('Use Lemmas?: ')
posin = raw_input('Use Part of Speech?: ')
lang = 'eng'
feats = dd(lambda:dd(int))
width = 2 #width we look on right side (window)
```

```
sys.stderr.write('Chosen - Lemma: %s\tPart of Speech: %s\n' % (lemin,
posin))
for root, dirs, files in os.walk('EngTokendir'):
    #for f in ['AA_wiki_00_tokens.txt']:
    for f in files:
      if f.endswith("txt"):
          fcount +=1
          sys.stderr.write('Reading %d of Tokendir Files\n' % fcount)
            filename = ('%s/%s') % (root,f)
            tokenlist = codecs.open(filename, 'r', encoding='utf-8')
          for l in tokenlist:
            (w, p) = l.strip('<>/:').split('\t')
                ### strip punctuation, should maybe tokenize better
                w = w.lower().strip("""-.:)(=";?!,""")
            p = pos2wn(p, 'eng')
            ### Skip to next if w is the empty string
                if not w:
                    continue
            if lemin == 'True' and posin == 'False':
                if p == 'x':
                 lm = w
                else:
                   lm = lemmatizer.lemmatize(w, p)
                feats_l.append(lm)
            elif lemin == 'True' and posin == 'True':
                if p == 'x':
                 lm = w
                else:
                   lm = lemmatizer.lemmatize(w, p)
                feats_l.append("%s_%s" % (lm, p))
            elif lemin == 'False' and posin == 'True':
                feats_l.append("%s_%s" % (w, p))
            elif lemin == 'False' and posin == 'False':
                feats_l.append(w)
          tokenlist.close()
          y = len(feats_l)
          for i in range(0, y-width, 1):
              for j in range(1, width+1):
                  feats[feats_l[i]][feats_l[i+j]] +=1
          del feats_l[:]

sys.stderr.write('Processing Corpus of %d word pairs\n' % y)
for e in engwnet:
    (n, w) = e.strip().split('\t')
    p = n[-1]
    if posin == 'True':
      wnet.add("%s_%s" % (w, p))
    else:
      wnet.add(w)


sys.stderr.write('Wordnet Read - %d unique words or word_pos\n' %
len(wnet))


if lemin == 'True' and posin == 'False':
    sys.stderr.write('Lemma Used\n')
    featbuild(feats, 2)
elif lemin == 'True' and posin == 'True':
    sys.stderr.write('Lemma_PoS Used\n')
```

```
    featbuild(feats, 3)
elif lemin == 'False' and posin == 'True':
    sys.stderr.write('Word_PoS Used\n')
    featbuild(feats, 1)
elif lemin == 'False' and posin == 'False':
    sys.stderr.write('Word Used\n')
    featbuild(feats, 0)
```

Appendix 2 dss_calc.py

```
from __future__ import division
import os
import codecs
import nltk
import re
import sys
from sklearn.neighbors import NearestNeighbors
from scipy.io import mmread
import numpy as np
import scipy as scipy
import heapq
import scipy.sparse as sparse
lang = 'eng'
sys.stderr.write('Starting Matrix Processing\n')
#arange the sparse matrix text into an array
fname = '%smatrix.mtx' % lang
spm = open(fname,"r")
fname = '%sindex' % lang
indexw = open(fname,"r")
X = scipy.io.mmread(spm)
spm.close()
sys.stderr.write("Sparse Matrix Loaded\n")

#X = X.todense()
#Using the while function to loop the knn function for each line

sys.stderr.write('Calculating K Nearest Neighbor\n')


neigh = NearestNeighbors(n_neighbors=100)
Z = X.tocsr()
neigh.fit(Z)

NearestNeighbors(algorithm='auto', leaf_size=30)

line = indexw.readline()
num = 0
arr=dict()
while line:
      line = line.rstrip('\n')
      arr[num] = line
#        print line
        num = num + 1
      line = indexw.readline()
indexw.close()

#open a new file and output to text file for y is a sentence in the array
output = open("dss_output", "w")
num = 0
for Y in Z:
#    print num, Y

    nk = neigh.kneighbors(Y)
 #   print nk
    output.write(arr[num] + "     ")
    for j in range(20):
       out = "%s %e" % (arr[nk[1][0][j]],nk[0][0][j])
       output.write(out + " ")
```

