

Some Experiments in Word Sense Disambiguation

A report for comp791a Project

Shahin Kamali (6023614)

s_kam@encs.concordia.ca

1 Introduction

The automatic disambiguation of word senses has been an interest and concern since the earliest days of Natural Language Processing. Sense disambiguation is an “intermediate task” which is not an end in itself, but rather is necessary at one level or another to accomplish most natural language processing tasks (semantic level).

The problem of word sense disambiguation has been described as *AI-complete*, that is, a problem which can be solved only by first resolving all the difficult problems in artificial intelligence (AI)¹ [1] it means that solving this problem is equivalent to solving the central Artificial Intelligence problem which is making computers as intelligent as people².

Although using statistical methods has been very successful for some of important problems in Natural Language Processing such as Part Of Speech tagging and alignment of parallel translation, an effective method for word sense disambiguation does not exist yet, consequently this problem is frequently cited as one of the most important problems in natural language processing research today.

This report reflexes my study and also results of a code implementation of some experiments related to Word Sense Disambiguation.

¹ Term “AI_Complete” in Artificial Intelligence is inspired by “NP_completeness” in complexity theory.

² Samples of other AI_complete problems are Computer vision, Natural language understanding and Passing the Turing Test.

2 My Approach, Tools and Techniques

Overview

There are many approaches for solving word sense ambiguity, of which supervised learning the most successful are.

In this project I start with a Naïve-bayes method, train it (using word sense tagged text) and analyze its results. Also some information coming from the entries of a dictionary (WordNet) are used to refine the trained system. Finally Spectral Graph Transductive learning method would be surveyed for Word Sense Disambiguation³.

Training Corpora

While using a supervised learning, I need previously tagged training corpora. I chose documents of SemCor as the training data; in this way I can use SemCor as a gold standard to compare results of my work with other researches. Another reason of choosing SemCor is that it is free⁴. Details about training Corpora would be explained later

WordNet

Tags in SemCor are based on tags of WordNet. Generally WordNet is extremely required in this project.

While there are lots of WordNet interfaces for Java, Perl, Python, etc. Unfortunately there is no specialized interface for C++⁵. So I used WNCOM⁶, which is a Component object, which works as an interface between my code and WordNet database. Also using this COM object I wrote a WordNet browser and attached it to my program⁷.

Evaluation

To evaluate the disambiguation method in each experiment, I use Senseval-3 to have a common framework to compare the WSD systems. The chosen task is Senseval-3 All-Words Task. Consequently I use the appropriate Scorer⁸ provided by Sense_Val3 to compute precision and recall of systems outputs.

To be able to work with this Scorer I use Version 1.7.1 of WordNet and also SemCor 1.7⁹.

³ Items mentioned in this paragraph would be clarified in the following pages.

⁴ I tried a lot, but could not find free tagged corpora of MEDLine

⁵ There exist some interfaces for c++ that are not confirmed by Princeton.

⁶ There is a link to WNCOM in official website of WordNet (in menu Related Projects)

⁷ Actually this part was one of the most time consuming parts of this project.

⁸ That is just a code in C.

⁹ The most recent versions of these two are 2.1

Testing Data

For all experiments we use Test Data provided Senseval-3 as test data. Apparently this data is not used as a training test at all. We do not look this data during training phase..

3 Software

The implemented program does not use any language modeling toolkit, available code, etc (except the COM object mentioned before). I autonomously implement everything. Even some classes available in language libraries are manually implemented (like hash-tables and a special class for graph). Consequently the result is a short portable code.

Choice of Programming Language

The programming language is C++ and the programming environment is Microsoft Visual Studio 6.0. The main reasons for using C++ are explained in the report of assign#1. Also I chose “Microsoft Visual Studio 6.0” programming environment since it has powerful tools for creating a user-friendly program.

An Interface for WordNet

As mentioned before I used a COM object to connect to the WordNet. This can be very useful for the people programming C++ in Microsoft Windows.

How to Work With the Program

First of all you should register the file sordnet.dll to the registry of your operating system.

After that it would very easy to run the code. It is enough to open the project file (NLP.dsw) in Microsoft Visual Studio 6.0 and execute or compile it. Also you can transfer the code to Microsoft Visual Studio .Net (In this case a little change in header files is necessary).

I have done my best to make a very user-friendly program. The software is a dialog based (also containing menu) application and user can set everything using edit-boxes, radio buttons, lists, etc. There are some facilities for opening and saving training and testing corpora and generated results.

4 General Notes about Experiments

A Baseline

We use WordNet to define a baseline for the system’s operation. In WordNet, a frequency tag is assigned to each sense entry. The entries with higher frequency have more chance to accrue. In this way if we assign the first sense of WordNet (which is the most probable sense) to get a baseline. The following table shows the precision and recall and also of the test corpus (Senseval3-all word task test data).

