Computer Science Department
CPSC 097

Class of 2003 Senior Conference on Natural Language Processing

Proceedings of the Conference

Spring 2003
Swarthmore College
Swarthmore, Pennsylvania, USA

Order copies of this proceedings from:

Computer Science Department
Swarthmore College
500 College Avenue
Swarthmore, PA 19081
USA
Tel: +1-610-328-8272
Fax: +1-610-328-8606
`richardw@cs.swarthmore.edu`

Introduction

About CPSC 097: Senior Conference

This course provides honors and course majors an opportunity to delve more deeply into a particular topic in computer science, synthesizing material from previous courses. Topics have included advanced algorithms, networking, evolutionary computation, complexity, encryption and compression, and parallel processing. CPSC 097 is the usual method used to satisfy the comprehensive requirement for a computer science major.

During the 2002-2003 academic year, the Senior Conference was led by Richard Wicentowski in the area of Natural Language Processing.

Computer Science Department

Charles Kelemen, Edward Hicks Magill Professor and Chair
Lisa Meeden, Associate Professor
Tia Newhall, Assistant Professor
Richard Wicentowski, Assistant Professor
Ali Erkan, Visiting Assistant Professor

Program Committee Members

Mark Angelillo	Haw-Bin Chai
Julie Corder	Hollis Easter
Sean Finney	Kuzman Ganchev
Todd Gillette	Feng He
Nori Heikkinen	Oliver Hsu
Yoshi Komori	J. McConnell
Shannon McGrael	Erik Osheim
Sven Olsen	Jeff Regier
Ben Schak	Mike Smith
Michael Spiegel	Daniel Sproul
Andrew Stout	Yeelin Tan
Anteneh Tesfaye	Pascal Troemel
Jonah Volk	Richard Wicentowski
Andy Zuppann	

Conference Website

`http://www.cs.swarthmore.edu/~richardw/cs97-s03/`

Conference Program

Thursday, April 17

2:40-3:05 *Hebrew Vowel Restoration With Neural Networks*
Michael Spiegel and Jonah Volk

3:05-3:30 *One Sense per Collocation for Prepositions*
Hollis Easter and Benjamin Schak

3:30-3:55 *The Structure, Computational Segmentation, and Translation to English of German Nominal Compounds*
Nori Heikkinen

3:55-4:20 *Using Semantic Information from Neural Networks to Detect Context-Sensitive Spelling Errors*
Julie Corder

Tuesday, April 22

2:40-3:05 *Part Of Speech Tagging Using A Hybrid System*
Sean Finney and Mark Angelillo

3:05-3:30 *Disambiguating Between 'wa' and 'ga' in Japanese*
Yoshihiro Komori

3:30-3:55 *Machine Translation Evaluation by Document Classification and Clustering*
Feng He and Pascal Troemel

3:55-4:20 *Constructing a Lossless and Extensible Part-Of-Speech Tagger*
Jeffrey Regier

Thursday, April 24

2:40-3:05 *A Hybrid WSD System using Word Space and Semantic Space*
Haw-Bin Chai and Hwa-chow Hsu

3:05-3:30 *A Minimally-Supervised Malay Affix Learner*
Yee Lin Tan

3:30-3:55 *code-name DUTCHMAN: A Text Summarization System*
Erik Osheim and Daniel Sproul

Tuesday, April 29

2:40-3:05 *Wordnet Wordsense Disambiguation using an Automatically Generated Ontology*
Sven Olsen

3:05-3:30 *A Connectionist Approach to Word Sense Disambiguation*
Andy Zuppann

3:30-3:55 *Automatic Rule Generation and Generalization for an Information Extraction System Using MAXIM*
J. McConnell and Anteneh Tesfaye

Tuesday, April 29

Thursday, May 1

Table of Contents

Hebrew Vowel Restoration With Neural Networks

M. Spiegel and **J. Volk**
Swarthmore College
Computer Science Dept.
{spiegel,volk}@cs.swarthmore.edu

Abstract

Modern Hebrew is written without vowels, presenting a problem for those wishing to carry out lexical analysis on Hebrew texts. Although fluent speakers can easily replace vowels when reading or speaking from a text, there are no simple rules that would allow for this task to be easily automated. Previous work in this field has involved using statistical methods to try to solve this problem. Instead we use neural networks, in which letter and morphology information are fed into a network as input and the output is the proposed vowel placement. Using a publicly available Hebrew corpus containing vowel and morphological tagging, we were able to restore 85% of the correct vowels to our test set. We achieved an 87% success rate for restoring the correct phonetic value for each letter. While our results do not compare favorably to previous results, we believe that, with further experimentation, our connectionist approach could be made viable.

1 Introduction

As would be expected from the lack of vowels, Hebrew text contains a large amount of ambiguous words. Levinger et al. (1995) calculated, using a Modern Hebrew newspaper corpus, that 55% of Hebrew words are ambiguous[1]. This ambiguity can take several forms: different vowel patterns can be used to distinguish between multiple verb or noun forms, as well as between multiple forms of other parts of speech, such as certain prepositions. Vowel patterns also distinguish between two unrelated words in certain cases. As an example of the following, the consonant string SPR[2] can be vowel tagged in one way such that it means "book" and another way such that it means "to count". This consonant string also has four other possible vowel patterns, each with a different translation.

The problem is further complicated by the fact that Hebrew has twelve vowels, designated in our corpus (discussed below) as {A, F, E, ", I, O, U, W., :, :A, :E, :F}. However, the number of vowels can sometimes be simplified by using phonetic groupings. These are groups of vowels for which all the vowels in the group produce equivalent (or near-equivalent) sounds. As will be discussed later, in certain situations it is enough to produce a vowel from the phonetic group of the target vowel, rather than having to produce the exact vowel. We have identified the following phonetic groups: {A, F, :A, :F}, each of which produce, roughly, the sound "ah" and {E, :E}, each of which produce, roughly, the sound "eh".

We believe that there are two main areas to which this work could be applied, each of which demands somewhat different goals. The first is automated speech generation, which, of course, requires vowel restoration. For this task, we would only need phonetic group accuracy because the vowels within phonetic groups all make the same sounds in spoken Hebrew. The second task is the restoration of vowels to Hebrew texts for the purpose of aiding people who are learning the language, either children learning it as their first language or adults. For this task, we would not be able to combine phonetic groups.

2 Previous Work

Previous relevant work falls into two main categories: work done in the field of Hebrew vowel restoration and work done using neural networks to solve lexical problems which could be similar to vowel restoration. Note that these categories are mutually exclusive; to the best of our knowledge, no previous work has been done which has attempted to combine these fields as we are.

