Undergraduate Research Opportunity Program
(UROP) Project Report

Word Sense Prediction Using Decision Trees

By

Heng Low Wee
(U096901R)

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Undergraduate Research Opportunity Program
(UROP) Project Report

Word Sense Prediction Using Decision Trees

By

Heng Low Wee
(U096901R)

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Project No: U079330
Advisor: A/P Min-Yen Kan
Deliverables:
Report: 1 Volume
Source Code: 1 DVD

Abstract

Word sense disambiguation (WSD) is a key task in Natural Language Processing. There are many existing systems that are able to achieve good performance in SensEval and SemEval tasks. However, few systems discuss about reducing the size of the training model and the time taken for the WSD process. For WSD systems to be applied onto real-time softwares, such as building a WSD software for mobile devices, they need to be scaled down in terms of its training model size, and be responsive to enhance user experience. With that, we propose the idea of building a WSD system that is light-weight and speedy.

In this report, we study how we can adopt using Decision Trees to predict the word senses of ambiguous words. Our system aims to reduce the size of the training model and the time taken to predict word senses. In our implementation, we reduced both the size of our training model and the time taken by over 98%, with our system performing speedily when disambiguating sentences, while retaining a reasonable performance for accuracy.

Subject Descriptors:
I.2.7 Natural Language Processing

Keywords:
decision tree, word sense disambiguation, word sense, ambiguous

Implementation Software and Hardware:

Software: Java, Perl, Weka 3.
Hardware: MacBook Pro, Intel Core 2 Duo 2.4GHz, 4GB Memory.

Acknowledgement

First of all, I would like to thank my supervisor, A/P Min-Yen Kan, for his support throughout the entire duration of this project. He was very patient, and always willing to answer my questions whenever I had any doubts. He also gave me advices on the project, and suggestions on how to make things better.

I am grateful to be offered this opportunity to participate in such a research project and be able to meet and work with members of Kan's research group, WING. I would like to thank the WING group for their advices during my practice talk sessions with them. On top of that, I would like to thank Wang Ao Bo and Chen Tao, for their advices and for their time spent on helping me make this report better.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

In the English language, there are many words and even phrases that have multiple interpretations depending on the context. These words have multiple *senses* or meanings, and are ambiguous. For instance:

1. She is interested in the *interest* rates of the bank.
2. He developed an *interest* in art.

It is not difficult for a human to see that the word *interest* has different meanings in the two sentences. We humans are able to determine the context of each sentence immediately, and hence able to identify the correct sense for the ambiguous word. A machine, however, needs to run through a series of analytical processes before it can determine the best answer. This process is termed Word Sense Disambiguation (WSD). More formally, WSD is the process of deciphering the intended meaning of an ambiguous word in a given context.

WSD is a key task in Natural Language Processing. It has many applications in computational linguistics, such as machine translation, text mining and information retrieval. It can also be applied within search engines such that the search results will be more relevant, when the results have the same intended meaning as the search query.

There are four conventional types of WSD, they are:

1. Dictionary-based & knowledge-based

   This method uses formal dictionaries and lexical knowledge bases as their primary source to disambiguate senses. Dictionaries provide the definitions of the possible senses of an ambiguous word. The definitions can then be used in WSD algorithms. One example is the Lesk Algorithm (Banerjee & Pedersen, 2002), that words in a given environment (sentence, paragraph etc) will tend to share a common topic. Banerjee & Pedersen, instead of using standard dictionaries, used WordNet (Fellbaum, 1998) as a sense inventory in their implementation of the Lesk Algorithm. WordNet is arranged semantically, creating an electronic lexical database that provides a rich hierarchy of semantic relations that can be exploited. The authors exploited this advantage of WordNet and integrated into the Lesk Algorithm. They found that the adapted implementation outperformed the traditional Lesk approach with double the accuracy.

2. Supervised

   Supervised WSD uses manually sense-tagged corpora as their primary resource to perform WSD. It can be formulated as a classification problem where word senses represent classes and a classifier assigns a class to a new instance of a word. Some classifiers are the Naive Bayes, Decision Lists (Escudero, Màrquez, & Rigau, 2000) and Support Vector Machines (Bennett & Campbell, 2000).

3. Semi-supervised

   This method makes use of both labeled and unlabeled data. Similarly, it refers to using multiple untagged corpora to provide concurrent information to supplement a tagged corpus.