Table 1

Test	Precision	Recall	Attempted
No stemming	0.320	0.242	75.75
Stemming applied	0.384	0.379	98.87

Stemming

To disambiguate a word in test data, first of all we need stemming to find the root of that word. It is because the WordNet entries are all stemmed. For example if you search in WordNet for the word “is”, you would not get a WordNet entity¹⁰. Instead, you would be linked to entities with word “be”¹¹.

I used the stemming functions offered by the WNCOM. (WNCOM itself uses functions offered by WordNet). This stemming function is just a set of morphological rules that are classified in some methods (considering the word’s Part of Speech”).

It is also possible that a word has both stemmed and not stemmed tags. For example the word “said” has 11 senses in which it is stemmed to the verb “say” and 1 sense in which it is an individual adjective. My code does support both possibilities in cases like this.

We have also ambiguity in stemming. Consider the word “Was” can be both stemmed to WA (Washington) as a noun and also to the word “be” as a verb. In the implementation I get the priority to verb stemming. However it is better to use part of speech tagging to choose the correct stemming.

As Table 1 shows, if we do not apply stemming in our system, recall and also percentage of time that the system attends to answer (regardless of accuracy of this answer) would be less. It means the system ignores a lot of test cases since it has no idea about them.

Data Structure

Although Data Structure may look be a matter of implementation, I feel it is necessary to have an idea about the used data structure, since having a good data structure is necessary while working with huge size data (on main memory).

Figure 1 shows interconnection between objects and also their connections to the WordNet. As you can see I used a graph representation for showing the relationship between sense objects (vertexes). Also this view showsthat each object of the class Sense is equivalent to an abstract “Synset” in WordNet.(one to one relationship)

¹⁰ All entities in WordNet are indexed through “Sense Key”. Consequently each entity has a unique Sense Key. For example consider the sense key “be%2:42:03::”

¹¹ Here the word “be” is a called the lemma of the wordnet entry. This Lemma is the first part of Sense Key mentioned before.

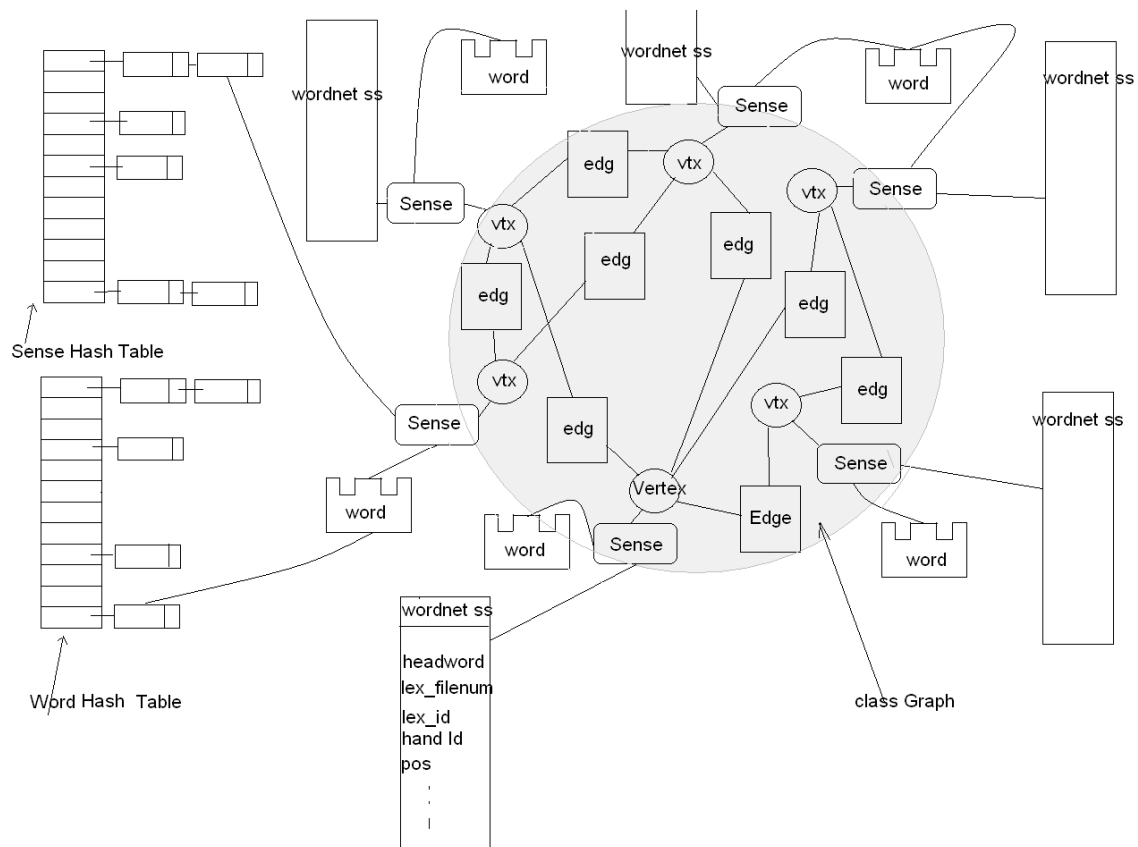


Figure 1 interconnection between objects and also their connections to the WordNet abstracts.