This work makes use of a number of performance metrics which are defined as follows: Word accuracy is the percentage of words which have their complete vowel pattern restored exactly. Letter accuracy is the percentage

[1] It is unclear whether this refers to types or tokens.

[2] Throughout this paper we have used an ASCII representation in place of actual Hebrew letters and vowels.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

of letters which are tagged with the correct vowel. W-phonetic group accuracy is word accuracy, allowing for vowels to be substituted for others within a given phonetic group. L-phonetic group accuracy is letter accuracy, allowing for vowels to be substituted within phonetic groups.

In the field of Hebrew vowel restoration, the most recent work can be found in Gal (2002). This paper attempts to perform vowel restoration on both Hebrew and Arabic using Hidden Markov models. Using a bigram HMM, Gal achieved 81% word accuracy for Hebrew and 87% W-phonetic group accuracy. He did not calculate letter accuracy, but we have to assume that it would have been higher than 81%. Gal also used the Westminster Database as a corpus and calculated that approximately 30

Similar work was also done in Yarowsky (1994), where he addressed accent restoration in Spanish and French. He, like Gal, uses statistical methods - decision lists in this case. His descision lists rely on both local and document-wide collocational information. While this task is quite similar to ours, French and Spanish accent patterns are much less ambiguous than Hebrew vowel patterns; Yarowsky cites baseline values in the 97-98% range. Given this baseline, he is able to achieve 99% accuracy.

Gal also cites a Hebrew morphological analyzer called Nakdan Text (Choueka and Neeman 1995) which uses context-dependent syntactic rules and other probabilistic rules. Nakdan Text uses the morphological information that it creates as a factor in vowel placement, meaning that its results are most comparable to ours obtained using morphology tags. Nakdan Text achieved 95% accuracy, but the authors do not specify if this is word accuracy or letter accuracy.

In the field of neural networks, networks have, in a number of past experiments, been applied to part-of-speech tagging problems, as well as used to solve other lexical classification problems. For example, Hasan and Lua (1996) applied neural nets to the problems of part-of-speech tagging and semantic category disambiguation in Chinese. In Schmid (1994), the author describes his Net-Tagger software which uses a connectionist approach to solve part-of-speech tagging. Schmid's Net-Tagger software performs as well as a trigram-based tagger and outperforms a tagger based on Hidden Markov Models. Given these results, it seems likely that a connectionist approach would produce satisfactory results when applied to vowel restoration in Hebrew, given that vowel restoration is closely linked with lexical categorization.

3 Corpus

We used the Westminster Hebrew Morphological Database (2001), a publicly available corpus containing the complete Tanakh (Hebrew Bible), tagged with both vowels and morphological information. We would have ideally used a corpus of Modern Hebrew, but to our knowledge, there are no Modern Hebrew corpora which are vowel tagged. Throughout this paper we have used the codes from the Westminster Database in which ASCII characters are substituted for Hebrew letters and vowels. Normally, Hebrew is written with the vowels positioned underneath the consonant, with each consonant taking either one vowel or no vowels. In the Westminster notation, vowels are positioned to the right of vowels. For example, the string 'RA' represents what would be written in Hebrew as the consonant 'resh' with the vowel 'qamets' underneath it.

3.1 Morphology

As mentioned above, the Westminster Database includes morphological tags. We chose to run our main experiment using these tags as input for our neural network, given that vowel placement is often tied to morphology. If this system were to be applied to a corpus which had no morphology tags, we would assume that the corpus would be first run through a morphology tagger. In addition, we ran our network once without morphological information as a baseline of sorts.

The morphology tagging in the Westminster Corpus is very broad and varied, although it has certain gaps, as mentioned in our data analysis section. There are several layers of tags, each of which provides more specific information. The most basic level includes tags to distinguish between particles, nouns, adjectives, and verbs. The particle tag is further broken down into tags for articles, conjunctions, prepositions, etc. The tags for pronouns, nouns, and adjectives are subdivided by tags for gender, plurality distinctions, 1st/2nd/3nd person distinctions, etc. The verb tags have a wide range of sub-tags, including the standard ones mentioned above, but also including a variety of tags designed to differentiate between different Hebrew and Aramaic[3] verb forms.

4 Implementation

4.1 Parsing the Corpus

The raw data from our corpus is first run through a series of parsing scripts before becoming input for our neural network. First we remove the initial part of the tag which corresponds to the location in Tanakh (book, chapter, verse, etc.). We then run the following series of parsing tools as necessary:

[3] Aramaic is an ancient Semitic language closely related to Hebrew. Portions of the Hebrew Bible are written in Aramaic, although Genesis, the only section we examined, contains only Hebrew.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

• We recombine words which are written as one word in standard Hebrew but which our corpus splits into multiple words for morphological purposes. These are quite often particles which can be attached to nouns as prefixes. These include the conjunction W, the prepositions B,K,L,M, the relative particle $, the definite article H, etc. The combined word has the morphological tags of both of its components. To illustrate the way this works, consider the Hebrew word "R")$IYT" meaning "beginning." The prefix "B.:" meaning "in" can be added to form the compound "B.:R")$IYT" meaning "in the beginning"[4]. In the Westminster Database, the two parts of this word are included as two separate words, the first of which is tagged as a preposition and the second of which is tagged as a noun. For our purposes, we recombine these parts into one word which is tagged as both a preposition and a noun.

• The corpus contains certain word pairs which are slightly modified versions of each other and which occupy a single morphological position within the sentence. One of these words is tagged as the Ketiv version (*) and one is tagged as the Qere version (**). For our purposes we have eliminated the archaic and unvoweled Ketiv version[5].

• In some cases, two words are joined together with either a Maqqef (-) or a compound joint (˜). In these cases we have split the word into two words, given that they can both function as independent words.

• While the dagesh (.) could be considered a vowel, we chose not to attempt to restore it. The dagesh is a character which can be used to distinguish between certain consonant pairs. For example, the string 'B' is pronounced like the English consonant 'v', while the string 'B.' is pronounced like the English consonant 'b'. The dagesh is placed inside the consonant, rather than under it, as with vowels, and a consonant can have both a vowel and a dagesh. Thus, using it would force us to modify our system, which is currently designed to handle one vowel per consonant. Furthermore, our system does not need to deal with the dagesh, as its placement always follows a set of simple rules.

• All other extraneous characters are thrown out: accents (ˆ), affix/suffix separators(/), etc.