4. Unsupervised

   Most challenging approach among all, this method may also be related to *Word Sense Induction*, where senses could be induced by analyzing words in a given text. Unsupervised methods perform WSD with minimal, or no, dependence on sense-tagged corpus. (Yuret & Yatbaz, 2010a) described a probabilistic approach based on the Noisy Channel Model that uses an unlabeled word corpus derived from publicly-available web pages. Their system outperformed all the unsupervised systems compared with, some of which the authors claimed that they should be classified as semi-supervised instead.

1.1 Motivation

There are many WSD systems that can provide great accuracies in disambiguating words. However, most of these systems are also not publicly available, either for its applications or for further research. Also, these systems are not suitable to be deployed as real-time softwares, such as for use on mobile devices, as the size of their training models are relatively large to be suitable.

The motivation in this project, other than to encourage the usage of WSD applications, is to build WSD systems that are small, and responsive such that they are more suited to be deployed on mobile devices or web browsers.

1.2 Goals

In this project, we explore how we can build a system that is small, suited for real-time applications. For that, we have considered three goals to measure the performance of our system in this project. They are:

1. Accuracy - percentage accuracy in predicting word senses
2. Speed - time taken to predict word senses and return the results
3. Size of Model - size of the training model used by the system to predict word senses

We intend to build a system that is meant to perform WSD on small amount of text, like a sentence. One application for such a system is a language-assistance based tool to be integrated with web browsers, like a plugin in Firefox, so that users are able to find out more information, like pronunciation and definition, about some words on the web page by selecting the text using the mouse. WSD comes into the picture as we see the need for these information to be context-relevant from where the selected text is from. However we must keep the response time small, so that we can maintain user's experience. One can imagine an user who has the habit of selecting text on web pages very often. Any language tool with slow response will be not ideal.

For that, as much as possible, we intend to keep the model size as small as possible, while retaining a reasonable level of accuracy. We want it to be speedy and responsive. Also, we hope to reduce the amount of pre-processing prior to the prediction of word senses. We believe that by including too many pre-processing features, the system would require additional supporting files that could be significant in terms of file size.

Chapter 2

Related Works

Word Sense Disambiguation has been a key task in the field of Natural Language Processing since late 1940s. In general, supervised WSD has been attaining performance better than the other types of WSD systems, as they utilized large sense-tagged corpus as training model. However supervised WSD systems do have their limitations, especially for their high dependence on sense-tagged corpus, that their WSD accuracies are only as good as how large and fulfilling their training models are. For the long term, we must consider unsupervised systems for their ability to perform independent of sense-tagged corpus.

In order to encourage WSD applications in other fields, we need to consider flexible systems that allow customization. At the same time, we must consider its portability too, in other words its model size and time needed for the WSD process. For that, we have selected the following works to study in detail.

2.1 Word Sense Disambiguation Using Wikipedia

Wikipedia is a free, web-based and collaborative encyclopedia that allows the public to contribute and edit almost all articles Wikipedia has, and therefore we can say that words in these articles are manually tagged. (Mihalcea, 2008) introduced an approach that exploits these manually tagged words, building a sense tagged corpus from Wikipedia to be used for WSD.

In Wikipedia articles, tagged words are in the MediaWiki syntax, which captures information about the word's topic, context or sense. For example, [[**bar(law)**|**bar**]] and [[**bar(counter)**|**bar**]]. The senses of the word *bar*, namely *law* and *counter*, can be derived by extracting them from these annotated links. With this information, Mihalcea built a sense-tagged corpora by first extracting all paragraphs from Wikipedia that contain the occurrences of a given word. Then the senses are mapped onto their corresponding senses in the WordNet sense inventory.

The advantage of this approach is it exploits the dynamic nature of Wikipedia. By using Wikipedia as a source for corpus, one can be sure that it will always be up-to-date and contains any new words or new senses. A disadvantage, however, is data inconsistency. Consider the following: *handphone* and *mobile phone.* Data inconsistency occurs here as different users might refer to the same object by a different name.