```
    output.write("\n")
    num = num + 1
output.close()

sys.stderr.write('Distributional Similarity Score Calculated\n')
```

Appendic 3 sss_calc.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
import os
from nltk.corpus import wordnet
from nltk.corpus import wordnet as wn
import codecs
import math
from math import *
import decimal
from decimal import *
import nltk
import re
import sys
score_l = []
dssraw = codecs.open('dss_output', 'r', encoding='utf-8')
sss = open('sim_output', 'w')
st = codecs.open('sim_total', 'w', encoding='utf-8')
dssout = dssraw.read()
chunk = dssout.strip('').split('\n')

def max_sim(synset1, synset2, score):        #source:http://www.coli.uni-
saarland.de/courses/python-11/contents/Slides/18_nltk_wordnet.pdf
   total = 0
   for s1 in synset1:
      maxSim = 0
      for s2 in synset2:
          sim = s1.wup_similarity(s2)
          if maxSim < sim:
            maxSim = sim
      if score != 1.0000 and maxSim != 0.0000:
         sss.write('%s %s %s %.4f %.4f\n' % (w1, s1, w2, score, maxSim))
           total+=maxSim
   if total != 0.0000:
        st.write('%s\t%s\t%.4f\n' % (w1, w2, total))

sys.stderr.write('Starting\n')
for i in range(0, len(chunk), 1):
    word = chunk[i].split('\t')
    if word[0] != '':
      w1 = word[0]
      tar = word[1].split(' ')
      for j in range(0, len(tar), 2):
          if j+1 < len(tar):
              w2 = tar[j]
              ds = tar[j+1]
            if ds == '0.000000e+00':
                dss = 1.0000
            else:
                ds = Decimal(ds)
                ds = 1/ds
                dss = math.log(ds)
            if '_' in w1:
                (t1, p1) = w1.strip().split('_')
                syn1 = wn.synsets(t1, pos = p1)
            else:
                syn1 = wn.synset(w1)
            if '_' in w2:
```

```
                (t2, p2) = w2.strip().split('_')
                syn2 = wn.synsets(t2, pos = p2)
            else:
                syn2 = wn.synsets(w2)
            max_sim(syn1, syn2, dss)

sys.stderr.write('Similarity Scores Calculated\n')
#               print w1,w2,ss,dss
#           score_l.append('%s, %s, %s, %f' % (w1, w2, dss, ss))
```

Appendix 4 ssstest.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
import os
from nltk.corpus import wordnet
from nltk.corpus import wordnet as wn
from nltk import defaultdict as dd
import codecs
import math
from math import *
import decimal
from decimal import *
import nltk
import re
import sys

def id2ss(ID):
    """Given a Synset ID (e.g. 01234567-n) return a synset"""
    try:
      return wn._synset_from_pos_and_offset(str(ID[-1:]), int(ID[:8]))
    except:
        sys.stderr.write('%s not in Wordnet\n' % ID)
        return None

def max_sim(synset1, synset2, score):        #source:http://www.coli.uni-
saarland.de/courses/python-11/contents/Slides/18_nltk_wordnet.pdf
    total = 0
    for s1 in synset1:
        maxSim = 0
      for s2 in synset2:
          sim = s1.wup_similarity(s2)
          if maxSim < sim:
            maxSim = sim
      if score != 1.0000 and maxSim != 0.0000:
         sss.write('%s %s %s %.4f %.4f\n' % (w1, s1, w2, score, maxSim))
           total+=maxSim
    if total != 0.0000:
        st.write('%s\t%s\t%.4f\n' % (w1, w2, total))


netset = dd(set)
lang = raw_input('Language Is?(English = eng; Chinese = cmn): ')
dssraw = codecs.open('zh_pos_dss_output', 'r', encoding='utf-8')
sss = codecs.open('sim_output', 'w', encoding='utf-8')
st = codecs.open('sim_total', 'w', encoding='utf-8')
dssout = dssraw.read()
chunk = dssout.strip('').split('\n')
if lang != 'eng':
    wnet = codecs.open('wn-data-'+lang+'.tab', 'r', encoding='utf-8')
    for l in wnet:
      if l.startswith('#'):
          continue
      (n, l, w) = l.strip().split('\t')
      p = n[-1]
      t = '%s_%s' % (w, p)
      if id2ss(n):
          netset[t].add(id2ss(n))
```