4.2 Implementing the Neural Network

The neural network consists of an input matrix, an output matrix, and, in some cases, a hidden layer (see Figure 2). We used a standard feed-forward, fully connected network. The input matrix consists of a single word with morphological tags. Before being used as input, the word is stripped of vowels, so only the consonants are used. Each word is represented as a two-dimensional array of 24 by 15 nodes, where 24 is the number of letters in the Hebrew alphabet plus one null character and 15 is the length of the longest word in the corpus. In addition, each has 104 nodes which represent morphological tags, as discussed above. The output matrix consists of a 14 by 15 node array. Each node in the output matrix represents a possible vowel with one additional node corresponding to the consonant having no associated vowel and one additional node corresponding to the null letter. The activated node in each row signifies the proposed vowel for the corresponding consonant.

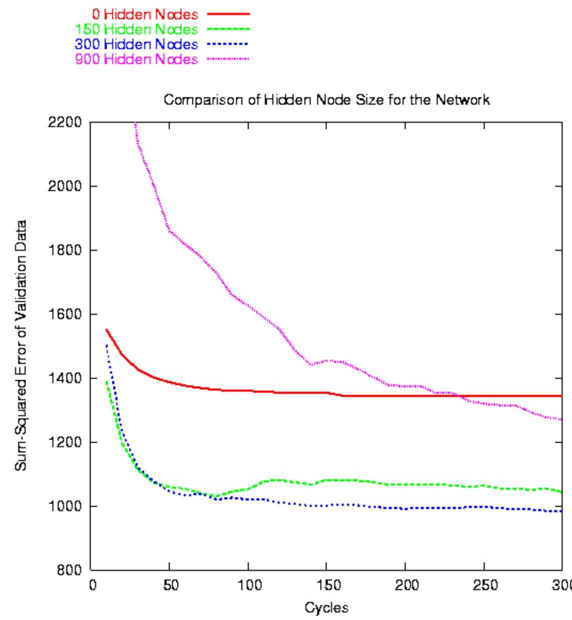


Figure 2: Network Error Based on Hidden Layer Size

Rather than using the entire Bible as input, we chose to use half of the Book of Genesis to cut down on computation time. This consisted of 10306 words, of which 90% were used for training and 10% for testing.

Although previous literature suggested that hidden nodes were not desirable for similar tasks, we ran a number of tests with hidden layers of different sizes, one containing 150 nodes, one with 300, and one with 900.

5 Results

As shown in Figure 2, the neural network with 300 hidden nodes outperformed all the other network configurations. Therefore, all of our results are based on that

[4]Technically, a more precise translation would be "in a beginning" or "at first," but "in the beginning" is the generally accepted translation.

[5]During the period when vowels were added to the Hebrew Bible (1st century CE), occasionally grammatical corrections were added to "fix" portions of the text. The original unvoweled text has been preserved as the "ketiv," and the modified voweled text is called the "qere."

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

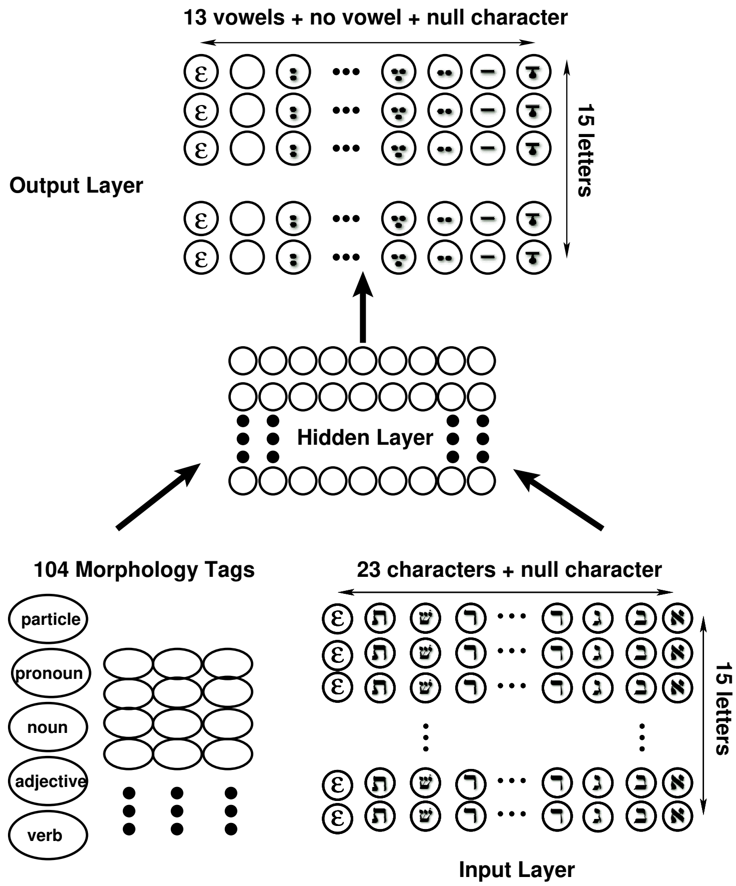


Figure 1: Neural Network Configuration

	A	F	E	“	I	O	U	W.	:	:A	:E	:F	_
Correct Vowels	356	352	216	155	277	224	2	65	381	70	12	0	1193

Table 1: Neural Network correct vowel recognition, with morphology information.

	A	F	E	“	I	O	U	W.	:	:A	:E	:F	_	ϵ
A		16	4	11	16	10	0	0	23	2	0	0	5	2
F	22		17	4	10	14	0	1	33	3	0	0	11	2
E	11	11		10	11	13	0	0	7	3	0	0	4	0
“	7	17	9		8	6	0	0	3	0	1	1	3	0
I	7	7	4	5		1	0	0	10	0	0	0	2	1
O	7	12	7	1	4		0	0	6	0	0	0	13	0
U	1	2	1	0	1	2		0	1	0	0	0	0	0
W.	1	2	0	0	0	0	0		5	0	0	0	19	0
:	12	19	8	4	6	8	1	1		0	0	0	9	1
:A	6	3	0	3	2	5	0	0	8		3	0	0	0
:E	1	0	0	2	0	0	0	0	0	0		0	0	0
:F	0	0	0	0	0	0	0	0	1	0	0		0	0
_	5	3	0	2	3	6	0	1	2	0	0	0		2

Table 2: Neural Network error distribution, with morphology information.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

	A	F	E	“	I	O	U	W.	:	:A	:E	:F	_
Correct Vowels	355	334	230	141	242	201	4	69	363	72	14	0	1190

Table 3: Neural Network correct vowel recognition, without morphology information.