We observe that this approach would have the flexibility to handle new words and senses. This is because, considering the popularity of Wikipedia is, any new word or sense used by people, say *tweet*, would most likely appear on Wikipedia faster than any formal dictionary databases. But of course, we must also consider the fact that if the number of new words and senses grows as fast as the number of articles, there would be a need to re-construct the sense-tagged corpus. Alternatively, we may use the APIs on MediaWIki to send query requests to retrieve information about a target word. From that we may build a real-time WSD system that is based on retrieving information from Wikipedia, making it independent of lexical databases and corpora. Performance issues aside, this could minimize the need to reconstruct a corpus on an occasional basis.

2.2 Word Sense Disambiguation Using Dependency Knowledge

In this paper by (Chen Ping, 2009), we focus on its problem formulation and WSD algorithm. The authors formulated the WSD problem into *weighted directed graphs*, and by simply computing values of the weighted nodes, it is effective in telling the dependencies between the words in a given text.

The approach begins with the construction of the corpus. Ambiguous words are sent to web search engines to retrieve the relevant pages. These pages are cleaned, segmented, and then parsed with a *dependency parser*, Minipar (Lin, 1998) to retrieve the parse trees, which are merged, by merging nodes from different dependency relations if they represent the same word, to form the context knowledge base (Chen Ping, 2009). In the weighted directed graphs, the weight assignments and score computations are handled by the *TreeMatching* function described by the authors. It is the score calculator for the weights in the parse trees. A target sentence is passed into the WSD algorithm together with WordNet sense inventory and the context knowledge base built earlier. *TreeMatching* then assigns weights to the nodes based on rules and dependency relation instances, and returns the score of a WordNet gloss that an ambiguous word was compared with. Subsequently, either the sense with the best score or the first sense will be determined as the correct sense.

Undoubtably, the algorithm is effective and accurate in matching the dependency relations to determine the correct senses, as shown in the evaluation results in (Chen Ping, 2009). However, before *TreeMatching* can be done, all the sentences and glosses have to be pre-processed, and parsed into parsing trees. The parsing process, as mentioned by the authors, takes a lot of time. Then, the authors highlighted there is this concern regarding the dependency parser, Minipar. Minipar is not 100% accurate in its parsing so it causes invalid dependency relations in the parse trees. Although the authors claimed that the WSD algorithm will minimize those erroneous output, it was not explicitly defined how it actually did it.

2.3 Word Sense Disambiguation Using Noisy Channel Model

In this paper, (Yuret & Yatbaz, 2010b) describes an unsupervised system that uses a generative probabilistic model for WSD. In their system, they used unlabeled data sets derived from publicly-available web pages instead of sense-tagged corpus such as SemCor. They demonstrated that the Noisy Channel Model can also be used for word sense disambiguation. The main contribution of this method is the reduction of the WSD problem into the estimation of two distributions: the distribution of words that can be used in a given sense and in a given context.

First, they estimated the context-dependent sense distribution without using any sense-tagged data. They modeled it according to the noisy channel model, with each context, C, as a distinct channel where the intended message is a word sense, S, and the signal received is the ambiguous word, W. With that they adopted the conditional Bayes' theorem, as in Equation 2.1. $P(S|W,C)$ is the probability of a sense S of word W in a given context C.

$$P(S|W,C) = \frac{P(W|S,C)P(S|C)}{P(W|C)} \tag{2.1}$$

To perform WSD means finding a sense S that maximizes the value of $P(S|W,C)$, which is equivalent to maximizing the value of $P(W|S,C)P(S|C)$, since the denominator is not dependent on S.

In their experiments the authors compared their system to some of the best supervised and unsupervised systems. Their probabilistic approach outperforms all the unsupervised systems that were compared with. In their paper the authors also noted that some of all the unsupervised systems available should actually be considered as semi-supervised because of their reliance on sense-tagged corpora for training. In other words, their unsupervised WSD had outperformed some of the semi-supervised ones, which should be considered a remarkable feat.

2.4 It Makes Sense

It Makes Sense (IMS) is a supervised English all-words WSD System introduced by (Zhong & Ng, 2010). The main contribution of IMS is being an open source system that allows users to customize it by integrating different preprocessing tools and additional features.

Another contribution is that the system addresses the issue regarding the lack of sense-annotated training examples, which is critical in the overall performance of a supervised WSD system. On top of using the widely used SemCor corpus, the authors also collect training data from six parallel corpora from the Linguistic Data Consortium. The idea is maximize the number of training examples for each of the words, ambiguous ones especially, to overcome the lack of sense-annotated training examples.