This data structure simplifies implementation in all experiments. Also everything is implemented using pointers and most of the time there is no need to copy object data (this is very important to save the main memory)

5 A naïve Biased Approach

We know that Naïve Biased approach is a supervised learning mechanism which tries to find s^* in the following equation:

$$s^* = \operatorname{argmax}_{s_k} P(s_k) \prod_{j=1}^n P(v_j | s_k)$$

The most important problem in this approach is the sense and data sparseness. In other word, for achieving high performance we need to have sufficient tagged data (which is too costly).

Some of the problems that may accrue are due sparseness are:

In testing data..

- We face a context word (in training set) that is not happened in training source¹².
- We do not see a sense s_k in the training set. As a result $p(v_j|s_k)$ would be zero -
- for all v_j . (s_k would be eliminated from the set of senses) We have seen both v_j and s_k , but not in the same window. So $(v_j|s_k)$ would be zero in this case.

To solve the first problem, we change the definition of window (context vector). I define the context vector of an ambiguous word be the bag¹³ of “previously seen” words surrounding it. In this way the words that cannot help in disambiguation would be eliminated.

About the second and the third problems, please mention that there are 207016 different senses in WordNet2. This huge number shows that it is very probable that the system do not ‘learn’ some senses in the training set.

To solve these problems, we start with a simple smoothing method called NG-Method.¹⁴ In this method whenever we have $P(v_j|s_k) = 0$, we define the value of $P(v_j|s_k)$ to be $P(S_k)/N$ In which N is the vocabulary size of training set¹⁵.

In this experiment, we consider 4 test cases that are shown in table 2. Each case is a set of randomly selected documents. Table 3 shows the precession and recall of the learnt system. (the size of windows is defined to be +-3)

Table 2

Test	Number of documents	Accumulative number of word senses
#1	1	1371
#2	2	3284
#3	5	10544
#4	10	23618

¹² It is due to Zipf’s law.

¹³ In next chapters we change this definition to an ordered set.

¹⁴ Suggested by NG in [4]

¹⁵ To have it mathematically correct an extra step is necessary in which we have to define new probabilities to previously seen events. However it is not mentioned in most of the papers.

Nom of used documents	Precision	Recall	Accumulative number of word senses
1	0.351	0.084	24.01
2	0.391	0.117	29.99
5	0.392	0.172	43.85
10	0.408	0.234	57.28

Table 3

As you can see, both precision, recall and Accumulative number of word senses get better when the size of learning data gets larger. Here the size of training data is rather low, consequently the parameters are very bad. (When use just one document to train learning method, the recall is even worse than based line)

Also you can see a high percentage of cases are ignored and it shows that learning data has not been sufficient enough.

So we can essentially improve the system using more and more documents, however we can do something, which costs less. We can change the size of windows defining context vector. It is expected that increasing the size of windows, improve the functionality of the system. We test this hypothesis by applying the method with different windows sizes on the same test case #4 of Table 2

Window size +-	Precision	Recall	Accumulative number of word senses
3	0.362	0.072	19.75
10	0.392	0.172	43.85
100	0.626	0.253	48.67
500	0.816	0.541	66.29

Table 4

As Table 4 shows, as the size of window (context vector) gets larger, the precision and recall of the system improves too.

6 Transductive Learning Approach

As mentioned before Naïve biased word sense disambiguation like other systems which are based on supervised learning is still limited in that it can not work well for all words in a language. One of the main reasons was the lack of sufficient training data.

One way to overcome this problem is using semi-supervised learning systems, in which the learning system can have access to the test data too. For example in k -nearest-neighbor problem, the goal is to classify the the $n - k$ items in classes headed by the other k elements, here that k elements are interoperated as the training data, others a the test data. Obviously we can have access to training data in this case (to know some of its characteristics that can help in learning) . This kind of learning systems is also known as transductive Learning.

Spectral Graph Transduction (SGT)

Spectral graph transduction is a new method in transductive learning introduced in (Joachims 2003). Given a set of labeled and unlabeled examples, the task of SGT is to tag unlabeled examples with either -1 or $+1$. When we want to use it in WSD, we say that we have n instances of a word whose sense can be either ‘-1’ or ‘+1’. Also we know the label of k of these words. Now the task is to disambiguate other $n-k$ senses. To do that, we build a graph in which each vertex is an instance of a word (one to one mapping) and each edge contains a weight which is the similarity of two words (vertexes). Also we know that some of the vertexes (instances) are already tagged as $+1$ or -1 (sense 1 or sense 2 of the ambiguous word)

Consider the following graph. You can see the vertexes contain instances of ambiguous word fish. We know the sense of some vertexes. Consider the dark grays express “Instrument” and light grey “animal”.