	A	F	E	“	I	O	U	W.	:	:A	:E	:F	_	ϵ
A		22	10	13	15	1	3	1	16	3	1	0	3	2
F	17		19	20	9	12	3	2	34	3	1	2	12	1
E	12	12		7	6	9	0	0	6	4	0	0	0	0
“	11	20	12		10	7	0	0	6	2	0	0	1	0
I	18	10	8	8		6	0	0	14	0	1	1	6	0
O	10	18	11	5	3		1	2	8	2	0	0	13	0
U	1	3	2	0	0	0		0	0	0	0	0	0	0
W.	1	2	0	0	1	0	0		2	0	0	0	17	0
:	26	24	6	4	9	7	1	1		0	1	0	8	0
:A	7	8	3	1	1	2	2	0	2		0	0	1	1
:E	1	0	0	0	0	0	0	0	0	0		0	0	0
:F	0	0	0	0	0	0	0	0	1	0	0		0	0
_	3	3	2	0	1	7	0	8	2	0	0	0		1

Table 4: Neural Network error distribution, without morphology information.

configuration. We achieved 85% letter accuracy, getting 3306 vowels correct out of 3883 total vowels. By merging phonetic groups, we achieved 87% L-phonetic group accuracy, getting 3363 vowels correct. Table 2 shows the error distribution for this experiment and Table 1 shows the distribution of correct results. In these tables, _ represents a consonant with no vowel attached to it, and ϵ represents a null consonant. Further results are forthcoming, including word accuracy, the average percentage of correct vowels per word, and results which look at the type level, rather than at the token level.

We also trained a network on the same data set, but without including morphological information. The hope was that this would let us achieve a baseline, to which our later results could be compared. This experiment achieved a letter accuracy of 74%. Thus, using morphology allowed us to increase our results by 11%. Table 4 shows the error distribution for this no-morphology experiment and Table 3 is the distribution of correct results.

6 Data Analysis

Table 2 demonstrates the errors produced by our network on a vowel by vowel basis. Each entry in the table contains the number of times that the vowel on the row was expected but the vowel on the column was placed instead. Based on this information, several trends emerge. The first is related to the high level of confusion between F and : (33 errors) and vice versa (19 errors). By examining our data, we were able to determine that the vast majority of these errors were due to a morphological feature of Hebrew: prepositions and articles are attached to nouns as prefixes and these preposition and article tags can often be combined to form a single prefix. For example, the prefix “L:-” means “to” or “to a[6],” as in “I went to a party.” It can then be combined with the definite article “HF-” to form the prefix “LF-” meaning “to the,” as in “I went to the party.” Several other examples exist that follow the same pattern; a complete list of prefixal particles is included in the section on parsing the corpus. However, there is no morphological information in the Westminster corpus that would distinguish between these two prefix forms. Given that these prefixes occur in very similar situations, the network was not able to determine the correct vowel to be used in such situations, leading to errors of the type that we found.

We also observed that the vowels “patah” and “qamets,” designated in the Westminster corpus as A and F, respectively, were frequently confused. We believe that this is due to the similarity of these vowels. While more detailed analysis of the Hebrew language would be necessary to determine this for sure, we believe that this similarity exists given that, in spoken Hebrew, these vowels make the same sound (equivalent to the English vowel ’a’). These errors are removed when phonetic group accuracy is calculated, and we believe that using a larger test set might help to minimize these errors. The large amount of errors where A was confused with : presumably result from the combination of these two problems,

[6]Both of these meanings are valid because Hebrew has no indefinite article.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

and would be solved if the other problems were solved.

Note the far-right column in Table 2. These were instances in which the network suggested that ϵ was the vowel to be placed under the current consonant. This, of course, makes no sense, given that ϵ represents the null-consonant vowel, in other words, the vowel that is to be placed with a null consonant. We have no idea why these cases exist, other then that it is simply a quirk of the neural network. Luckily, there are very few of these cases - it is not a major problem.

7 Future Work

There are several areas in which we believe our approach could be improved to hopefully produce results comparable to those obtained through other means. First, we could use more input text. For our results presented here, we chose not to use a larger data set due to a desire to minimize time spent running the network rather than due to a lack of data. This would potentially minimize errors produced due to the network never having seen certain words in the testing corpus.

Second, we could experiment more with hidden layer sizes. We only tried running the three variations mentioned above; perhaps if we had tried others we would have come across one that outperformed the network with 300 hidden nodes.

Third, we could expand our network input to include a broader context of text, perhaps one word on either side of the target word. Given that a bigram model has been shown to work, there is clearly a correlation between word context and vowel placement. However, it is possible that the morphological information that we provide performs a similar role to the role that would be played by context information. In addition, if we wanted to include such information, we would have to find some way of representing it within the context of the network. A problem arises because the network is letter-based, so we would have to figure out how to use words as input. One solution would be to include only the morphological information for the preceding and following words, possibly only the part-of-speech tag. It seems possible that this morphology would be the crucial element in determining the vowel placement for the word in question.

Finally, we could modify the corpus to include tags that distinguish between definite and indefinite article-prefix compounds. The Westminster Database does not include such tags presumably because this distinction does not arise outside of these compounds, given the lack of an indefinite article in Hebrew. This would hopefully solve the problem mentioned earlier that was accounting for a large amount of our vowel placement errors.

In addition to applying these measures to improve our results, a further test of our system would be to apply it to Modern Hebrew text, such as that from an Israeli newspaper archive. The results obtained from such a test would certainly be poor, given the fairly major differences between Biblical and Modern Hebrew. In a Modern Hebrew corpus, we would certainly encounter words that do not appear in the Bible, as well as words which were in the Bible but have since been modified in some way. Our morphological information would also potentially be faulty due to changes in the language over the years. If we wanted to optimize our system for Modern Hebrew we would definitely have to obtain a Modern Hebrew voweled corpus, either by finding one or by creating one by having native speakers tag an unvoweled corpus.

8 Conclusions

From our results, we have to confront the apparent fact that neural networks are not ideal for solving Hebrew vowel restoration. Given that we were including morphological information, we would hope that our results would be comparable to those achieved by Nakdan Text, the only other Hebrew vowel restorer which takes morphology into account. However, our results are a full 10% behind Nakdan Text, assuming that Nakdan is citing a letter accuracy (if they are citing a word accuracy, the difference would be even greater). This data, combined with the Gal results, certainly seems to suggest that a statistical approach to the problem may be superior. However, given the possible improvements to our methods as well as the fact that our results were at least promising, we believe that it might be possible to develop a connectionist system that could perform equally well as, or even outperform, a statistical system.