Regarding the performance, on a default configuration of modules, IMS attained state-of-the-art accuracies on all-words and lexical-sample tasks, with performance comparable to the top performing system compared to in the paper.

2.5 Using Decision Trees Of Bigrams To Predict Word Sense

In (Pedersen, 2001) presents a corpus-based approach to WSD where a decision tree assigns a sense to an ambiguous word based on the *bigrams* that occur nearby. The author took this approach because surface lexical features like bigrams often contribute a great deal to disambiguation accuracy. In this paper, the author demonstrated that Decision Trees can be used as an accurate predictor for word senses.

It begins with building the feature set of bigrams. Pedersen defined bigrams as two consecutive words that occur in a text. He used Power Divergence Family and Dice Coefficient to select the bigrams to be added to the feature set, so that only the useful and interesting bigrams are added into the system.

The feature set is then applied onto decision trees. The prediction process involves applying test instances on the decision trees, searching for a path through the decision tree from the root to the leaf node, which captures the prediction. In the system the Weka[1] implementation of the C4.5 decision tree learner was used.

We included this work in our review for the same idea of using decision trees for word sense prediction. However in the current implementation of our system we do not include bigram or

[1]http://www.cs.waikato.ac.nz/ ml/

collocation features. We consider this as a possible option in our future work.

2.6 Conclusion

From the above works, we have learnt the importance of a sense-tagged corpus for a supervised WSD system. It is important for a system to have sufficient, if not abundant training examples in order to attain higher WSD accuracies. However the disadvantage for relying on sense-tagged corpus is the constant need to update the training model used by the system. (Yuret & Yatbaz, 2010b) noted that in general, increasing the size of the training model only improves its performance by 3 to 4%. This is not cost effective as manual tagging is time consuming and costly.

An idea worth mentioning is regarding the *portability* of the system. By portability we refer to the system's training model's size, and the time taken to perform WSD. In order to encourage WSD applications, there is a need to reduce the size of the system. Also, in most papers related to WSD, little was mentioned regarding reducing the model size and time taken.

With the above in mind, we describe our approach as follow.

Chapter 3

System Description

Following the study of the above works related to Word Sense Disambiguation, we find the need to build a system, which is directed at minimizing the size of the training model and the time taken to predict word senses, more significant. For that, we populated the following characteristics that we want in our system:

1. Requires minimal pre-processing for training and test inputs
2. Training model must be small, and easy to interpret
3. Perform speedily

Decision Trees is one of the simplest predictive model and thus it is simple to interpret and understand. It requires little data preparations, and able to have value even with little hard data. It is able to process large amount of data in a short time. With that, we decided to implement our system using Decision Trees. We use the Weka[1] implementation of the J48 decision tree learner to generate the model files to help us predict word senses.

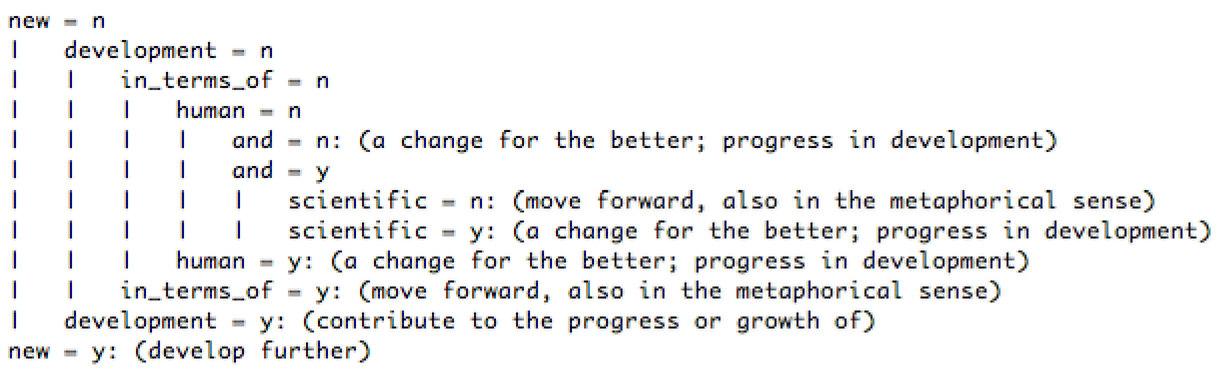
The intuition of our method is that a sentence is the *environment* that helps determine the word senses of ambiguous words. Consider a simple case where there is only one target word that needs to be disambiguated. We look at the surrounding words in the same sentence, such

[1]http://www.cs.waikato.ac.nz/ml/weka/

that if there exists a particular set of determining words, we predict that the target word would possess a specific word sense.