```
sys.stderr.write('%s WordNet Read - %d Words\nStarting...\n' % (lang,
len(netset)))
for i in range(0, len(chunk), 1):
    word = chunk[i].split('\t')
    if word[0] != '':
      w1 = word[0]
      tar = word[1].split(' ')
      for j in range(0, len(tar), 2):
          if j+1 < len(tar):
              w2 = tar[j]
              ds = tar[j+1]
            if ds == '0.000000e+00' or ds == '0.00':
                dss = 1.0000
            else:
                ds = Decimal(ds)
                ds = 1/ds
                dss = math.log(ds)
            #if '_' in w1:
                #ind = [k[0] for k in netset].index(w1)
                #syn1 = id2ss(netset[ind][1])
            #if '_' in w2:
                #ind = [k[0] for k in netset].index(w2)
                #syn2 = id2ss(netset[ind][1])
            syn1 = netset[w1]
            syn2 = netset[w2]
            max_sim(syn1, syn2, dss)

sys.stderr.write('Similarity Scores Calculated\n')
#               print w1,w2,ss,dss
#           score_l.append('%s, %s, %s, %f' % (w1, w2, dss, ss))
```

Appendix 5 pscore.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
import os
from nltk.corpus import wordnet
from nltk.corpus import wordnet as wn
import codecs
import math
from math import *
import decimal
from decimal import *
import nltk
from nltk import defaultdict as dd
import re
import sys

sys.stderr.write('Starting...\n')
total = codecs.open('sim_total', 'r', encoding='utf-8')
sim_score = codecs.open('sim_output', 'r', encoding='utf-8')
prv_score = codecs.open('preval_output', 'w', encoding='utf-8')
tsall = {}
prev =dd(lambda: dd(int))
count = 0
sys.stderr.write('Reading Similarity Totals...\n')
for l in total:
    (w1, w2, t) = l.strip().split('\t')
    wp = '%s %s' % (w1, w2)
    tsall[wp] = t

sys.stderr.write('Totals Read\nReading Similarity Scores and
Calculating...\n')
for n in sim_score:
    (w1, s1, w2, ds, ss) = n.strip().split(' ')
    w = '%s %s' % (w1, w2)
    if w in tsall:
      ts = tsall[w]
      ss = str(ss)
      s = float(ss)/float(ts)
      ps = float(str(ds))*s
      prev[w1][s1] += ps
    else:
      sys.stderr.write('%s does not have a total!!\n' % w)
      continue

sys.stderr.write('Calculating Prevalence Score and Sense...\n')

for w in prev:
    maxscore = max(prev[w][s] for s in prev[w])
    for s in prev[w]:
        if maxscore == prev[w][s]:
          prv_score.write('%s\t%s\t%s\n' % (w, s, maxscore))
          count+=1
sys.stderr.write('Predominant Word-Senses for %d Words Calculated\n' %
count)
```

Appendix 6 eval_test.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division
import os
from nltk.corpus import wordnet
from nltk.corpus import wordnet as wn
import codecs
import nltk
from nltk import defaultdict as dd
import re
import sys

sys.stderr.write('Starting...\n')
raw = codecs.open('preval_output', 'r', encoding='utf-8')
evaltest = codecs.open('eval_output', 'w', encoding='utf-8')
evalpostest = codecs.open('evalpos_output', 'w', encoding='utf-8')
count = 0  #Count differing prevalent senses
nvcount = 0#Cout differing senses due to different word-classes
check = 0  #Count matching senses
mulc = 0   #Count senses with more than 1 prevalent sense
syncheck = set()
sys.stderr.write('Comparing Prevalent Senses...\n')
for r in raw:
    (t, syn, s) = r.strip().split('\t')
    (w, p) = t.strip().split('_')
    ts = '%s,%s' % (t, syn)
    testsyn = sorted([(l.count(), l) for l in wn.lemmas(w,
pos=p)],reverse=True)[0][1].synset
    if not t in syncheck:
      syncheck.add(t)
      if str(testsyn) != str(syn):
          count+=1
          evaltest.write('%s %s\t%s\n' % (t, syn, testsyn))
          if str(testsyn)[-6] != str(syn)[-6]:
              #print t, syn, testsyn
              evalpostest.write('%s %s\t%s\n' % (t, syn, testsyn))
              nvcount+=1
          else:
            continue
      else:
          check+=1
          continue
    elif t in syncheck:
      mulc+=1
      sys.stderr.write('Multiple Entry %s!!\n' % ts)
      continue

acc = (check/len(syncheck))*100
sys.stderr.write('%d Differing Word-Senses\n%d True Differing Word-
Senses\n%d Multiple Senses of the Same Word\n' % (count, nvcount, mulc))
sys.stderr.write('Accuracy : %.3f Percent\n' % acc)
```