9 References

References

Y. Choeka and Y. Neeman. 1995. Nakdan Text, (an In-context Text-Vocalizer for Modern Hebrew) *BISFAI-95, The Fifth Bar Ilan Symposium for Artificial Intelligence*

Ya'akov Gal. 2002. An HMM Approach to Vowel Restoration in Arabic and Hebrew *Semitic Language Workshop*

M. Levinger, U. Ornan, A. Itai. 1995. Learning Morpho-Lexical Probabilities from an Untagged Corpus with an Application to Hebrew *Computational Linguistics*, 21(3): 383-404.

Md Maruf Hasan and Kim-Teng Lua. 1996. Neural Networks in Chinese Lexical Classification

Westminster Hebrew Institute. 2001. The Groves-Wheeler Westminster Hebrew Morphology Database, Release 3.5

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

David Yarowsky. 1994. Decision Lists for Lexical Ambiguity Resolution: Applications to Accent Restoration in Spanish and French *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 1–7
Computer Science Department, Swarthmore College

One sense per collocation for prepositions

Hollis Easter & Benjamin Schak

May 7th, 2003

Abstract

This paper presents an application of the one-sense-per-collocation hypothesis to the problem of word sense disambiguation for prepositions. The hypothesis is tested through translation using a bilingual French-English corpus. The paper shows that one-sense-per-collocation does hold for prepositions.

1 Introduction

The one-sense-per-collocation hypothesis (Yarowsky 1993) states that words[1] tend to occur with only one sense within different instances of the same collocation. Yarowsky (1993) tested this hypothesis with strong results on coarse-grained senses of ambiguous nouns, verbs, an adjectives. Although Martinez and Agirre (2000) achieved weaker results for fine-grained sense distinctions, the hypothesis can help a wide range of natural language processing tasks. Since the one-sense-per-collocation hypothesis is implicit in much of the previous work, such as (Japkowicz, 1991) on translating prepositions, an evaluation of the Hypothesis could yield improvement in translation systems. This paper discusses compelling reasons for why the Hypothesis should hold, and tests the Hypothesis on a bilingual English-French corpus.

Our first problem is how to define senses for prepositions. Yarowsky (1993) gives several ways to approach this. One way is the "hand-tagged homograph method," in which one uses a corpus tagged with the correct senses of each word. This won't work for us because no corpus known to us has reliable sense distinctions for prepositions. We also want to avoid methods based on

[1] Words with more than one sense are polysemes.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

homophones, ambiguities in online character recognition, and pseudo-words because the closed class of prepositions is too small. So, we equate the notion of a sense with that of a French translation.

1.1 Subcategorization

As noted above, there are two linguistic observations that recommend the one-sense-per-collocation hypothesis. The first of these is subcategorization, the notion that every noun, verb, and adjective selects (or "takes") certain types of phrases for complements, and can determine the heads of those complements. For example, consider the English adjective *interested*, translated into French as *interessé*. Sentences (1) and (2) show that *interested* must take a prepositional phrase headed by the preposition *in* as its complement, while *interessé* must take a prepositional phrase headed by *par*.

(1) John is interested *math / in math / *for math / *to math / *mathematic / *to do math.

(2) Jacques est interessé *les maths / par les maths / *pour les maths / * aux maths / *mathématique / *faire les maths.

It should be clear that there is nothing about mathematics *per se* that requires one preposition or another; while one can be interested *in* math, one can also rely *on* math or be afraid *of* math or look *to* math.

1.2 Noun-complement specificity

The second encouraging observation, used by Japkowicz and Wiebe (1991), is that many nouns may only be complements of certain prepositions. They assert that most nouns may only be used with particular prepositions, and that analogous nouns in different languages (English and French, for example) admit different propositions because the languages conceptualize those nouns differently. For example, in saying *on the bus* but *dans l'autobus* (literally "in the bus"), "English conceptualizes the bus as a *surface* that can *support* entities, by highlighting only its bottom platform, while French conceptualizes the bus as a *volume* that can *contain* entities, by highlighting its bottom surface, its sides, and its roof altogether." (Japkowicz, 1991)[2]

[2]Readers may wonder when prepositions are determined by a preceding word and when they are determined by a complement. We suspect that adverbial prepositional phrases, such as Jap-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

1.3 Local collocations

In testing one-sense-per-collocation for nouns, verbs, and adjectives, Yarowsky (1993) tested only local collocations. That is, he ignored the possibility that distant content words could give reliable information sense disambiguation. We do the same here, and with better cause. While it is somewhat plausible that senses of nouns, verbs, and adjectives—categories whose words are replete with meaning—could be inferred from distant context, such a situation seems unlikely for prepositions.

1.4 Potential problems

Given these sensible arguments for the Hypothesis, why bother testing it? Trujillo (1992) provides examples where the one-sense-per-collocation hypothesis fails. He presents an English sentence (3) with three plausible Spanish translations (4).

(3) She ran under the bridge.

(4) Corrió debajo / por debajo / hasta debajo del puente.

The first translation implies that she was running around under the bridge, the second that she ran on a path that went under the bridge and kept going, and the third that she ran up to a position under the bridge and stopped. We hope, however, that this example is of an infrequent special case, and can be overcome. Sentence (3) usually translates best with *por debajo*, and the same sentence with the verb *rested* translates best with *debajo de*.

Another possible problem is that individual speakers may use different prepositional phrases for essentially the same concept. While one speaker may use *on top of*, another may use *atop*, another *on*, and so on. Given these issues, additional testing is warranted.

2 Methods

To test the Hypothesis, we used the sentence-aligned Hansards of the 36th Parliament of Canada, a French-English bilingual corpus. (*Hansards*, 2001) Our

kowicz and Wiebe's locatives, are determined by their complements, while prepositional phrases governed by a preceding noun, verb, or adjective are determined by their governor.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

analysis takes four steps:

1. We preprocess the French sentences, changing *au* to *à le*, *aux* to *à les*, *du* to *de le*, *des* to *de les*, and *d'* to *de*.

2. We create a database, for each preposition in our list[3], with one record for each appearance in the training corpus (36.5 million words). Each record contains the surrounding four English words and the preposition's French translation.

3. We create a list, for each preposition, of English context words, along with the most frequent translation for the preposition given each context word.

4. We test our list's predictions on a held-out portion (4.5 million words) of the Hansard corpus. We also test the performance of a naïve translation algorithm for a baseline.