We used the SemCor1.7.1 corpus as our training data set. We used the tagged Brown Corpus files that have all content words tagged. We were able to identify a total of 20450 *lemmas*. By *lemma* we refer to the root form of a word. For example, the root form for the word "ate" is "eat", and root form for "runs" is "run". We predict word senses based on the sense keys extracted for these lemmas. Since our approach predicts word senses by looking at the entire sentence, we parsed the training corpus and saved it into numerous sentences. Then for each lemma, we save a sentence as an *instance* for that lemma if the sentence contains that lemma.

For each lemma, we generated the model files with pruning to remove any irrelevant branches in the trees. Moreover, itreduces the size of the trees, and will be even smaller and more efficient to store and use. In these model files there are **2261** trees that are *unary*, that it is simply a leaf node, and captures only one result for word sense. After checking against the original corpus, it was found that these unary files were simply due to the fact that these lemmas had only one instance of occurrence in the corpus and thus only one word sense. For the other trees, their *depths* range from **5 to 49**, and the number of *leaf nodes* ranges from **2 to 25**. Figure 3.1 shows an example of the tree structure (definitions in brackets are displayed only for illustration purpose).



```
new = n
|   development = n
|   |   in_terms_of = n
|   |   |   human = n
|   |   |   |   and = n: (a change for the better; progress in development)
|   |   |   |   and = y
|   |   |   |   |   scientific = n: (move forward, also in the metaphorical sense)
|   |   |   |   |   scientific = y: (a change for the better; progress in development)
|   |   |   human = y: (a change for the better; progress in development)
|   |   in_terms_of = y: (move forward, also in the metaphorical sense)
|   development = y: (contribute to the progress or growth of)
new = y: (develop further)
```

Figure 3.1: Tree for *advance*

3.1 Predicting Word Sense

Prediction of a target word begins with capturing the entire sentence the target word resides in. This is the *environment* we mentioned in our intuition. Consider a sentence with only one ambiguous word to be disambiguated.So we use the rest of the sentence to predict the word sense of that target word. Consider a set of all English words, $\varepsilon = \{\text{all English words}\}$. Ignoring the order of words and repeated usage of the same word in the same sentence, an English sentence can be denoted as a set of English words, as such: $S_i = \{a_1, a_2, \ldots, a_k\},\ S \subseteq \varepsilon$. Therefore, suppose we perform WSD on a_j, we consider the set $S - \{a_j\}$ as its *environment* that determines its word sense.

In order to test for a_j, we check our training data set for existing data about a_j. Using the set $S_i - \{a_j\}$, we generate a single data instance to test against our training data. From there we utilize the Weka's Java packages to predict the word sense of a_j.

We test our system for the SensEval and SemEval all-words tasks. We pre-processed the test corpus such that we were able to test sentence by sentence. For all the sentences, we removed all quotations and punctuations, retaining only the English words as we only consider the actual words for word sense predictions.

In the following sections, we describe our implementation of the system in iterations. We will also provide the test results at each respective stage. We conducted our experiments using SemCor1.7.1 as the training corpus for all the test corpus we were testing against. Our *Baseline* is the WordNet-Sense-One, which assigns the most frequently tagged sense as the answer. The results of the WSD accuracies are shown in Table 3.3.

3.2 Iteration 1 (DT v0.1)

Suppose we are performing WSD for every word in a sentence, a single-instance test file is generated for every testable word in the sentence. Then for each test file we run the prediction process to get the word sense. Consider the following example:

Before: the **quick brown** fox jumps **over** the lazy **dog**

After: the **quick(1) brown(1)** fox jumps **over(2)** the lazy **dog(1)**

Figure 3.2: Output from iteration DT v0.1

In this example, the bold words are ambiguous, and we run them through the prediction process. The numbers in brackets are the results of the prediction process, and each of them refers to their corresponding word sense. The actual sense keys are stored in a different location to make the output cleaner.

In this iteration, the only feature we used is the occurrence of the neighboring words of a testable word, the *environment* as we defined. As we were experimenting on this iteration, we quickly noticed that the speed of the prediction process was very slow. On average, the time taken to perform WSD on a sentence can easily be **10 to 40 seconds**. We picked a few random sentences from the test corpus to make comparison with the IMS system by (Zhong & Ng, 2010), and the IMS performs at least 70% faster than our system. Also, we compared our model size with IMS's, and our system's model size was twice as large (refer to Table 3.1). We address this two issues in Iteration 2.

	Model Size
IMS	$\approx$ 275MB
DT v0.1	$\approx$ 545MB

Table 3.1: Model size comparison

3.3 Iteration 2 (DT v0.2)

We found that in our model files, there are a lot of information that we can omit. More specifically, we only require the actual *tree* itself for the prediction process, like the one illustrated in Figure 3.1. Hence we extracted only these trees, and further compacted the output such that each tree is now represented in a single line, as illustrated here:

advance>new=n!|development=n!||in-terms-of=n!|||human=n!||||and=n:1(4/1)!||||and=y!. . .

With this we compacted all trees into a single file, having the size of approximately **1 MB**. At this point, we were able to keep our model size very small, and to fulfill one of our goals: Size of Model.

Since the format of the model files was modified, the prediction procedures were also modified. In the modified procedures we dropped the usage of Weka to process the test instance. So we also did not need to generate a temporary test instance file to test against the existing file in our system. Instead, we used the trees extracted and treated each node as a test condition. So prediction means testing against nodes of the tree, eventually leading to a leaf node that provides a word sense.

We evaluated this iteration of our system to first make sure that the WSD accuracy remain the same, and also to determine the time taken. In the previous version, the prediction process for just one sentence can easily take 10-40 seconds, depending on length of the sentence. In the new procedure, all the trees are captured within a single file and we load the entire file into memory during the prediction process, in the form of a hash. With that, we were able to process all sentences from the three test corpus we tested against in approximately 2 minutes, averaging at about **0.10 seconds** per sentence. Compared to the previous iteration, we were able to reduce the time taken by at least **98%**. With that we had also fulfilled the other goal: Speed.

	Before DT v0.2	DT v0.2
Model Size (MB)	545	**1**
Time Taken (secs per sentence)	10 to 40	**0.10**

Table 3.2: Comparison of model size and time taken

	SensEval-2 (%)	SensEval-3 (%)	SemEval-2007 Coarse-Grained (%)
Baseline (WNs1)	55.9	48.8	64.3
DT v0.1	48.5	45.5	57.8
DT v0.2	48.5	45.5	57.8
DT v0.3	49.2	46.2	58.3
IMS	68.2	67.6	82.6

Table 3.3: WSD accuracies on SensEval and SemEval all-words tasks

3.4 Iteration 3 (DT v0.3)

In the previous iteration, we did not lemmatize the attributes in our training data files. We speculated that if we lemmatize the attributes, we should be able to get better results compared to the previous implementation. By lemmatizing the attributes, we remove redundant attributes. For example, "ate" and "eat", where "ate" is considered redundant, since our system does not consider the tense of words. By doing so we also remove redundant nodes in the trees generated from the decision tree learner. For that the size of the new model trained from a new version of training data set was further reduced by **2%**, from 1,052,703 to 1,032,414 bytes. Zooming in to each lemma's tree, we have reduction of tree sizes of up to **22%**.

In this iteration we also added the *confidence* feature, whereby if the confidence in the prediction is too low, we will switch to using the first sense of WordNet, that is, the sense with most frequency count. The value of confidence ranges from 0 to 1. As illustrated in Figure 3.3, the confidence values are now reflected alongside the prediction.

Before: the **quick brown** fox jumps **over** the lazy **dog**

After: the **quick**(**1:0.71**) **brown**(**1:0.97**) fox jumps **over**(**2:0.92**) the lazy **dog**(**1:0.98**)

Figure 3.3: Output from iteration DT v0.2

We evaluated our system with this new iteration and we attained improvements for all the corpus we tested against (refer to Table 3.3). With this we demonstrated that even with a reduction of model size, we were able to attain an improvement in WSD accuracy.