Other white vertexes are to be classified into dark or light grey. To do that, we try to find the minimum cut of the graph. For examples the two sets of vertexes shown in figure2 express a cut. The weight of a cut is the summation of weights edges crossing it. In Figure 2 crossing edges are starred.

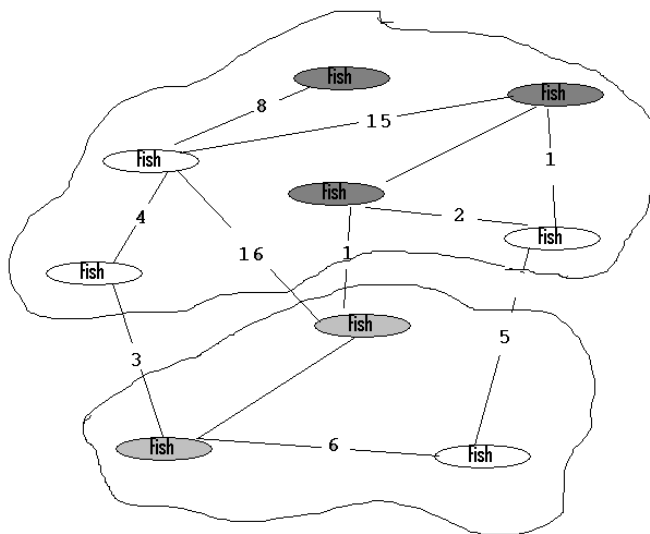


Figure 2

If we can find the cut with the lowest weight (cost) then we can label, all unlabeled data based on its cut. But there are some constraints. First is that each pair of previously labeled vertexes with the same labels should be in the same cut.

The second problem is that finding the minimum cut problem is an NP_Complete problem, It means probably (P?=NP) there is no polynomial algorithm for it. So we have to use approximation algorithm for findings the minimum cut:

$$\min_{\vec{y}} \frac{\text{cut}(G^+, G^-)}{|\{i : y_i = 1\}| |\{i : y_i = -1\}|}$$

Another problem is that, what should we define the weight of edges be?

The graph g is usually defined as a k -nearest neighbor graph. It means each term is connected to k terms which are nearest to it.¹⁶

What happens if the word “fish” has $k > 2$ senses?

In this case, my labeling indicates a selection tree. We divide the potential senses into two subsets, every where the instance places, do the same algorithm to that set.

Applications

The most important application is automatically generating semantically tagged. In this way we can provide sufficient data for supervised algorithm to work better.

My approach

I start with a non-tagged data and use Naïve-Biased method to tag it. I know the results are not good (Tables 1,2,3,4) But about some of tags I am sure that provided tag is correct. I can do that, if the probability (or priority or...) of the tag is more than others [this probability is driven out of more instances in training corpora], its tag would be a classifier for other instances,

Then I use the tags with higher reliability to build SGT and classify unlabeled data (words). Even I can define a reliability function for SGT too and make an evolutionary algorithm out of these two processes.

However in implementation, I had some problems.

I need an approximation algorithm for finding minimum cut and each approximation can be wrong.

The time complexity of this algorithm is rather high. Consider for Just one ambiguous word, we need to build some graphs and solve an NP_Complete Problem for it!! , which is not applicable for huge corpora like those of NLP.¹⁷

¹⁶ My Implementation shows this factor may not be appropriate enough for NLP. At least there is no scientific reason for this.

¹⁷ The most important problem was lack of time. I was working on this algorithm till one day after the deadline of project, but unfortunately the progress was not that much. So in this report I do not present any result about this part of project [I think other parts can somehow be sufficient as a project]

7 References

- [1] *Word Sense Disambiguation: The State of the Art* (PDF) A comprehensive overview
By Prof. Nancy Ide & Jean Véronis (1998).
- [2] Rada Mihalcea, Dan I. Moldovan “*An Automatic Method for Generating Sense Tagged Corpora*”.
- [3] Thorsten Joachims . *Transductive Learning via Spectral Graph Positioning* In *ICML-2001*.
- [4] Thanh Phong Pham, Hwee Tou Ng¹, Wee Sun Lee “*Word Sense Disambiguation with Semi-Supervised Learning*”.
- [5] H. T. Ng, ‘Exemplar-Base Word Sense Disambiguation: Some Recent Improvements’, in *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing, EMNLP*, (1997).

Appendix I

A view of the implemented program:

