Word Sense Disambiguation using supervised and unsupervised methods based on Lexical Chains

A THESIS

submitted by

SANJAY GURUPRASAD

for the award of the degree

of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF PHYSICS

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

Thesis Certificate

This is to certify that the thesis entitled Word Sense Disambiguation using supervised and unsupervised methods based on Lexical Chains, submitted by **Sanjay Guruprasad**, to the Indian Institute of Technology, Madras, for the award of the degree of Bachelor of Technology, is a bona fide record of the project work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. Ravindran

Project Guide

Associate Professor,

Dept. of Computer Science and Engineering,

IIT-Madras, 600 036

Place : Chennai

Date :

Acknowledgements

This work would not have been the same without the people that helped create it. I would like to begin by expressing my heartfelt gratitude to my guide, Prof. Ravindran. First, for all his help, guidance and direction - he allowed an engineering physics student to roam into unexplored territories and made sure he didn't get lost. The learning curve on this project has been fantastic. Second, for being like a friend - flexible, accommodating and encouraging - it has been great to have this opportunity to work with him. And third, of course, for being an awesome fellow Apple fan.

I would also like to thank my co-guide, Prof. Aditi for readily helping me take up this project in the Computer Science department and also helping me with my self study course. My faculty advisor, Prof. A.R. Ganesan was kind enough to allow me to pursue my interest in NLP and for this and for all his support over the last 4 years, I would like to thank him profusely.

I would like to take this opportunity to thank a very special group of friends I found here at IIT Madras - they know who they are. Thank you for everything.

Finally, I would like to thank my parents - from best friend to role model to help desk - they pretty much do it all. Their love, help and support has been invaluable - my father's 24x7 sql help line and my mother's constant reminders to eat carried me through this final semester. They will always be my greatest inspiration. Steve Jobs comes second.

Sanjay Guruprasad

May, 2012

Abstract

Keywords : Word Sense Disambiguation, Supervised WSD, Unsupervised WSD, lexical chaining, Naive Bayes Classifier, Galley-McKeown's Algortihm

Many words in the English languages are polysemous - they have more than one meaning. The meaning of such words depends on the context. When a polysemous word is present in a body of text, a particular sense of the word is activated based on the context. It is critical to understand which sense of a polysemous word is activated in a given context to derive meaning from a text containing polysemous words. Word Sense Disambiguation (WSD) is defined as the ability to computationally determine which sense of a word is activated by its use in a particular context. WSD has been a popular area of research and a number of supervised and unsupervised methods have been constructed to perform WSD. Supervised methods use Machine Learning techniques to perform WSD.

In texts, words that are related semantically come together to collectively convey an idea. Lexical chaining is the process of identifying sets of semantically related words in a text. These sets of words, which we call lexical chains, provide a rich representation of the text and are used in a variety of Natural Language Processing tasks. When a lexical chain is constructed, it is based on the meaning of words in a text, thus lexical chaining algorithms intrinsically perform WSD during the construction of Lexical chains. Galley-McKeown's lexical chaining algorithm, boasts good WSD performance compared to other Lexical Chaining algorithms.

In this work, we construct an ensemble classifier, consisting of a supervised classifier (Naive Bayes Classifier) and Galley McKeown's algorithm to enhance the performance of Galley-McKeown's algortihm. We use the supervised classifier to shortlist the top three senses of each of the words in a text and then use Galley-McKeown's algorithm on this reduced solution space. We show that the Galley-McKeown algorithm boasts of a 24.1% increase in accuracy in WSD performance when shortlisting is carried out by the Naive Bayes Classifier. Since WSD performance is one of the ways of measuring Lexical Chaining performance, the ensemble classifier can be used to construct better lexical chains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The English language, like most other languages, is ambiguous. Among its many ambiguous features - the relation between phonetics and spelling, the rules of its grammar etc. - is a very curious property termed polysemy. Polysemy indicates that a word can have more than one meaning. This particular property is at the heart of most languages - the meaning of a sentence is not merely the sum of the meanings of the individual words. Each word has a number of meanings and can be interpreted differently in different "contexts". Thus, when we try to find meaning in a sentence, we must realize that there is a very complicated interplay between different words in a body of text to produce meaning. The idea is elucidated by the examples below -:

> **Lead** is a bluish-gray, soft, dense metal that has a bright luster when freshly cut.
>
> To **lead** them to victory on that bright saturday afternoon, Rahul, though normally soft-spoken, gave a rousing speech to the ensemble of bluish-gray uniforms standing in a huddle around him.
>
> He placed his pencil softly on the bluish-gray metal table and noticed that its **lead** had been sharpened to a point.

These examples show how easily one can construct sentences, with a large number of words in common, using completely different senses of the word 'lead'. And yet, knowing which sense of the word is activated in the context of each of the sentences is paramount in understanding their meaning. While the task of interpreting which sense of a word is used in a sentence appears to be simple, unfortunately this is not the case. However, it is clear that the ability to computationally determine which sense of each word is used in a sentence will be a giant leap forward in the way we percieve information today. The problem of Word Sense Disambiguation (WSD) is formally motivated and defined in the next two sections.

1.2 WSD - An Introduction

1.2.1 History in Brief

> If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [...] But, if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then, if N is large enough one can unambiguously decide the meaning of the central word. [...] The practical question is: "What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?" [Weaver, 1949]

Much of the early foundations of WSD were laid down in the 1950s. As a matter of fact, it was conceived as a fundamental task in Machine Translation (MT) in the 1940s. Above is the famous memorandum by Weaver who saw the need for WSD in the 1940s and introduced the basis for an approach in WSD which formed the foundation for all early researchers on the subject. At the time, researchers had already understood many of the essential ingredients of WSD, but did not have the

computational power to put these ideas to test. During the 1970s, WSD and MT were attacked by AI methods that aimed at natural language understanding. Work on WSD reached a turning point in the 1980s with the advent of machine readable dictionaries, text corpora and other large scale lexical resources. With the increase in computational power, the 1990s saw large scale implementation of statistical methods in WSD. Periodic competitions and campaigns in WSD became famous, beginning with the Senseval-1 evaluations in 1998.

1.2.2 Motivation

SemEval has evolved from it's Senseval days as a pure WSD competition, into a comprehensive periodic challenge in conquering new frontiers in computational semantic analysis. Now there are a plethora of WSD competitions and discussions held year on year to acheive greater accuracy - there is no doubt that a convincing solution will break new barriers and change the way we percieve, organize and share the wealth of digital knowledge available to us today.

WSD can potentially provide a major breakthrough in the realization of the so-called semantic Web, "an extension of the current Web, in which information is given well-defined meaning, better enabling computers and people to work in co-operation" [Berners-Lee et al. 2001]. Solving the WSD problem could also lead to a major breakthrough in Machine Translation - opening up a world of possiblities not just in natural language translation, but also in human-computer interaction. Exciting concepts like translating natural language into code will become more plausible.

1.2.3 Definitions

> Word sense disambiguation (WSD) is the ability to computationally determine which sense of a word is activated by its use in a particular context. [Roberto Navigli, 2009]

A text **T** can be viewed as an ordered group of sequences of words where punctuation is used as a separator between these sequences.

$$T = \{(w_1, w_2, w_3, ...), (w_{25}, w_{26}, ...), ...\}$$

A word is an entity that has a property called **sense** defined in a dictionary **D**, such that given a word, we can look up the dictionary to get a set of senses for the given word.

$$Senses_D(w_i) = set\ of\ all\ senses\ of\ the\ word\ w_i$$

Then WSD can be formally defined as the task of creating a mapping A, such that

$$A(w_i, T) = The\ correct\ sense\ of\ the\ word\ w_i\ from\ among\ Senses_D(w_i)$$

Clearly,

$$A(w_i, T) \subseteq Senses_D(w_i)$$

WSD can thus be viewed as a classification task. A number of classification tasks are present in the field of Natural Language Processing (NLP) - including part-of-speech tagging and text categorization. However, there is an important difference between these tasks and WSD. In the case of WSD, every distinct word in the text represents a classification task of its own - there are essentially n distinct classification tasks where n is the size of the document.

There is also the question of defining how fine-grained the 'disambiguation' needs to be. Coarse-grained WSD or topic specific WSD are easier problems as we can evolve better rules in these special cases. We assume henceforth that we are only concerned with WSD where the sense distinction is at the level of standard dictionary (WordNet 3.0 for example). The set of target words to disambiguate is another important part of defining the problem - WSD is normally restricted to nouns or content words (nouns, adjectives, verbs, adverbs). In this work, we concentrate on WSD in nouns.

1.2.4 Complexity

WSD is considered to be an AI-complete problem [Mallery 1988], i.e. a problem which is as complex to solve as central problems in Artificial Intelligence. There are a number of factors that contribute to the complexity of WSD. In the case of fine-grained WSD, even human beings assigned the task of disambiguating a group of words may disagree on many cases. This places an upper limit on the accuracy that can be achieved in WSD. The **inter-annotator agreement** (ITA), that is, the percentage of words that are tagged identically by two different human taggers, is widely used as an upper bound in WSD accuracy. In binary or coarse-grained WSD, the ITA is calaculated to be around 90% [Roberto Navigli, 2007]. However, on fine-grained WordNet-style sense inventories, the ITA ranged between 67-80% [Chklovski and Mihalcea 2003, Snyder and Palmer 2004]. The complexity of WSD can be attributed to a number of factors that include (but are not restricted to) the ambiguous nature of the language, the difficulty in problem definition and, most importantly, the **knowledge acquisition bottleneck**.

1.2.5 Knowledge Acquisition Bottleneck

WSD relies heavily on knowledge. The essential framework used by any WSD system can be summarized by the diagram above. Without some form of knowledge, it is impossible to perform WSD. The sources of this knowledge can be unstructured raw text (Gutenberg corpus) parsed corpora with labelled data (Brown Corpus), Sense tagged corpora (SemCor, Senseval) or highly structured sources (searchable dictionaries, semantic networks like WordNet etc.). However, the creation of these knowledge sources is a very expensive and time consuming effort. The task also needs to be repeated again in the case of different scenarios (different languages, new sense inventories etc.). This fundamental obstacle in performing WSD is termed the **Knowledge Acquisition Bottleneck** and is a major hindrance to progress in the field.

> I estimate that a sensetagged corpus of 3200 words is sufficient to build a broad coverage, high accuracy WSD program capable of sig-

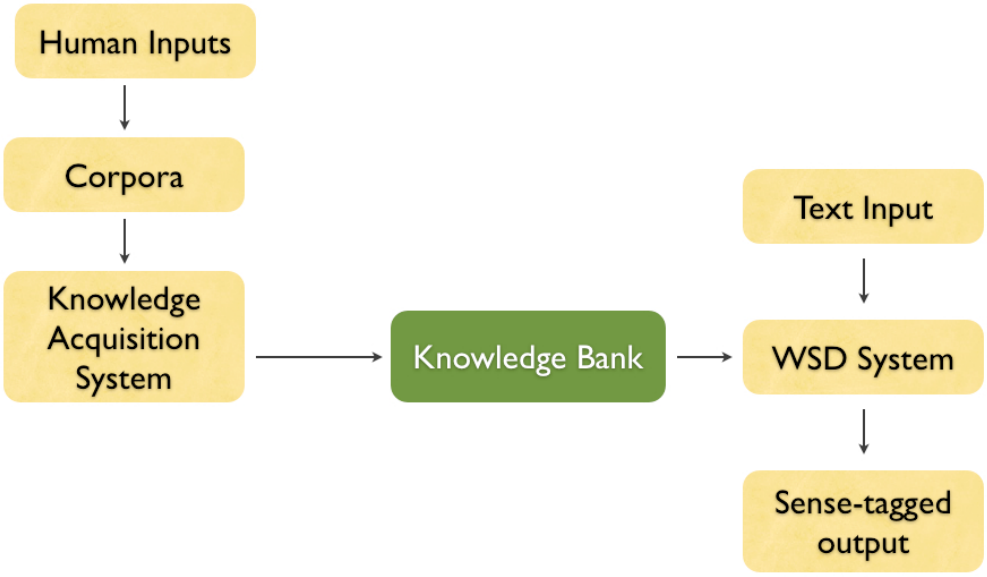

Figure 1.1: The Knowledge Acquisition Bottleneck in WSD

> nificantly outperforming the most-frequent-sense classifier on average over all content words appearing in an arbitrary, unrestricted English text. Assuming an average of 1000 sense-tagged occurrences per word, this will mean a corpus of 3.2 million sense-tagged word occurrences. Assuming human sense tagging throughput at 200 words, or 200,000 word occurrences, per man-year (which is the approximate human tagging throughput of my completed sense-tagging mini-project), such a corpus will require about 16 man-years to construct. [Hwee Tou Ng]

Above, we see the performance of the exemplary LEXAS algorithm [Ng and Lee, 1996] as a function of the number of training examples, which reflects the importance of a large amount of training data in WSD performance. Even the most-frequent-sense classifier shows marginal improvement with increase in training data. SemCor 3.0, one of the largest corpora currently available contains less than 200,000 sense tagged word occurences - only 6.25% of Hwee Tou Ng's estimated 3.2 million. Thus, the sparseness of the data currently available poses a serious problem in achieving greater WSD accuracy.

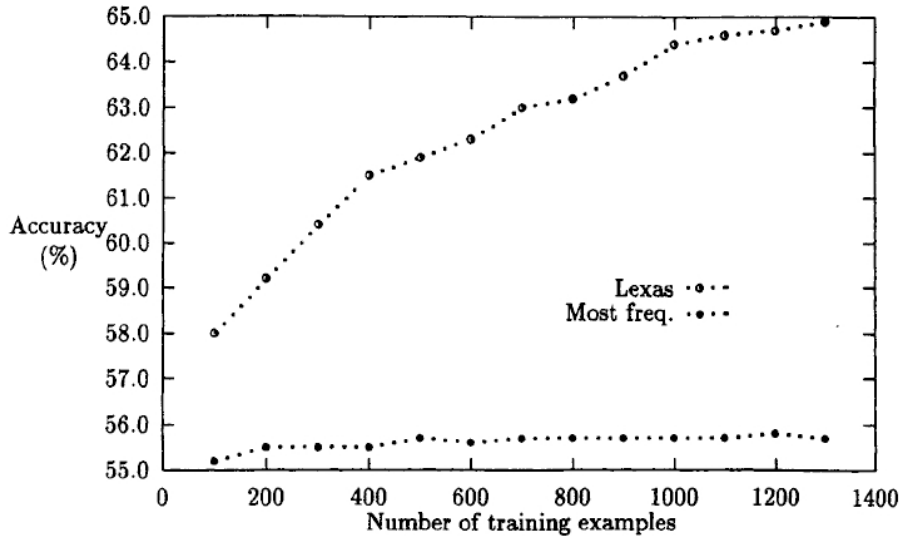

Figure 1.2: Effect of # of training examples on WSD accuracy

1.3 WSD : Task Description

As earlier mentioned, WSD can be viewed as a classification task, but with a major difference. In WSD, we actually have **n** classification tasks where **n** is the size of the lexicon. This is because every word has a unique set of word senses to be classified into. In conventional classification tasks, there is a predefined set of classes, rather than a set of classes that varies depending on the word to be classified. Thus, every WSD task requires two main components - selection of word senses, i.e. classes for the classification tasks and the selection of a classifcation method. In order to facilitate the task, we also require external knowledge sources. Another important element of the task is the method chosen to represent the context. These four defining elements of the WSD task are discussed in detail in this section.

1.3.1 Sense Inventories

A word sense can be defined as a widely accepted meaning of a word in a language. In linguistics, there are a number of definitions for words which have the

Noun

- S: (n) **lead** (an advantage held by a competitor in a race) *"he took the lead at the last turn"*
- S: (n) **lead**, Pb, atomic number 82 (a soft heavy toxic malleable metallic element; bluish white when freshly cut but tarnishes readily to dull grey) *"the children were playing with lead soldiers"*
- S: (n) **lead**, track, trail (evidence pointing to a possible solution) *"the police are following a promising lead"; "the trail led straight to the perpetrator"*
- S: (n) **lead** (a position of being the initiator of something and an example that others will follow (especially in the phrase `take the lead')) *"he takes the lead in any group"; "we were just waiting for someone to take the lead"; "they didn't follow our lead"*
- S: (n) **lead** (the angle between the direction a gun is aimed and the position of a moving target (correcting for the flight time of the missile))
- S: (n) **lead**, lead-in, lede (the introductory section of a story) *"it was an amusing lead-in to a very serious matter"*
- S: (n) **lead** ((sports) the score by which a team or individual is winning)
- S: (n) star, principal, **lead** (an actor who plays a principal role)
- S: (n) **lead** ((baseball) the position taken by a base runner preparing to advance to the next base) *"he took a long lead off first"*
- S: (n) tip, **lead**, steer, confidential information, wind, hint (an indication of potential opportunity) *"he got a tip on the stock market"; "a good lead for a job"*
- S: (n) **lead**, lead story (a news story of major importance)
- S: (n) spark advance, **lead** (the timing of ignition relative to the position of the piston in an internal-combustion engine)
- S: (n) leash, tether, **lead** (restraint consisting of a rope (or light chain) used to restrain an animal)
- S: (n) **lead**, leading (thin strip of metal used to separate lines of type in printing)
- S: (n) **lead**, pencil lead (mixture of graphite with clay in different degrees of hardness; the marking substance in a pencil)
- S: (n) jumper cable, jumper lead, **lead**, booster cable (a jumper that consists of a short piece of wire) *"it was a tangle of jumper cables and clip leads"*
- S: (n) **lead** (the playing of a card to start a trick in bridge) *"the lead was in the dummy"*

Figure 1.3: WordNet's sense inventory for **lead**

same spelling, but different meanings. There are fine distinctions between polysemes, homonyms etc. based on etymology and semantics. However we choose to use a simple set of definitions in this work - the same nomenclature is widely used in literature.

1. A word is defined by a combination of spelling and part of speech. Every word consists of a string of characters and a part of speech tag.

2. Each word has a number of **senses**, which are semantic classes that words (of the same spelling and part of speech) are divided into.

Let us take the example of **lead**. The verb 'lead' and the noun 'lead' are considered to be **different** words. However, pencil **lead** and a **lead** in a detective case are considered to be two **senses** of the word lead (noun).

A sense inventory is defined as an enumeration of the various senses of the words to be disambiguated. Now we are posed with the problem of defining the various senses of different words. Enumerating the senses of a word is a complex task as word senses are seldom easy to discretize. There is also the question of the granularity in the discretization of various senses of a word - whether we use coarse grained or fine grained sense inventories? The exact requirements of the sense inventory will vary depending on the application being served by the WSD task. However, throughout this task, we will choose to use WordNet 3.0 as our sense inventory. WordNet, which is explained in detail in detail in a later section, not only serves as a sense inventory but also as a powerful knowledge source for disambiguation. The WordNet sense inventory is fine grained, with very subtle sense distinctions. This makes our disambiguation task even harder. Figure 2.1 shows an example from WordNet to illustrate the degree of subtlety in the distinctions between various senses of the word lead.

There are some issues, however, with the enumerative approach. Is the idea to 'split' the senses (fine-grained) or 'lump' the senses (coarse-grained)? Natural languages are living languages - words continually acquire new meanings with time when used in varying the contexts. How then can words be assigned a pre-defined set of senses? To answer these questions, an alternate **generative** approach has

been suggested [Pustejovsky 1991, 1995]. However, it can also be argued that it is extremely difficult to measure the performance of a WSD system if we don't use an enumerative approach. Throughout the remainder of this work, WordNet is used as the sense inventory for disambiguation.

1.3.2 Representation of the context

The context contains all the information needed to decide the sense of the words in a text. It is fair to say that words only have meaning in a context. However, text in a natural language is structurally ambiguous. For an automatic classifier to be able to interpret this unstructured information source, it is normally preprocessed into a more suitable format.

1.3.2.1 Preprocessing

The preprocessing task normally includes some or all of the following steps-:

1. Tokenization - Splitting the text into a sequence of tokens (usually words or small groups of words) to capture the basic semantic units of the text.
2. Part of Speech tagging
3. Lemmatization - to reduce morphological variants to their base form
4. Elimination of stop words - words like "and", "a" are eliminated if they do not provide meaningful contextual information for disambiguation.
5. Other methods - Parsing, chunking etc. [Refer to Navigli 2009 for details]

1.3.2.2 Feature Selection

After pre-processing, a set of features are carefully chosen to represent the context. The features used can be broadly divided into the following categories -:

1. Local features - These features represent information related to the immediate surroundings of a word usage - the words, part of speech tags etc. in the local neighbourhood of the word.

2. Topical features - These features aim to capture the general context of the word - for example, the topic or domain of the text or discourse in question.

3. Syntactic and Semantic features that aim to exploit information related to the structure of the text and meaning of past occurences of the word etc.

Normally, the word and its feature set are written as 'feature vectors'. **(word, pos, position in sentence)** is a simple example of a feature vector. A classifier normally acts on a set of feature vectors of the words forming the context to predict the sense of the target word.

The 'size' of the context is also a very important. There are numerous context sizes used, that can be described by n-gram models (n words including the target word) - unigram models (n=1), bigram models (n = 2) etc. We can also have models which use a context size of one sentence or one paragraph.

1.3.3 Classification Methods

The classification methods used in WSD can be broadly divided into -:

1. Supervised methods - These WSD methods use training data (data labelled with a number of feature tags as well as the correct sense tags) to train the classifiers using machine learning methods. These classifiers then apply what they have "learned" from the training data to classify texts.

2. Unsupervised methods - These WSD methods only use unlabelled corpora and do not require manually tagged corpora to perform the classification task.

Further, we can also distinguish between "knowledge-based" and "corpus-based" methods. The former rely on using machine-readable dictionaries, thesauri, WordNet etc. while the latter do not make use of these knowledge sources to carry out

the WSD task. We can also consider semi-supervised methods, which use a combination of supervised and unsupervised methods.

Note on Semisupervised and ensemble methods

Ensemble methods are combinations of classifiers that are employed together to improve the overall classification accuracy. Ensemble methods attempt to use independent features to get different or independent "views" of the text. Some of these methods are also termed "Multiview methods" for this precise reason. These ensemble methods are becoming increasingly popular as they allow one to overcome the weaknesses of individual techniques. Single classifers are combined using a number of methods, including majority voting, adaptive boosting, rank-based combinations and probability mixtures [for details, refer to Navigli 2009].

1.4 Lexical Chaining

In a typical discourse, words that are related semantically come together to collectively convey an idea. Consider these examples the following examples

> The apple, formally known as Malus Domestica, grows on a tree that is small and deciduous. The fruit matures in autumn and is typically 5 to 9 centimetres (2.0 to 3.5 in) in diameter. Five carpels arranged in a five-point star characterize the Malus Domestica.
>
> Lead is a bluish-gray, soft, dense metal that has a bright luster when freshly cut.

The words Malus Domestica, apple and fruit are semantically related. Apple and Malus Domestica are synonyms, while fruit is the general class that apples belong to. All three words are used to speak about the same idea - an apple. In natural languages, using these related words to convey an idea is very common and widely used. If one looks strictly at the meaning of each word then this may be deemed inaccurate – all fruits are not apples. However, given the context, "the fruit" now

refers to an apple - the author relies on relaxed connotations to cast an idea into words. Thus, most ideas in a text are presented via groups of related words that collectively point towards a central meaning.

The occurrence of these related words, often in a localized region in a text, is loosely said to form a context. A context is instrumental in interpretating the meaning of a text. This is especially true when the text contains polysemous words, metaphors, sarcasm etc. A simple example is the interpretation of the word 'apple' in the example above. Clearly, we are talking about the fruit apple, and not the company "Apple" that makes iPhones and MacBooks. Words like 'fruit', 'tree', 'Malus Domestica' characterize the context and allow us to interpret the meaning of the word apple. The significance of contexts to understanding of text make their automatic identification highly desirable. Methods of automatically identifying contexts intimately relate to the precise definition of a context.

"Context" has now been vaguely defined as a related set of words that imbue text with meaning. The task at hand often makes it necessary to further qualify this definition. Such definitions range from only considering words within a fixed distance from each other as forming a context, to using sophisticated mathematical models of word distribution in a text to describe it. Our work concerns itself with a family of algorithms called lexical chaining that are used to identify contexts.

A lexical chaining algorithm identifies sets of words called lexical chains as contexts, with words in the set adhering to certain constraints regarding relationships to each other and their physical location in text. The group of words {apple, Malus Domestica, fruit} from the example above, is an example of a lexical chain that a chaining algorithm might discover.

One of the most important features of a lexical chaining algorithm - one that we aim to exploit in our work - is that the formation of lexical chains automatically leads to WSD. By relating the words {apple, Malus Domestica, fruit}, we fix the meaning of the word apple. Similarly, by forming the chain {Lead, metal} we fix the meaning of the word lead. On the other hand, if we were able to form a different chain, containing words like {lead, pencil...} we see that we have fixed a different meaning for the word lead. Thus, it is evident that Lexical Chaining algorithms are powerful tools in WSD. However, existing Lexical Chaining algorithms

have poor WSD performance. In our work, we aim to boost the performance of Lexical Chaining algorithms in performing WSD by using them in tandem with a supervised classifier to produce a state-of-the-art WSD system.

1.4.1 Properties of Lexical Chains

[Adapted from Supervised Lexical Chaining, Abhishek Ghose 2011]

The idea of lexical chaining was first introduced by Morris and Hirst [1991]. They also proposed the first lexical chaining algorithm. Various other chaining algorithms have been evolved since. Chaining algorithms may differ significantly in their implementations. However, they are characterised by certain general principles regarding how they perform chaining. Below is a list of these properties.

1. **External knowledge source** : A chaining algorithm needs to know the semantic relationships between words in order to group them appropriately. Referring to the examples above, a chaining algorithm must have a way to know that apple is a specific kind of fruit or a lead is a specific kind of metal. Such knowledge is provided by an external knowledge source that the algorithm consults. The first chaining algorithm [Morris and Hirst, 1991] used the Rogets Thesaurus as its knowledge source. Most modern algorithms use a dictionary like Wordnet (read section 1.5).

2. **"Closeness" of relationships that may be allowed** : Two words may be transitively related. For example, a dog is a canine which is a type of mammal. And mammals are a type of animal. Thus a dog is a type of animal. A chaining algorithm must be equipped to discover such relationships. However, it must also be wary of following too long a transitive path. Long paths tend to relate words incorrectly. Morris and Hirst [1991] provide an example of the risks of allowing unlimited transitivity: the transitive chain {cow, sheep, wool, scarf, boots, hat, snow}, where every two consecutive words are related, misleads a chaining algorithm to assume cow and snow are related and miss potential context boundaries. Chaining algorithms typically impose a restriction on the lengths of transitive relations.

3. **"Type" of relationships that may be allowed** : Lexical chaining algorithms specify what kind of relationships between words they consider valid. For example, a part-of/whole-of relationship, in which one entity is a part of the other entity, like the relationship between finger nail and finger, is considered to be a valid relationship in the algorithm proposed in St-Onge [1995], but not in the one proposed in Galley and McKeown [2003].

4. **Locality of words** : Chains also assume that words may be indicative of a context only within a certain local region in text. This is not a strict rule as certain words are considered related even if they occur far apart in text, but such physical nearness is preferred. The assumption stems from the observation that individual ideas are often described in highly localized regions.

Lexical Chaining and Word Sense Disambiguation (WSD)

The relationship between a polysemous word and other words depends on the meaning it represents in a particular occurrence. If the word dog occurs as a part of the term hot dog in a text, it clearly is not related to canine. Hence, chaining algorithms determine the intended meaning of polysemous words as a necessary side-effect – since without this knowledge it is impossible to semantically group words. Thus, Lexical Chaining algorithms instrinsically perform WSD when they form lexical chains.

The above properties broadly capture the essence of chaining algorithms. It may be noted that most lexical chaining algorithms discover relationships only between nouns. Henceforth, when we mention words in the context of chaining they may be assumed to be nouns unless otherwise stated.

1.5 Galley-McKeown algorithm

The Galley-McKeown algorithm is a lexical chaining algorithm. It uses WordNet as a knowledge source to build chains of candidate nouns that are semantically related. A weight is assigned to each semantic relation. The algorithm can be

decomposed into 3 steps, of which the first two are of vital importance to this work. The steps are as follows -:

1. Build a representation of all possible interpretations of the text (every possible combination of word senses of the given words, with a restriction of one sense per word per discourse).
2. Disambiguate the word senses
3. Build the lexical chains

The Galley-McKeown algorithm has two distinct steps where it first performs WSD and then proceeds to build lexical chains. This has two advantages. First, the algorithm performs better at both disambiguation and chaining than conventional algorithms like St. Onges, which perform both disambiguation and chaining in a single-step parallel fashion. Second, since the algorithm has a separate disambiguation step, it can easily be used in WSD. The algorithm provides a lexical resource (WordNet) based WSD mechanism, which we employ in our ensemble classifier.

Disambiguation Algorithm

First, the whole text is processed and all senses of every distinct noun that appears in the text is added to our "work table". Thus, by selecting one sense of each word, we can create every possible semantic permutation of the text. Note that there is only one entry per distinct word - the algorithm implicitly makes the **one sense per discourse** assumption. Once this table has been created, we move to the second step, which involves checking for semantic relations. We create a link between every pair of senses in our work table that has one of the following semantic relations - synonymy, hypernymy, hyponymy or sibling pairs. Sibling pairs are hyponyms of hypernyms (ex. cat and tiger). The weight of the link is based, not just on the type of semantic relation, but also on the distance between the words being considered in the text. The exact choice of weights is based on

Semantic relation	1 sent.	3 sent.	1 par.	other
synonym	1	1	0.5	0.5
hypernym/hyponym	1	0.5	0.5	0.3
sibling	1	0.3	0.2	0

Table 1.1: Assigning weights in Galley's Algorithm

table 1.1. Thus, a disambiguation graph is built, with connections between all semantically related senses of all nouns in the text.