The first step is justified because in French a word like *au* is equivalent to the preposition *à* combined with the article *le*. Since combination with an article doesn't affect the sense of a preposition, this is fine to do.

In the second and fourth steps we need the correct translation of each English preposition. Since the Hansards are not word-aligned, this is difficult to do accurately. Consider the following sentence pair:

> I went to a library yesterday.
>
> Je suis allé à la bibliothèque hier.

We make the (rather large) assumption that if an English preposition is found n% of the way through a sentence, then its translation will be found n% of the way through its sentence as well. Since *to* is word number 2 (starting counting from 0) out of six words, and since the French sentence has seven words, our initial guess is that the translation of *to* is at position $2(7-1)/(6-1) \approx 2$. We find the word *allé* in that position, which is not an acceptable translation (taken from the *Collins-Robert French-English English-French Dictionary* (Atkins, 1996) of *to*. So, we look in the positions surrounding *allé*, and find *à*, an acceptable

[3]We use the prepositions *against*, *around*, *at*, *before*, *by*, *during*, *for*, *from*, *in*, *like*, *of*, *off*, *on*, *out*, *through*, *up*, and *with*. These were chosen because some are polysemous and some are monosemous, thereby providing a diverse set of test cases.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

translation, and halt. (In fact, we give up after searching four words ahead and behind.) This approach seems to work fairly well for the Hansard corpus, in large part because of the stilted, literal translations in it. Clearly, a word-aligned corpus would make better predictions here, particularly in instances where either English or French uses a multi-word preposition (e.g., *off of* or *autour de*).

In the fourth step, we get a baseline by measuring how a naïve word-for-word translation does on our held-out corpus. We simply translate each English preposition with its most common (or at least most canonical) French translation: *at* to *à*, *in* to *dans*, and so on.

3 Results

We tabulated results for each preposition. The following are typical of our results:

for		
Context	Precision	Accuracy
Two before	.9625	.6886
One before	.9564	.7027
One after	.9683	.6842
Two after	.8880	.6938
None	1.0000	.2857

of		
Context	Precision	Accuracy
Two before	.9817	.9169
One before	.9795	.9175
One after	.9826	.9172
Two after	.8993	.9155
None	1.0000	.9181

The precision is the number of times our translation list made a prediction divided by the number of prepositions encountered in the testing corpus. The accuracy is the number of times our translation list made a *correct* prediction divided by the number of times it made any prediction. Clearly, the improvements are much greater for some prepositions than for others. The results for all prepositions combined are:

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

Total Context	Precision	Accuracy
Two before	.9457	.7936
One before	.9394	.8084
One after	.9510	.8190
Two after	.8618	.8166
None	1.0000	.6140

The results show that surrounding context includes sufficient information to improve translation of most prepositions into French. In general, context words closer to the preposition give better information. We find this somewhat strange, since the word directly after a preposition is often an article, which should contribute little sense information.

Different prepositions give much different results, as shown in the sample data above. Why, in particular, are our results for *of* so poor compared with the baseline? Suppose we are testing the $+1$ position for *of*. If the word after *of* in our testing corpus is *Parliament*, for example, our system will guess whatever the most common translation of *of* before *Parliament* was during training. Since *of* almost always translates as *de*, the guessed translation will be *de* for almost any context word, and therefore our accuracy results will be much like the baseline accuracy for *de*.

4 Conclusion

All four context positions (two before, one before, one after, and two after the English preposition) were helpful in translation, giving clear benefits over the baseline. However, the best results came from the word immediately after the preposition.

There are several ways to improve on these results. First, a word-aligned corpus would erase the error introduced by our translation-guessing algorithm. Second, we might improve results by looking at more than one context word at a time, or by weighting the predictions based on some context words higher than others. However, even our limited results show that the one-sense-per-collocation hypothesis is often reliable for English prepositions.

It is possible that idiomatic usage occurs in the Hansard corpus enough to throw off the results. Therefore, it would be interesting to see the preposition-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

translation model applied to a number of different languages in parallel. At present, the lack of multilingual aligned corpora makes this infeasible, but should they become available, that experiment would have stronger results.

References

[Atk] Atkins, Beryl T., et al. *Collins-Robert French-English English-French Dictionary*, 3rd ed. New York: HarperCollins Publishers and Dictionnaires Le Robert, 1996.

[Han] Germann, Ulrich, ed. *Aligned* Hansards *of the 36th Parliament of Canada, Release 2001-1a*. 2001.

[Jap] Japkowicz, Nathalie, and Janyce Wiebe. "A System for Translating Locative Prepositions from English into French." *Meeting of the Association for Computational Linguistics* 1991: 153-160.

[Mart] Martinez, David, and Eneko Agirre. "One Sense per Collocation and Genre/Topic Variations." *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*: 207-215.

[Word] Princeton University, Cognitive Science Laboratory. *WordNet, version 1.7.1*.

[Tru] Trujillo, Arturo. "Spatial Lexicalization in the Translation of Prepositional Phrases." *30th Annual Meeting of the Association for Computational Linguistics* 1992: 306-308.

[Yar] Yarowsky, David. "One Sense per Collocation." *Proceedings, ARPA Workshop on Human Language Technology* 1993: 266-271.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 8–14
Computer Science Department, Swarthmore College

The Structure, Computational Segmentation, and Translation to English of German Nominal Compounds

Nori Heikkinen
Computer Science Department
Swarthmore College
Swarthmore, Pennsylvania, USA
nori@sccs.swarthmore.edu

Abstract

Among the most daunting words in German to a non-native speaker are the longest ones – those that near forty characters in length, and are comprised of many smaller words. A problem posed by these words is that of automatic segmentation into pieces in order to glean the meaning of the entire word. It is vital that these words be correctly parsed, interpreted, and translated as part of the ongoing work in multilingual computation linguistics, such as incorporation into German language spell-checkers.

Here, I describe the problem posed to morphological disambiguation in compounding languages, concentrating on the structure of German nominal compounds. Using regularities about their structure, I then devise a computational model for both determining linking elements of morphemes, and segmenting the words correctly. I discuss the inherent problems with such a model, and speculate about alternative approaches.

1 Note

This thesis was submitted as part of the requirements for the Bachelor of Arts in Linguistics and Computer Science. A full copy of this thesis can be obtained from the Linguistics Department at Swarthmore College.

References

Becker, Thomas. Compounding in German. *Rivista di Linguistica*, 4(1):5-36, 1992.