For the time taken to perform WSD, we evaluated our system against the IMS system. Our system's model size is approximately 1MB, whereas IMS's is about 275MB. During the WSD process, both systems would need to load their models into memory. Clearly, loading IMS's model would take a longer time. In order for the comparison to be more fair, the number of test sentences have to be large enough so that the loading time would be less significant in the total amount of time taken. For that we use all the sentences we parsed from the test corpus, SensEval-2, SensEval-3 and SemEval, to form a single test file of **789** sentences.

	IMS	DT v0.3
For 789 sentences	50.322 secs	81.148 secs
Time Taken per sentence	0.0637 secs	0.103 secs

Table 3.4: Speed Test Experiment 1

Our system loads the model into memory every time it performs WSD on a new sentence. For IMS, its model files are loaded only once to perform WSD on all 789 sentences. The time taken for this speed test is summarized into Table 3.4. Clearly, when performing WSD on large amount of text, IMS performs about **38%** faster when compared with DT v0.3.

However this is not the intended nature for our system. Our system is meant to only perform WSD on a single sentence. In order to illustrate how our system actually performs, we need to perform a second speed test.

	IMS	DT v0.3
For 62 sentences	17.636 secs	**7.611 secs**
Time Taken per sentence	0.284 secs	**0.123 secs**

Table 3.5: Speed Test Experiment 2

In this second speed test, we picked **62** random sentences from the test corpus. We ran these sentences through both system for **10** rounds each, and computed the average score for each round. For this speed test, we used the original DT v0.3 to compare against IMS. Making reference to the figures in bold in Table 3.5, our system is faster in this speed test, when the setting is only to test for small amount of text. The speed performance of our system fulfills one of our goals: Speed.

Chapter 4

Conclusion

In this paper we focused on the need of a Word Sense Disambiguation (WSD) system that is directed at keeping the model size and time taken small. We found that even though there is much study made on WSD, few focused on introducing *portability* into the system. For such a system, we intended to keep the size of model to minimal, while attempting to retain a reasonable level of accuracy. We opted to reduce the amount of pre-processing, like POS tagging, prior to the prediction of word senses. We believed that by doing so we incur additional supporting data, which increases the size of model. Also, we wanted to build a system that is speedy and responsive, so as to enhance users' experience. We wish to implement such a system so that we can perform real-time WSD on platforms like web browsers, or even mobile devices. On these platforms, we expect users to only perform WSD on small amount of text, like a sentence. For the project we identified three goals, namely: Accuracy, Speed and Size of Model.

We used Decision Trees in our system. It is simple to interpret and understand. It requires little data preparations, and able to have value even with little hard data. It is able to process large amount of data in a short time. We use the Weka implementation of the J48 decision tree learner to generate the model files to help us predict word senses.

In our system, we used SemCor1.7.1 as our training corpus. Following our intuition on how the system would work, we extracted the training data sentence by sentence, so that each sentence serves as the *environment* for each ambiguous word. For evaluation we tested our system against the all-words tasks of SensEval-2, SensEval-3 and SemEval.

We acknowledge that our system produces low WSD accuracies. It seems a better choice to use the Baseline, which is fast, only requires a sense inventory file of about 7MB and has a higher accuracy. However there is still plenty of room of improvement for our system. As of now, the *only* information that our system use is the occurrences of neighboring words. It does not have any other information, such as parts-of-speech, in the training and testing sentences. There are many other surface lexical features that can applied into our system. For example, we can add the *bigrams* feature as described by (Pedersen, 2001). Otherwise, we can adopt collocations, which is also featured in IMS's system. Considering the potential for growth, our system is still better than the Baseline.

SensEval-2 (%)	SensEval-3 (%)	SemEval-2007 Coarse-Grained (%)
67.6	65.6	84.7

Table 4.1: Corpus overlap with SemCor1.7.1 (with respective test corpus)

We conducted additional post-experiment analysis and it helped us identified that the amount of overlap of the training and test corpus affected the WSD accuracies. Comparing Table 3.3 and 4.1, we can see that the percentages in overlap are consistent with the performances in various test corpus. In other words, the higher the overlap, the higher the performance.

Though our analysis on the amount of corpus overlap showed that it played a part in the performance, it also showed the weakness of our system's inability to process *unknown* words to the system. An immediate course of action from this point might be to increase the amount of training data, rather than only using SemCor1.7.1. Since our system captures training instances by sentences, we can *learn* from WordNet itself, as WordNet do provide sample sentences for most of the words in its inventory. The advantage in this move is that it would expose the system to more words and senses. However, it also increases the system's dependency on a lexical database. Increasing the size of training corpus would directly increase the size of the model and conflict with one of our goals — to reduce the size of model in order to make the system portable. Furthermore, in (Yarowsky & Florian, 2002), the authors reported that doubling the corpus would only reduce errors by 3 to 4%. Therefore, in the longer term, we wish to introduce some *unsupervised* characteristics to the system, so that it is less dependent on back-end resources.

Through the various iterations of our system's implementation, we demonstrated that we reduced the time taken and size of model needed to predict word senses. Prior to switching our system to use a modified model, the prediction process easily took 10-40 seconds to process a single sentence, and the size of model, at more than 545MB, was very large. We changed how we store our model and how we perform the WSD procedure, we reduced the time taken to an average of 0.10 seconds per sentence, and the size of model to only 1 MB.

We claimed that our system is able to perform speedily and responsively. To justify that we conducted 2 speed tests. The conclusion drawn from these speed test is our system performed at least twice as fast compared to the IMS system when performing WSD on a single sentence. This satisfies our intention to implement such a system into real-time applications like web browsers and mobile devices. We make reference to previous work done on DiCE Translator[1] prior to this system. As a Firefox plugin, DiCE's purpose is to provide bilingual translations for text on web pages to encourage language learning across a second language. Despite after extracting dictionary content from Wiktionary[2], DiCE lacks the ability to translate accurately according to the context. Integrating our system into DiCE, we plan for DiCE to be able to translate more relevantly, while still maintaining the overall responsiveness in using the tool.

There are many systems that attain high accuracies in WSD tasks. However, few the take size of model and time taken into consideration. We demonstrated that our system is able to keep both the model size and time taken to the minimal. To conclude, we had successfully created a system that is light-weight and speedy.

[1]https://addons.mozilla.org/en-US/firefox/addon/dice-translator/

[2]http://www.wiktionary.org/

References

Banerjee, S., & Pedersen, T. (2002). An adapted lesk algorithm for word sense disambiguation using wordnet. *CICLing '02: Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing* (pp. 136–145), London, UK, 2002: Springer-Verlag.

Bennett, K. P., & Campbell, C. (2000). Support vector machines: hype or hallelujah? *SIGKDD Explor. Newsl.*, *2*, December, 2000, 1–13.

Chen Ping, Ding Wei, C. B. . D. B. (2009). A fully unsupervised word sense disambiguation method using dependency knowledge. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics on - NAACL '09*, (June), 2009, 28.

Escudero, G., Màrquez, L., & Rigau, G. (2000). A comparison between supervised learning algorithms for word sense disambiguation. *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning - Volume 7*, ConLL '00 (pp. 31–36), Stroudsburg, PA, USA, 2000: Association for Computational Linguistics.

Fellbaum, C. (Ed.). (1998). *Wordnet an electronic lexical database.* Cambridge, MA ; London: The MIT Press.

Lin, D. (1998). A dependency-based method for evaluating broad-coverage parsers. *Nat. Lang. Eng.*, *4*(2), 1998, 97–114.

Mihalcea, R. (2008). Using wikipedia for automatic word sense disambiguation, 2008.

Pedersen, T. (2001). A decision tree of bigrams is an accurate predictor of word sense. *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL '01 (pp. 1–8), Stroudsburg, PA, USA, 2001: Association for Computational Linguistics.

Yarowsky, D., & Florian, R. (2002). Evaluating sense disambiguation across diverse parameter spaces. *Nat. Lang. Eng.*, *8*(4), 2002, 293–310.

Yuret, D., & Yatbaz, M. A. (2010a). The noisy channel model for unsupervised word sense disambiguation. *Comput. Linguist.*, *36*(1), 2010, 111–127.

Yuret, D., & Yatbaz, M. A. (2010b). The Noisy Channel Model for Unsupervised Word Sense Disambiguation. *Computational Linguistics*, *36*(1), March, 2010, 111–127.

Zhong, Z., & Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. *Proceedings of the ACL 2010 System Demonstrations*, ACL ’10 (pp. 78–83), Morristown, NJ, USA, 2010: Association for Computational Linguistics.