Next is the actual disambiguation step. All the weights of the links attached to each sense of a word are summed up. Now there is a weight for every sense of each word. The sense of each word with the maximum weight is chosen as the correct sense. In case of ties, the word with the higher rank (more frequent word sense according to WordNet) is used. Note that the disambiguation of each distinct word is done in a single step unlike St-Onge's algorithm where each word occurence is disambiguated separately.

The third step involves dropping all links in the graph that contain "incorrect senses" of the words to obtain the Lexical Chains. However, since we are only concerned with determining the most probable sense of each word, we stop at step 2. For a detailed explanation of the algorithm, please refer to ImprovingWord Sense Disambiguation in Lexical Chaining, Michel Galley and Kathleen McKeown [2003].

1.6 WordNet

WordNet is a lexical resource for English words. It's development began in 1985 at Princeton's Cognitive Sciences Lab. As of WordNet 3.0, the database contained 155,287 words organized in 117,659 synsets for a total of 206,941 word-sense pairs.

> WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked

> by means of conceptual-semantic and lexical relations. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. [WordNet Website]

The difference between WordNet and a thesaurus, although not immediately apparent, is critical to WSD. WordNet relates groups of **word senses** rather than words. All relations in the Net are between synsets, i.e. specific senses. Thus, WordNet captures the actual semantic information stored in word sense.

1.6.1 Structure

The main relation used in WordNet is synonymy. Synonyms are grouped into unordered sets (synsets). Each synset is linked to other synsets by means of "conceptual relations." Additionally, a synset contains a brief definition ("gloss") and, in most cases, one or more short sentences illustrating the use of the synset members. Each form-meaning pair in WordNet is unique.

> glosses : physics.n.01 - the science of matter and energy and their interactions ; physics.n.02 - the physical properties, phenomena, and laws of something
>
> Synets('argument') - 'argument.n.01', 'controversy.n.01', 'argument.n.03', 'argument.n.04', 'argument.n.05', 'argument.n.06', 'argumentation.n.02'

1.6.2 Relations

Hypernymy and Hyponymy

The super-subordinate relation is captured here. It links more general synsets to increasingly specific ones. Ex: Canine –> Dog –> Puppy. All noun hierarchies ultimately go up the root node {entity}. The hyponymy relation is transitive: if an armchair is a kind of chair, and if a chair is a kind of furniture, then an armchair is a kind of furniture.

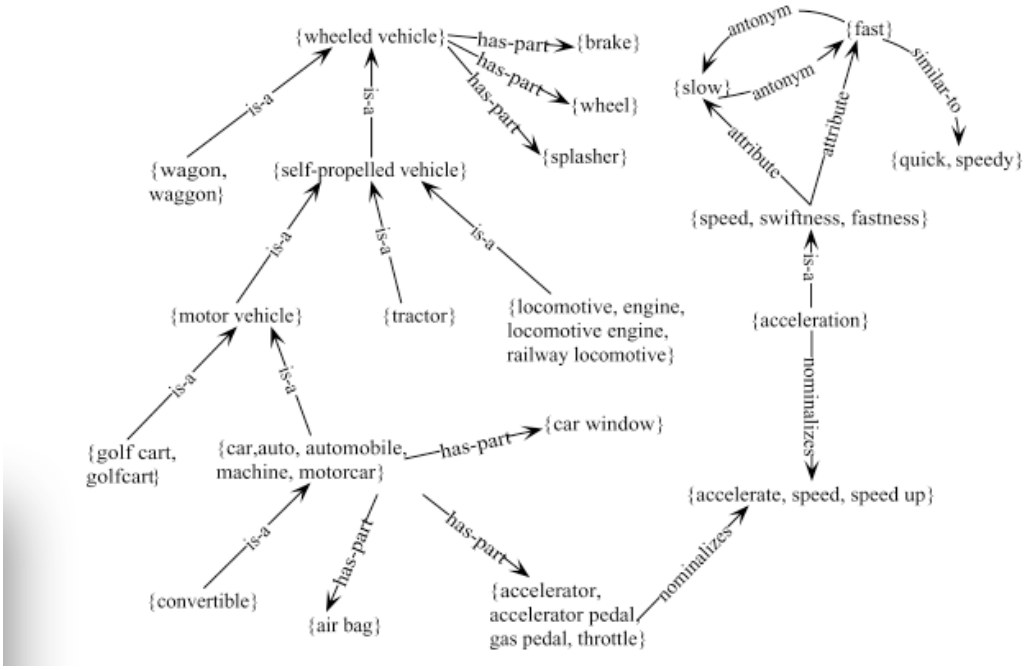

Figure 1.4: An excerpt from WordNet [Navigli 2009]

Meronymy

This is the part-whole relation holds between synsets like {chair} and {back, backrest}, {seat} and {leg}. Parts are inherited from their superordinates: if a chair has legs, then an armchair has legs as well. Parts are not inherited "upward" as they may be characteristic only of specific kinds of things rather than the class as a whole: chairs and kinds of chairs have legs, but not all kinds of furniture have legs.

Other Parts of Speech

Information regarding verbs are also organized in similar heirarchies, but only noun WSD related information is provided here. The majority of the WordNet's relations connect words from the same part of speech (POS). Thus, WordNet really consists of four sub-nets, one each for nouns, verbs, adjectives and adverbs, with few cross-POS relations.

1.7 SemCor

SemCor [Miller et al. 1993] is a subset of the Brown Corpus, whose content words have been manually annotated with part-of-speech tags, lemmas and word senses from theWordNet inventory. SemCor is composed of 352 texts. SemCor contains roughly 234,000 semantically annotated words and is the largest sense-tagged corpus available at present. Therefore, it is widely used in training supervised classifiers. Here is an excerpt from the corpus. This contains one sense tagged sentence.

```
<s snum=1>
<wf cmd=ignore pos=DT>The</wf> <wf cmd=done rdf=group
    pos=NNP lemma=group wnsn=1 lexsn=1:03:00:: pn=group
    >Fulton_County_Grand_Jury</wf>
<wf cmd=done pos=VB lemma=say wnsn=1 lexsn=2:32:00::>
    said</wf>
<wf cmd=done pos=NN lemma=friday wnsn=1 lexsn
    =1:28:00::>Friday</wf>
<wf cmd=ignore pos=DT>an</wf>
<wf cmd=done pos=NN lemma=investigation wnsn=1 lexsn
    =1:09:00::>investigation</wf>
<wf cmd=ignore pos=IN>of</wf>
<wf cmd=done pos=NN lemma=atlanta wnsn=1 lexsn
    =1:15:00::>Atlanta</wf>
<wf cmd=ignore pos=POS>'s</wf>
<wf cmd=done pos=JJ lemma=recent wnsn=2 lexsn=5:00:00:
    past:00>recent</wf>
<wf cmd=done pos=NN lemma=primary_election wnsn=1 lexsn
    =1:04:00::>primary_election</wf>
<wf cmd=done pos=VB lemma=produce wnsn=4 lexsn
    =2:39:01::>produced</wf>
<punc>``</punc>
<wf cmd=ignore pos=DT>no</wf>
<wf cmd=done pos=NN lemma=evidence wnsn=1 lexsn
    =1:09:00::>evidence</wf>
<punc>''</punc>
<wf cmd=ignore pos=IN>that</wf>
<wf cmd=ignore pos=DT>any</wf>
```

```
<wf cmd=done pos=NN lemma=irregularity wnsn=1 lexsn
    =1:04:00::>irregularities </wf>
<wf cmd=done pos=VB lemma=take_place wnsn=1 lexsn
    =2:30:00::>took_place </wf>
<punc>.</punc>
</s>
```

The original SemCor was annotated according to WordNet 1.5. However, we use a more recent mapping of SemCor - the SemCor 3.0 data set which maps to the WordNet 3.0 inventory.

1.8 Related Work

As discussed before, WSD has been widely studied and a number of WSD techniques have evolved over time. A brief overview of popular ideas in WSD and lexical chaining is provided in this section. First we take a brief look at the various supervised and unsupervised classifiers.

This diagram [Navigli, 2009] gives us a good idea of the various approaches to WSD.

1.8.1 Supervised WSD

A number of machine learning techniques are applied to WSD and they all fall under the umbrella of supervised classifiers. Among them, Neural Networks, Support Vector Machines (SVMs) and k-nearest neighbours are popular examples. However, neural networks has major drawbacks - difficulty in interpreting the results, the need for a large quantity of training data, and the tuning of parameters such as thresholds, decay etc.

In k-nearest neighbours (kNN), training data is retained in the memory as points in a feature space, and test data is progressively added to this space as it is classified. It is a high performing method in WSD [Ng 1997; Daelemans et al. 1999]. kNN belongs to a class of methods that are collectively called “examplar-based” or

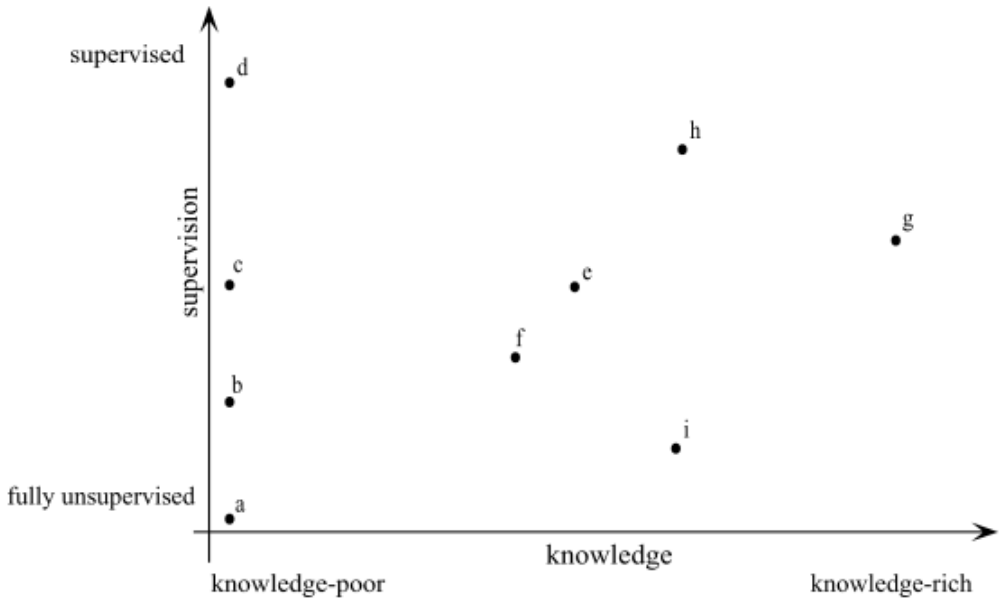

Figure 1.5: The various approaches to WSD

"instance-based" methods as the classification model is built from examples. At present, exemplar-based methods achieve state-of-the-art performance in WSD [Escudero et al. 2000b; Ng and Lee 1996].

The essential idea in SVMs is to construct a linear hyperplane from the training set, that separates positive examples from negative examples. SVMs attempt to simultaneously maximize the geometric margin between positive and negative examples, while minimizing the empirical classification error (as illustrated in the figure). SVMs are binary classifiers and thus need to be used in a multinomial setting to perform WSD. SVM performs well compared to many other supervised methods in performing WSD.

1.8.2 Semisupervised and Unsupervised WSD

Bootstrapping is a popular form of semi-supervised WSD. Yarowsky's [1995] boostrapping method is a self-training approach. There are also co-training approaches, where 2 classifiers alternately and iteratively tag data and then learn from the tagged data. Yarowsky's method relies on two heuristics:

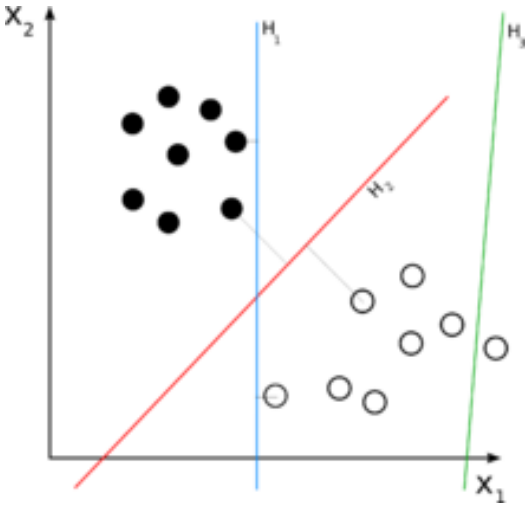

Figure 1.6: Support Vector Machines

1. One sense per collocation [Yarowsky 1993]: nearby words strongly and consistently contribute to determine the sense of a word, based on their relative distance, order, and syntactic relationship;

2. One sense per discourse [Gale et al. 1992c]: a word is consistently referred with the same sense within any given discourse or document.

The approach exploits decision lists to classify instances of target words. The decision lists are then iteratively trained on the freshly tagged target words to gather new information to disambiguate further words. On small-scale binary data sets very high accuracy can be achieved using Yarowsky's method.

Unsupervised WSD methods have the potential to overcome the Knowledge Acquisition Bottleneck and therefore they have been studied closely for decades now. These approaches are based on concepts like - the same sense of a word will appear in similar contexts. 'Pure' unsupervised disambiguation also means that the method makes use of no dictionaries or thesauri. This can pose to be a serious disadvantage however. We consider systems that also use dictionaries. Pure WSD is considered to be Word Sense **Discrimination**, where similar senses are grouped together. Common techniques include methods like context clustering, word clustering and concurrence graphs.

Knowledge rich methods take us towards lexical chaining algorithms.

1.8.3 Lexical Chaining

A first computational model of lexical chains was introduced by Hirst and St-Onge [1998]. However, this method suffered from poor WSD performance since the words are immediately disambiguated in a greedy fashion leading to inaccuracies. Barzilay and Elhadad [1997] dealt with the inaccuracy of the original approach by keeping all possible interpretations until all the words to be chained have been considered. While this improved WSD performance significantly, the computational inefficiency of the huge number of combinations still remained until Silber and McCoy (2003) proposed an efficient linear time algorithm. Finally, Galley and McKeown proposed a modification from the Silber and McCoy algorithm that improved accuracy even further. Galley's algorithm has been discussed in detail in sections 1.5 and 2.3.

1.9 Contribution of the thesis

1.9.1 Objectives and motivation

This work attempts to use an ensemble classifier constructed using Galley-McKeown's algorithm (a knowledge-based unsupervised method) along with a simple Naive Bayes based supervised classifier to improve performance and achieve state-of-the-art disambiguation. The method also aims to improve the efficiency of Galley-McKeown's algorithm. By building this classifier, we aim to strengthen the case for ensemble or multi-view classifiers as the way forward in WSD. We also aim to show how Galley's lexical chaining algortihm and lexical chaining algorithms in general can be made both more accurate and more efficient by the use of these ensemble methods.

1.9.2 Overview of our work

We construct an ensemble classifier and measure its performance with respect to the original Galley-McKeown algorithm as well as state-of-the-art WSD systems. The algorithm uses a simple probability-based supervised classifier to rank the top 3 most likely senses of each word in a text after which we use Galley's algorithm to choose the correct word sense from the reduced solution space. The classifier shows significant improvements in terms of both speed and accuracy. We test the performance of the classifier on the Senseval-2 and Senseval-3 competition test data to show performance on par with state-of-the-art fine-grained WSD systems.

1.10 Organization of the thesis

The purpose of this chapter was to present an overview of WSD, motivate the problem, discuss the ideas employed in performing WSD and lay the framework for understanding the WSD task that we are going to present in the next chapter. We also provide the objective of this thesis.

Chapter 2 - Ensemble Classifier Model - This chapter discusses our WSD system in detail and the ideas used in our model.

Chapter 3 - Algorithms, Implementation and Performance - The fourth chapter provides details of the algorithms and their performance with respect to various benchmarks.

Chapter 4 - Conclusions and Future Work - In the final chapter, we discuss the conclusion of our work, possible improvements in our methods, insights gained from this implementation and avenues for future work.

Chapter 2

Ensemble Classifier Model

Our ensemble classifier carries out WSD using a supervised classifier (Naive Bayes classifier) and an unsupervised classifier (the Galley-McKeown lexical chaining algorithm). In figure 2.1, we use a block diagram to illustrate the basic functioning of the classifier.

A major problem faced by supervised classifiers in WSD is the Knowledge Acquisition Bottleneck. Thus, the sparseness of the training data is the major concern when constructing a supervised classifier. On the other hand, lexical chaining algorithms are unsupervised, knowledge rich algorithms that rely on WordNet, which can be considered a fairly complete knowledge resource. However, existing lexical chaining algorithms suffer from poor accuracy in WSD performance. Ideally, the Naive Bayes classifier should be able to shortlist a set of 'most probable' senses from which the Galley-McKeown algorithm then builds chains, thus improving WSD performance of the Galley-McKeown algorithm and therefore helping in building better lexical chains.

A feature of the Naive Bayes Classifier we have carefully selected is the choice of context. The supervised classifier works on a context window of one sentence. Thus, the Naive Bayes classifier bases its predictions on local contextual data. On the other hand, Galley's algorithm constructs lexical chains across an entire document. Thus, one classifier exploits local contextual data, while the other classifier constructs document wide chains, thus gathering topical information to

perform WSD.

2.1 Classifier Model

Our ensemble classifier does not use conventional methods to combine the results of the two classifiers. Rather, it uses the Naive Bayes classifier to shortlist the top 3 most probable senses of each word, and then uses Galley-McKeown's algorithm to choose the best sense from amongst these shortlisted senses. In cases where the probabilities assigned to the second or third shortlisted senses are very low, these entries are eliminated from the shortlist. Galley-McKeown's algorithm needs to form chains from a shortlisted set of senses.

2.1.1 Efficiency

Galley-McKeown's algorithm is computationally intensive as it searches through a solution space containing every sense of every word present in a document. The algorithm necessarily checks the semantic relationships between every pair of senses in the document. By reducing the number of senses under consideration to a maximum of 3 senses per word, the method greatly enhances the efficiency of Galley-McKeown's algorithm.

2.1.2 Multiple Views - Accuracy

The Naive Bayes classifier provides the top 3 most probable senses based on past experience - it ranks senses of a word based on the probability of their occurence, given the other words in the same sentence. The remaining words in the sentence are considered, irrespective of whether they are nouns, verbs, adverbs etc. Thus, this classifier captures relationships like {eat(v), food(n)} or {greenery, tree} which Galley-McKeown's algorithm will not recognize (as it only looks for hypernymy/hyponymy and sibling type relationships and it only considers noun-noun relations). Since the Naive Bayes classifier is based on probability of oc-

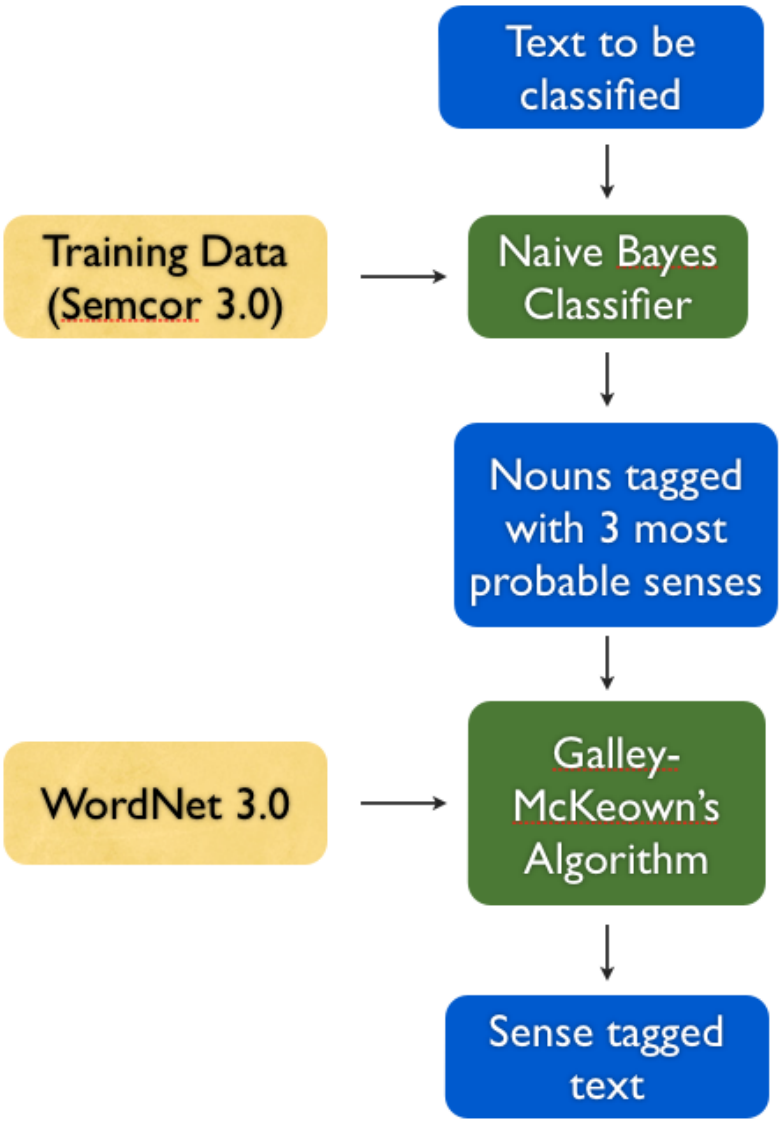

Figure 2.1: Block Diagram of the classifier model

curence, and is trained on the SemCor dataset, one can atleast be confident that senses that occur together in a sentence regularly will be easily disambiguated.

On the other hand, Galley-McKeown's algorithm captures relationships between words across the file that are lexically related as hypernyms, hyponyms or siblings. Thus, the algorithm captures semantic relationships between nouns across the entire file. Thus, if the words "lead" and "element" appear in the same document a number of times, but never in the same sentence, the lexical chaining algorithm will choose lead|Pb as the correct sense.

Thus, he two classifiers can complement eachother effectively by compensating for eachother's weaknesses to increase the overall WSD performance of Galley-McKeown's Algorithm.

2.2 Naive Bayes Classifier

The Naive Bayes classifier is a simple probabilistic classifier based on the application of Bayes' theorem. It relies on the calculation of conditional probabilities of each sense S_i of a word w given the features f_j in the context. The sense $\hat{S}$ which maximizes the following formula is chosen as the most appropriate sense in context:

$$\hat{S} = argmax_{S_i \subseteq Senses_D(w)} \quad P(S_i|f_1, f_2, f_3, f_4 ... f_n)$$

$$\hat{S} = argmax_{S_i \subseteq Senses_D(w)} \quad \frac{P(f_1, f_2, f_3, f_4 ... f_n|S_i).P(S_i)}{P(f_1, f_2, f_3, f_4 ... f_n)}$$

Applying the 'Naive' Bayes assumption (assuming independence of features)

$$\hat{S} = argmax_{S_i \subseteq Senses_D(w)} \quad P(S_i).\prod_1^n P(f_j|S_i)$$

2.2.1 Feature Selection

We consider feature vectors of the following kind while constructing our Naive Bayes classifier.

> **(word1 + word1_pos, word1_sense, co-word1 + co-word1_pos, co-word2 + co-word2_pos, co-word3, co-word3_pos...)**

1. The **word1** in the above vector is a noun from the training data. Since our classifier is designed to disambiguate nouns only, we only consider those words which have the part-of-speech-tags NN or NNS or NP or NPS.
2. The **word_pos** field records the part-of-speech tag of word1.
3. **word1_sense** contains the word sense number (based on WordNet 3.0) of word1.
4. **co-word** indicates a contextual word. In this case we limit our classifier's context to one sentence. Thus, co-word is any word appearing in the same sentence as word1.
5. **co-word_pos** is the part of speech tag of co-word. The co-word is not restricted to nouns only and can be any word in the same sentence as word1.

The feature set is not "ordered". Thus, we use a bag of words approach where we choose all the words in a sentence, together with their pos tags as the feature set. Thus, each f_j in the notation used above, denotes a co-word (and its part of speech tag) from a sentence containing w.

2.2.2 Training

Thus, given a word w, we need to find $\hat{S}$ such that

$$\hat{S} = argmax_{S_i \subseteq Senses_D(w)} \quad P(S_i).\prod_1^n P(f_j|S_i)$$

word1	word1_pos	word1_sense	co-word	co-word_pos	count	probability
lead	NN	2	element	NN	15	15/80
lead	NN	1	element	NN	1	1/80

Table 2.1: Training of the Naive Bayes Classifier

To do so, we need the individual probabilities which are calculated as follows.

$P(S_i)$ simply denotes the probability of the sense S = S_i of word w occuring.

$$P(S_i) = \frac{number\ of\ occurences\ of\ sense\ w(S_i)}{number\ of\ occurences\ of\ word\ w}$$

The probabilities needed are easily obtained from the training data (SemCor 3.0).

To calculate the conditional probabilities we need to gather the following information from the training data

$$P(f_j|S_i) = \frac{number\ of\ occurences\ of\ S_i\ and\ f_j\ in\ the\ same\ sentence}{number\ of\ occurences\ of\ w\ and\ f_j\ in\ the\ same\ sentence}$$

This is easily done by constructing a table of counts and then calculating the required probabilities. An example is shown to illustrate the process in table 3.1

Thus, if metal and lead occur in the same sentence, then the sense of lead is more likely to be 2 (lead as in Pb, the metal) than 1 (lead as in advantage held by a competitor in a race). This is then used to disambiguate the sense of the word lead which might appear in the text to be classified.

2.2.3 Classification

The text to be classified (henceforth referred to as the "test data") is then analyzed by the classifier. We explain the classification process by considering a single target word. This same process is repeated for all target words (nouns) in the test data. Consider the target word **lead** in the sentence-:

> Ronaldo was in his element and his brilliant goal took Brazil into the **lead**.

First the stop words are dropped. This leaves us with

> sentence = Ronaldo element brilliant goal took Brazil **lead**

As lead is the target word it will be disambiguated as follows -:

```
for sense in senses_of_lead:
        for co-word in sentence:
                Score(sense) = sum[Probability(lead.sense)*log{
                    Probability(co-word|lead.sense)}]
```

It is a common practice to use the log sum instead of the product of the conditional probabilities. The 3 senses of lead with the highest scores will be chosen as the 3 most probable senses of the target word lead. Thus, every noun in the test data will be disambiguated by the target classifier and the top 3 senses of the word will be passed on to the next classifier.

2.2.4 Smoothing

Given the sparse training data available, a number of token-feature pairs will be seen for the first time in the test data. Since they will have zero counts they will reduce the overall probabilities to zero as we have a product of probabilities. Thus, to ensure there is no "problem of zeroes", we need to smooth the data. There are a number of smoothing techniques available. In this case, we use additive smoothing (also called Laplace Smoothing).

Additive Smoothing

In additive smoothing, all the conditional probabilities are smoothed by adding a very small constant α to each of the counts and dividing correspondingly by N*α where N is the total number of counts to which α was added. Thus, the smoothed probabilities obtained are

$$P(f_j|S_i) = \frac{number\ of\ occurences\ of\ S_i\ and\ f_j\ in\ the\ same\ sentence\ + \alpha}{number\ of\ occurences\ of\ w\ and\ f_j\ in\ the\ same\ sentence\ + N\alpha}$$

These smoothed probabilities will ensure there is no "problem of zeroes".

2.2.5 Cut-off parameter

Out of the three generated senses for each word, those senses which have very low probabilities (i.e. very low likelihood of being correct) are eliminated. The cut-off parameter is used to quantify what we mean by "very low probabilities". However, we also ensure that there is always atleast one "predicted sense" per word - we never eliminate all 3 predicted senses for any word. The classifier was tested for 3 values of the cut-off parameter. Details are provided in the next chapter.

2.3 Galley-McKeown's algorithm

2.3.1 Why Galley-McKeown's Algorithm?

Galley-McKeown's algorithm reports one of the highest WSD accuracies among all lexical chaining algorithms [Galley and McKeown, 2003; Navigli, 2009]. Here are the details of the performance of the algorithm as opposed to other popular lexical chaining algorithms-:

Algorithm	Accuracy
Barzilay and Elhadad	56.56%
Silber and McCoy	54.48%
Galley and McKeown	62.09%

The Galley-McKeown algorithm has two distinct steps where it first performs WSD and then proceeds to build lexical chains. Since the algorithm has a separate disambiguation step, it can easily be used in conjunction with another classifier to perform WSD.

2.3.2 Modifications to the algorithm

The algorithm is implemented almost identically to the original algorithm, except that the senses allowed for each word are limited to the three suggested by the Naive Bayes Classifier. Thus the classifier performs the following steps-:

1. It creates a table containing the 3 suggested senses of every distinct noun in a file from the test data.
2. It then forms links between every pair of senses in this table that are related as hypernyms, hyponyms or siblings.
3. It computes the "relation weight" for each of these pairs by consulting the table below which also factors in the distances between the sense pairs.
4. It sums over all the relation weights attached to each sense to produce the sense score.
5. For each word it selects that sense which has the highest sense score. This is the final word sense predicted by the ensemble classifier.
   (a) Note that in this step, the classifier assigns only 1 sense to every distinct noun in a file. There is an implicit assumption that each word will have only one sense per discourse.
6. This process is repeated for every file in the test data.

2.4 Test Data

The test data chosen were the Senseval-2 and Senseval-3 contest test databases. Almost all cutting edge WSD systems have been tested against Senseval contest data and therefore, this is a good benchmark to measure the performance of our classifier. Here is a review of the two competitions from Navigli, 2009.

Senseval 2 : Senseval-2 [Edmonds and Cotton 2001] took place in Toulouse (France) in 2001. Two main tasks were organized in 12 different languages: all-words and lexical sample WSD (see Section 2). Overall, 93 systems from 34 research groups participated in the competition. The WordNet 1.7 sense inventory was adopted for English.

Senseval-3 : The English all-words task [Snyder and Palmer 2004] saw the participation of 26 systems from 16 teams. The test set included 2037 sense-tagged words. The best system attained a 65.1% accuracy, whereas the first sense baseline (cf. Section 7.2.2) achieved 60.9% or 62.4% depending on the treatment of multiwords and hyphenated words.

Chapter 3

Implementation and Performance

The entire classifier was coded from scratch using Python, the nltk WordNet interface and sqlite 3. This chapter briefly describes the implementation and then measures the performance of the classifier constructed.

3.1 Preprocessing data

Since the entire program was written using sqlite databases, the first task was we faced was parsing the training and test data into sqlite databases. This was performed with the help of python's regular expressions module. The advantages of using sqlite were numerous - rapid table joins, easy looking up of data and data manipulation etc. The SemCor raw data was in the form of a group of folders containing sequentially named xml files, each containing one document of the corpus. There were 352 files in total. The following code snippet was written and used to construct an sqlite database from the xml files.

```
import sqlite3
import re
import nltk
from nltk.corpus import semcor
from sqlite3 import dbapi2 as sqlite
conn = sqlite.connect("semcor.db")
cur = conn.cursor()

cur.execute('drop table semcordata')
```

```
cur.execute('create table semcordata(serial integer, token varchar, cmd varchar,
    pos varchar, lemma varchar, wnsn integer, lexsn varchar, pn varchar, pnum
    integer, snum integer, filename varchar, filenumber integer, test integer,
    predictedwnsn integer, sgroupnum integer)')
text = re.findall('<.*>',semcor.raw())
#Skipping variable declarations here
for w in text:
        serialv = serialv + 1
        temp = re.search('<context filename="(.*?)"',w)
        if(temp):
                filenamev = temp.group(1)
                pnumv = '0'
                filenumberv = filenumberv + 1
        temp = re.search('<p pnum="(.*?)"',w)
        if(temp):
                pnumv = temp.group(1)
        temp = re.search('<s snum="(.*?)"',w)
        if(temp):
                snumv = temp.group(1)
        if(re.search('<wf*',w)):
                temp = re.search('>(.*?)</wf>',w)
                if(temp):
                        tokenv = temp.group(1)
                temp = re.search('cmd="(.*?)"',w)
                if(temp):
                        cmdv = temp.group(1)
                temp = re.search('pos="(.*?)"',w)
                if(temp):
                        posv = temp.group(1)
                temp = re.search('lemma="(.*?)"',w)
                if(temp):
                        lemmav = temp.group(1)
                temp = re.search('wnsn="(.*?)"',w)
                if(temp):
                        wnsnv = temp.group(1)
                temp = re.search('lexsn="(.*?)"',w)
                if(temp):
                        lexsnv = temp.group(1)

cur.execute('insert into semcordata (serial,token,cmd,pos,lemma,wnsn,lexsn,pn,
    pnum,snum,filename, filenumber, test) values (' + str(serialv) + ',lower("' +
     tokenv + '"),"' + cmdv + '","' + posv + '","' + lemmav + '","' + wnsnv +
    '","' + lexsnv + '","' + pnv + '","' + pnumv + '","' + snumv + '","' +
    filenamev + '",' + str(filenumberv) +','+ str(testv) + ' null, null)')

conn.commit()
conn.close()
```

An almost identical code snippet, with changes to factor the slightly different formatting of the Senseval-2 and Senseval-3 data was used to add the same to the sqlite database.

3.1.1 Stop Words

All stop words had the value "ignore" in the cmd field. Thus, once the sqlite database was constructed, the stop words could be removed with one single command.

3.1.2 Inconsistencies in test data

The test data had certain words classified with word sense tags -1. These were used to indicate words that were marked as "unknown" by the human taggers at Senseval-2 and Senseval-3. These words were ommitted from the testing data.

3.2 Implementation of the Naive Bayes Classifier

The Naive Bayes classifier was implemented in a simple fashion by maintaining tables with prior and conditional probabilities. Below is the code snippet that executes the classification task on the test data and selects the top 3 most probable senses, which is then passed on to the Galley-McKeown algorithm based classifier.

```
cur1.execute('select count(*) from classifyTestData')
totalnum = cur1.fetchone()
delta = 0.0001
theta = 0.000000001
cur1.execute('update classifyTestData set weight = (? + (count*1.0))/((totalcount
    *1.0) + (?*?))',(delta,delta,totalnum[0]))
cur1.execute('select *,rowid from classifyTestData')
rowData = cur1.fetchone()

while(rowData):
        newWeight1 = math.log(rowData[10])
        newWeight2 = -1.0*(rowData[9] + theta)*newWeight1/(1.0 + (totalnum[0]*
            theta))
```

```
        cur2.execute('update classifyTestData set weight = ? where rowid = ?',(
            newWeight2, rowData[11]))
        rowData = cur1.fetchone()

cur1.execute('select *, rowid from predictionsNBC')
rowData = cur1.fetchone()
ctr = 0
while(rowData):
        cur2.execute('select predwnsn, sum(weight) as netWeight from
            classifyTestData where token = ? and pos = ? and filenumber = ? group
             by predwnsn order by netWeight desc, predwnsn asc limit 3',(rowData
            [1], rowData[2], rowData[4]))
        senseRankData = cur2.fetchone()
        ctr = 0
        while(senseRankData):
                ctr = ctr + 1
                cur3.execute('insert into finalClassifyNBC values
                    (?,?,?,?,?,?,?,?)', (rowData[0], rowData[1], rowData[2],
                    rowData[3], rowData[4], senseRankData[0], senseRankData[1],
                    ctr))
                senseRankData = cur2.fetchone()
        rowData = cur1.fetchone()
```

3.3 Implementation of Galley-McKeown's Algorithm

Finally, the Galley-McKeown algorithm is implemented to obtain the final results. The code iterates, disambiguating one file at a time. For each file, the algorithm executes four major steps, each executed by one of the functions below.

```
#creates a cartesian product of the allowed senses in a document
createsensePairsNewForFile(filenumber)

#finds the weight of the semantic relationship between each pair in the cartesian
     product
getRelationWeightsForFile(filenumber)

#finds the distances between words
getPairDistancesForFile(filenumber)

#chooses the best sense based on the highest sum of weights
chooseBestSense(filenumber)
```

3.4 Evaluation Measures

We present here the evaluation measures and baselines employed for in vitro evaluation of WSD systems, that is, as if they were stand-alone, independent applications. We then employ these very measures to our classifier.

Let T = ($w_1, \ldots, w_n$) be a test set.

Let A be the "answer" function that provides a mapping from each word $w_i \in T$ to the appropriate sense from the sense inventory D i.e., A(i) $\subseteq$ $Senses_D(w_i)$.

We define coverage C as the percentage of tokens in the test set for which the system provided a sense assignment-:

$$\text{Coverage} = C = \frac{Number\ of\ answers\ provided}{Total\ number\ of\ answers\ that\ should\ be\ provided}$$

The precision of the system is the percentage of answers provided that are correct-:

$$\text{Precision} = P = \frac{Number\ of\ correct\ answers\ provided}{Total\ number\ of\ answers\ provided}$$

The Recall of the system is defined as the number of correct answers given by the system divided by the total number of answers that the system should have provided.

$$\text{Recall} = R = \frac{Number\ of\ correct\ answers\ provided}{Total\ number\ of\ answers\ that\ should\ be\ provided}$$

In WSD, recall is also commonly referred to as accuracy, although accuracy is given a different definition from Recall in other fields (like machine learning and information retrieval).

The F_1-measure or balanced F-score determines the weighted harmonic mean between the Precision and Recall. It is defined as

$$F_1 = \frac{2P.R}{P+R}$$

It has been argued that these measures do not take into consideration questions like "how much did it miss the mark by?" in terms of distance between the predicted sense and the actual sense. However, these measures have been widely used and it is useful to use the same results as it enables comparison with previously established results.

3.5 Baselines

There are many popular baselines in literature - the random baseline, the first sense baseline, the Lesk approach etc. However, since we are testing our classifiers on Senseval data, we can use the competition baselines to evaluate our performance. Unfortunately, the senseval competitions All Words Tasks included disambiguation of Adjectives, Verbs and Adverbs. We consult the paper by Martinez, Lopez and Agirre titled "On the Use of Automatically Acquired Examples for All-Nouns WSD", 2008 for the "Nouns-only" scores of the various systems participating in Senseval-3. We also use the Nouns-only Semcor baseline and measure our performance.

Finally, we also compare the performance of our ensemble classifier with the our implementation of Galley-McKeown's classifier, illustrating the improvement in performance brought about by ensemble classifier.

3.6 Performance of the Ensemble Classifier on Senseval

The details of the cut-off parameter have been explained in the previous chapter. Here are the results for various settings of the cut-off parameter.