Brill, Eric. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21(4):543-565, 1995.

Fleischer, Wolfgang. *Wortbildung der deutschen Gegenwartssprache.* VEB Bibliographisches Institut, Leipzig, 1969.

Jurafsky, Daniel and Martin, James H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, New Jersey, 2000.

Ngai, Grace and Florian, Radu. fnTBL. http://nlp.cs.jhu.edu/~rflorian/fntbl/.

Ngai, Grace and Florian, Radu. Transformation-Based Learning in the Fast Lane. In *Proceedings of NAACL-2001*, pages 40–47, 2001.

Ortner, Hanspeter and Ortner, Lorelies. *Zur Theorie und Praxis der Kompositaforschung*. Gunter Narr Verlag, Tübingen, 1984.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, page 15
Computer Science Department, Swarthmore College

Using Semantic Information from Neural Networks to Detect Context-Sensitive Spelling Errors

Julie Corder
Swarthmore College CS97
Spring 2003

Abstract

This paper proposes a means of using the internal representations of an artificial neural network to represent the semantic contexts in which a word can appear. Once the network has been trained, its hidden layer activations are recorded as a representation of the average context in which a word can appear. This context can then be compared to the contexts in which a word appears in novel text to detect context-sensitive spelling errors. While no significant results are found in the trials described here, several modifications of the system are proposed that might prove promising in future work.

Introduction

Context sensitive spelling correction is the process of identifying words in written text that are spelled correctly but are used in the wrong context. Kukich (1992) discusses various studies that show that between 25% and 40% of spelling errors in typed text result in legal words. This category of spelling errors includes word pairs that are easily mistyped (e.g. "form" and "from"), homophones (e.g. "they're", "their" and "there") and words with similar usages (e.g. "affect" and "effect"). Because all of these errors result in words that are valid, an approach that relies on just a dictionary look-up process will not detect them as spelling errors. Further, Atwell and Elliott [1987] found that 20% to 38% of errors in texts from a variety of sources resulted in valid words that did not result in local syntactic errors. Since dictionary- and syntax-based approaches are not able to detect most context-sensitive spelling errors, semantic clues must be taken into account to determine if the correct word is being used in a given context.

Previous Work

Instead of relying on a comparison to a dictionary of valid words, researchers interested in context sensitive spelling correction must find ways to represent the semantic context in which a word occurs to determine if it is spelled correctly. This approach may be as simple as calculating statistical probabilities of words appearing in certain n-grams, or they may involve greater syntactic and semantic analysis of a corpus. Jones and Martin [1997] report accuracy rates of 56% to 94% for various sets of confusable words using Latent Semantic Analysis. Granger [1983], Ramshaw [1989] and others have used expectation-based techniques. Their systems maintain a list of words that they expect to see next in

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

a corpus based on semantic, syntactic, and pragmatic information in the text. If the next word that appears is not on the list of words that were expected, it is marked as a spelling error. In this way, the systems can both detect spelling errors and learn the meaning of new words (by comparing to the meanings of the expected words when a novel word appears).

In all of these cases, though, the researcher must specify the level of information that is relevant to the task. Jones and Martin [1997], for example, specifically tell their system to look at a window of seven words before or after the word in question to build the initial matrices for their analysis. They rely on the researcher to determine how big the window should be. Further, since they look at words before and after the word in question, their method is only useful with complete texts.

These limitations can, perhaps, be avoided by a system that incorporates a neural network. Artificial neural networks (ANNs) are well-suited to a variety of NLP tasks; they can develop their own characterization of which features of a problem are most significant. In addition, simple recurrent networks can store a copy of their previous hidden layer activations. In this way, they are able to build up abstract representations of patterns that occur throughout time [Elman et al. 1996]. Thus, a simple recurrent network should be able to develop an abstract representation of the current context by looking at its internal representation of any number of the words that come before the current word. Given this context, an expectation-based system should be able to predict which words should be able to come next in the text. If the actual next word is not on this list, it should be marked as a spelling error. Further, this system can be combined with a shortest path algorithm to select a word from the list as the correct word, as Wang and Jean [1993] did to correct spelling errors resulting from character merging during OCR. Because this method does not look at future words, it would be useful in applications like word processing systems, where the most recently entered word can be examined for a potential context-sensitive spelling error before more text is entered.

Methods

One of the most limiting aspects of neural networks is the fact that the time needed to train them increases rapidly as the size of the network increases. To test my method, it was necessary to drastically limit the size of the input representation for the network. Consequently, a very small vocabulary represented in localist binary vectors was used to encode the corpus. Vocabulary words were represented by vectors whose length was equal to the number of words in the vocabulary. For a vocabulary of twenty-five words, then, only twenty-five bits were needed to represent any given word. Each vector consisted of exactly one bit that was "on," and the rest of the bits were set to zero.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

Training and testing data came from a part of speech tagged Wall Street Journal corpus. Several categories of words were collapsed into a single “pseudoword” based on part of speech as a means of decreasing the vocabulary size. In particular, the part of speech categories of NN, NNP, NNS, JJ, VBD, VBN, VBZ, DT, and MD were only recorded in the training data by their part of speech class. Further, all punctuation marks were collapsed into the single pseudoword *PUNCT*. Finally, all numerals and number words were changed to the pseudo word *CD* since each number is relatively uncommon in the training text but numbers can usually appear in the same positions in texts. The remaining words, including most function words, were not collapsed into pseudowords at all.

Next, the 25 word vocabulary for this project was selected. The three words to be looked at as possible context-senstive spelling errors were automatically added to the vocabulary. In this trial, that meant that *to*, *too* and *CD* (which contains *two*) were added to the vocabulary. The trianing corpus was then examined, and the 22 most common words were added to the vocabulary. Sentences in the corpus that contained words that were not part of the vocabulary were deleted since they could not be encoded. To ensure that enough data was available about the three target words, sentences that did not include one of those words were also deleted; essentially, a new training and testing corpus was generated by accepting the first sentence that could be encoded that included *to*, then the next sentence that included *too*, and then the next sentence that included *cd* until fifty examples of each had been found. This corpus was encoded as described above and passed to a neural network for training.

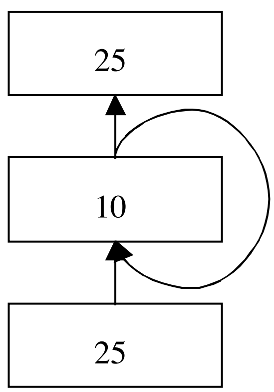


Figure 1: Architecture of recurrent neural network with 25 input and output nodes and a 10-unit hidden layer.