3.6.1 Senseval-2

Without the use of a cut-off parameter

Coverage = 100%

Precision = Recall = F-measure = 562/1027 = 54.722%

Using cut-off parameter set to remove all extra senses that are less than 1/3rd the average score

Coverage = 100%

Precision = Recall = F-measure = 636/1027 = 61.927%

Using cut-off parameter set to remove all extra senses that have less than half the average score

Coverage = 100%

Precision = Recall = F-measure = 649/1027 = 63.193%

Using cut-off parameter set to remove all extra senses that have a score less than the average score

Coverage = 100%

Precision = Recall = F-measure = 667/1027 = 64.946%

3.6.2 Senseval-3

Without the use of a cut-off parameter

Coverage = 100%

Precision = Recall = F-measure = 480/872 = 55.045%

Using cut-off parameter set to remove all extra senses that are less than 1/3rd the average score

Coverage = 100%

Precision = Recall = F-measure = 523/872 = 59.977%

Using cut-off parameter set to remove all extra senses that have less than half the average score

Coverage = 100%

Precision = Recall = F-measure = 531/872 = 60.894%

Using cut-off parameter set to remove all extra senses that have a score less than the average score

Coverage = 100%

Precision = Recall = F-measure = 548/872 = 62.844%

Code	Revised Precision	Precision	Recall	F-measure	S/U
SenseLearner	70.75745413	65.9	65.9	65.9	S
SenseCorpus-S	70.27924312	65.3	65.3	65.3	S
LCCaw	70.27924312	65.3	65.3	65.3	S
kuaw.ans	69.88073394	64.8	64.7	64.7	S
R2D2English	69.64162844	64.5	64.5	64.5	S
GAMBL-AW	68.68520642	63.3	63.3	63.3	S
upv-eaw.upv-eaw2	68.68520642	63.3	63.3	63.3	S
Meaning	68.60550459	63.2	63.2	63.2	S
upv-eaw.upv-eaw	68.36639908	62.9	62.9	62.9	S
Semcor Baseline	67.96788991	62.4	62.4	62.4	S
UJAEN2	67.96788991	62.4	62.4	62.4	S
Constructed Ensemble Classifier	62.844	?	62.844	62.844	S + U

Figure 3.1: Performance against state-of-the-art classifiers

3.6.3 Comparitive Performance

The performance of our classifier is shown along side the best supervised, semisupervised and unsupervised classifiers on Senseval-3 data, by setting the cut-off parameter to remove all extra senses that have less than half the average score. Please note that the scoring system used here is **different from the scoring system** used elsewhere in this paper. The scoring system used in this table considers the classification of polysemous words only. To obtain the revised score for these systems, it was assumed that the systems tagged all monosemous nouns correctly. The official Senseval results are also published below. Please note, however, that the official results are for the All-Words-Task (adverbs, verbs, nouns and adjectives).

3.6.4 Ensemble classifier versus Galley-McKeown's algorithm

The ensemble classifier improves the Galley-McKeown algorithm dramatically with respect to WSD performance. The comparative results on a test data set consisting of both Senseval-2 and Senseval-3 is shown below.

Performance of our implementation of Galley-McKeown's Algorithm (without shortlisting by Naive Bayes)

All Words		
Accuracy	System	U/S
69.0	SMUaw	S
63.6	CNTS-Antwerp	S
61.8	Sinequa-LIA	S
57.5/56.9	UNED—AW-U2	U
55.6/55.0	UNED—AW-U	U
57.0	MFS BL	S

Figure 3.2: Official Senseval All-Words scores

Coverage = 100%

Precision = Recall = F-measure = 979/1899 = 51.553%

Performance of ensemble classifier without the use of a cut-off parameter

Coverage = 100%

Precision = Recall = F-measure = 1034/1899 = 54.449%

Performance of ensemble classifier with the use of a cut-off parameter set to remove all extra senses that have less than half the average score

Coverage = 100%

Precision = Recall = F-measure = 1180/1899 = 62.1379%

Using cut-off parameter set to remove all extra senses that have a score less than the average score

Coverage = 100%

Precision = Recall = F-measure = 1215/1899 = 63.981%

Chapter 4

Conclusions and Future Work

In this chapter we present conclusions from our research in terms of the advantages our ensemble classifier provides over Galley's algorithm, how it compares with other state-of-the-art WSD systems and the disadvantages of the ensemble classifier. We also discuss some possibilities for future research from ideas related to this work.

4.1 Advantages of using the ensemble classifier

It is clearly evident from the results obtained that the ensemble classifier has a number of advantages over a direct implementation of Galley-McKeown's algorithm-:

1. Accuracy - The WSD performance goes up dramatically from a low 51% to as high as 64% when the ensemble classifier is used.

2. Computational efficiency - The ensemble classifier is far more computationally efficient as the number of senses per word is limited to 3 (where as in some cases it can be as high as 40). Thus the number of sense-pair comparisons reduces and the algorithm performs more efficiently. While the Naive Bayes Classifier does take some time to train, it is a one time effort, after

which the classifier is ready to tag test data. Since Galley's algorithm is already an O(n) algorithm and the ensemble classifier can perform both faster and better, it is evident that there are many uses to this approach.

3. Lexical Chain construction - The ensemble classifier may be a way forward for all lexical chaining algorithms. By restricting the number of senses over which the chains are built, we ensure that unrelated but "valid" chains are successfully ignored, chains are chosen with greater accuracy and chains are constructed with greater efficiency. WSD performance is one of the ways of measuring the performance of a lexical chaining algorithm - we see a marked increase in WSD performance in the ensemble classifier.

4. While Galley-McKeown's algorithm itself fails to beat an independent Naive Bayes Classifier on the Senseval test data (Naive Bayes Classifiers have shown accuracies of upto 71% when used with rich feature sets on Senseval data [Le and Shimazu, 2004]), it shows dramatic improvement from it's previous performance. This is an encouraging sign for the idea of building better lexical chaining algorithms by using ensemble classifiers that take the help of a supervised classifier to shortlist the allowed senses of each word.

4.2 Fall-backs of the classifier

The ensemble classifier failed to beat the Naive Bayes Classifier in performance. Thus, while the ensemble classifier is a great improvement over a traditional lexical chaining algorithm, it fails to achieve WSD performance that beats the performance of the Naive Bayes Classifier. If the Naive Bayes Classifier was just made to choose it's top 1 sense for every word, it achieves an accuracy of 69% (which translates to around 65% if we consider polysemous nouns only) thus beating the ensemble classifier independently.

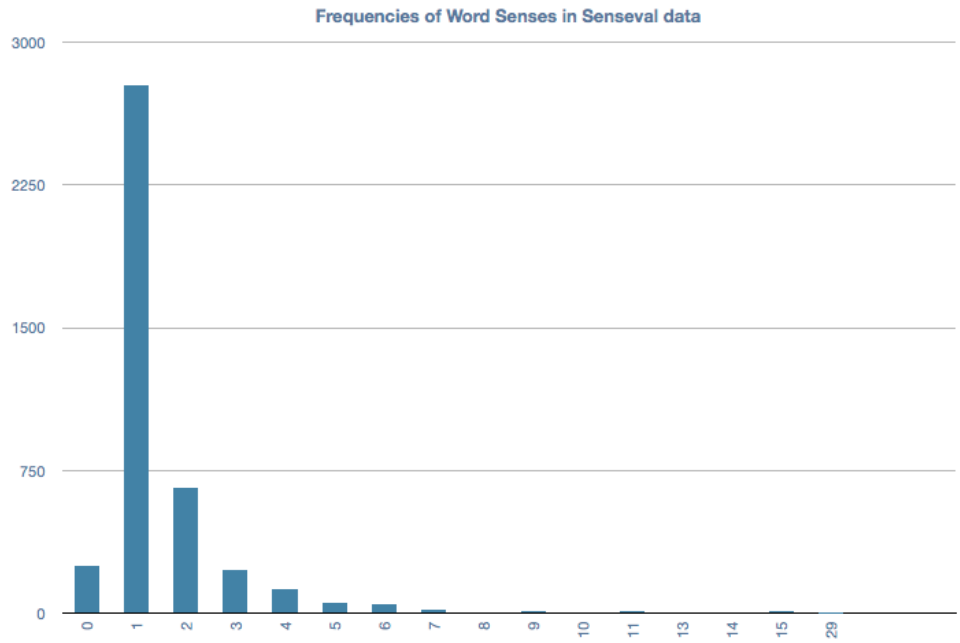

Figure 4.1: Sense frequencies in senseval data

4.3 Future Work

This work opens up the case for building a lexical chain algorithm that goes hand in glove with a supervised classifier. Since the supervised classifier will increase the efficiency by shortlisting candidate senses, we can use a more computationally intensive lexical chaining algorithm. We can also carefully choose the properties of the lexical chaining algorithm and the supervised classifier so that they act together in a complementary fashion to produce better results. Given the high occurence of first sense terms, it would perhaps be wiser to construct a powerful binary classifier, that simply classifies words as “first sense” or “not first sense”. After this, the lexical chaining algorithm can try and determine the senses of the “not first sense” words, assuming that the remaining words are all first sense. This approach is further supported by this graph of sense frequencies in senseval. SemCor also shows a similar graph.

Bibliography

1. **Roberto Navigli. 2009. Word Sense Disambiguation: A Survey.** Universit ‘a di Roma La Sapienza, ACM Computing Surveys, Vol. 41, No. 2, Article 10, Publication date: February 2009.

2. **Michel Galley and Kathleen McKeown. 2003. Improving word sense disambiguation in lexical chaining.** In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI, Acapulco, Mexico). 1486–1488.

3. **C. Manning and H. Schutze. 1999. Foundations of Statistical Natural Language Processing.** MIT Press, Cambridge, MA.

4. **David Martinez, Oier Lopez de Lacalle, Eneko Agirre. 2008. On the Use of Automatically Acquired Examples for All-Nouns Word Sense Disambiguation.** In Journal of Artificial Intelligence Research 33 (2008) 79-107.

5. **Abhishek Ghose and B. Ravindran. 2011. Supervised Lexical Chaining.** Masters thesis, IIT Madras.

6. **W.A. Gale, K. Church, and D. Yarowsky. 1992. One sense per discourse.** In Proceedings of the DARPA Speech and Natural Language Workshop (Harriman, NY). 233–237.

7. **Cuong Anh Le and Akira Shimazu. 2004. High WSD accuracy using Naive Bayesian classifier with rich features.** PACLIC 18, December 8th-10th, 2004, Waseda University, Tokyo.

8. **Rada Mihalcea. 2004. Co-training and Self-training for Word Sense Disambiguation.** In Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL, Boston, MA). 33–40.

9. **Anja Nikunen. 2007. Different approaches to word sense disambiguation.** Language technology and applications

10. **G. Hirst and D. St-Onge. 1998. Lexical chains as representations of context for the detection and correction of malapropisms.** In WordNet: An Electronic Lexical Database, C. Fellbaum, Ed. MIT Press, Cambridge, MA, 305–332.

11. **Tim Berners-Lee, J. Hendler and O. Lassila. 2001. The semantic Web.** http://www.sciam.com/article. cfm?id=the-semantic-web&page=2.

12. **Jurafsky and Martin**. **2000. Speech and Language Processing**. Prentice Hall, Upper Saddle River, NJ.