A simple recurrent network was trained on the first half of the encoded corpus. The network had 25 input nodes, corresponding to the 25-bit vector representations of words in the corpus. It also had a 10 node hidden layer and a 25 output nodes. The hidden nodes fed into the output as well as back into the hidden layer. The overall architecture of the network is shown in Figure 1. At each time step, the network’s task was to predict the word that would come next.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

The network was trained on the entire sequence for 50 epochs using back-propagation.

Once training was completed, the network's representation of the context in which each word appeared was of more interest than the network's prediction of the next work in the corpus. This context representation is stored in the activations of the network's hidden nodes. One final pass through the entire corpus was completed with learning turned off, and the activations of the hidden nodes were recorded at each time step. The hidden layer is the place where the network can establish its own categorization of inputs before having to generate the correct output, so looking at the hidden layer activations gives an illustration of the way that the network is categorizing words internally. The average activation of the hidden layer nodes right before a word was presented was recorded for each word in the training corpus. Because the hidden layer represented the network's representation of the semantic context in which a word would be expected to appear, this vector will be referred to as the "expectation vector" for a given word.

The expectation vectors for all of the words in the vocabulary can be mapped in n-dimensional space; nodes that are closer together in this space can appear in similar semantic contexts, while nodes that are further apart in this space appear in more drastically different semantic contexts.

Context-sensitive spelling errors result, in general, when a word appears in the wrong semantic context. That is, "form" is a spelling error in the sentence "The letter arrived form Cleveland" because it does not appear in a valid location in the sentence (and not because it is an invalid word in English). To detect context-sensitive spelling errors, then, one need only compare the hidden layer activations representing the current context of a network to the average hidden layer activations when the next word is about to appear. If the two are substantially different, the word should be marked as a spelling error, even if it is a valid word in a dictionary.

For each word in the testing part of the corpus, the euclidean distance between the expected context (that is, the average context in which the word appeared in the training corpus) and the actual context (that is, the current hidden layer activations) is calculated. If the current word is one of the target words, then the current hidden layer activation is also compared to each of the expectation vectors of the other target vectors. A large (order of magnitude) difference between the distances for words found in the corpus and the alternative pairings of target words would indicate that the use of the wrong target word somewhere in a novel corpus could be identified by examining the euclidean distance between its expectation vector and the current hidden layer activation.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

Results

Unfortunately, there did not seem to be a clear distinction between expectation vectors and the actual hidden layer contexts for different target words. The average euclidean distance between the network's hidden layer activations and the expectation vector for the correct next word was 0.921. The average euclidean distance between the hidden layer activations and the expectation vectors of the other (wrong) target words' expectation vectors was 0.975 (Figure 2). The distance values varied greatly from one word to the next; the standard deviation for both sets of distances was over 0.19, so the difference between the two is clearly not significant.

	Mean distance	Standard Deviation
Correct Expectation Vector	0.921	0.191
Wrong Expectation Vector	0.975	0.196

Figure 2: Average Distance between actual hidden layer activations and the average context for the same (left bar) or different (right bar) target words. Standard deviation is .191 for same-target and .196 for different-target words.

This is a disappointing result. Ideally, the distance to the correct expecation vectors would be significantly smaller than the distance to the wrong expectation vectors. Then the distance between the current hidden layer activation, for example, and the next word typed in an application could be used to predict if there was a context-sensitive spelling error in the current word in the document. Latent semantic analysis could then be used to suggest words whose expecation vectors more closely match the current hidden layer activation. Without a clear distinction between correct and incorrect target words, though, no further analysis can be conducted in terms of application of this process to an actual instance of context sensitive spelling correction. These results do not, however, mean that there is definitely not a way to use an approach of this sort; the following section discusses some of the limitations inherent in this particular study and ways that they might be addressed in future work in this area.

Discussion

One of the most substantial limitations of this project was the small vocabulary size. By collapsing full part of speech categories into a single pseudoword, much of the semantic content that might originally have been available in the text was lost. While this simplified the search space, it also may have resulted in a loss of information that would have been particularly useful to the network in its task.

The solution to this problem is not as simple as just increasing the number of words in the vocabulary and the corresponding number of nodes in the

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

representation of each word. For very large networks, the cost of backpropogating error can be prohibitably expensive. Consequently, a localist representation quickly becomes unreasonable as vocabulary size increases.

One possible way to address this problem is through a more distributed representation. Words can be represented, for example, as a binary representation of their unigrams and bigrams. The first twenty-six nodes in the representation vector correspond to the unigrams that may be present in a word. If the current word contains a unigram, then the corresponding node in the input vector is activated. The rest of the nodes correspond to potential bigrams. Bigrams may contain any combination of the alphabetic letters, an end of word marker, and a beginning of word marker. In total, this results in an input vector whose length is 754. For example, in the representation of "two," the nodes representing "t", "w", "o", "_t", "tw", "wo" and "o_" would have values of one, while all other nodes would have values of zero. This representation is drastically larger than the one used for the trials discussed in this paper. It has the advantage, though, of scaling well for extremely large vocabularies. In fact, for a corpus with a vocabulary of more than 754 words, this representation will actually result in a smaller network than will a localist representation. Since 754 words is not a very large vocabulary size for a real-life corpus, a representation of this sort seems essential to further study in this area.

Another possibility is that error was inherent in the process of averaging the context vectors for each word. If the contexts for a given word are clustered in two or more drastically different locations, then averaging them together results in a single vector that is not really representative of any of them. Once the contexts for each step through the training corpus have been gathered, it may be beneficial to conduct some sort of clustering analysis on the vectors. This could avoid the creation of artificial and misrepresentative average vectors that are currently used as the basis for comparison during testing. Unfortunately, only the average contexts for each word were recorded in this experiment, so the presence of this sort of error cannot be confirmed, but it seems like a likely problem that is worth exploring before future work in this area is conducted.

The final possibility is that better results could be found by adjusting the learning parameters of the neural network itself. The epsilon, tolerance and momentum values can all be tweaked. In addition, changes to the number of hidden nodes or the number of training epochs might provide interesting enhancements in the overall system performance. Without any reliable means of predicting what sorts of adjustments would be likely to be beneficial, it was not feasible to test adjustments of these factors in this trial; varying these parameters on a smaller-scale problem would not give a useful indication of how they would affect the larger network or a longer training process, and training a large network takes long enough that running multiple trials of the entire experiment was not possible.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

References

Atwell, E. and S. Elliot. 1987. “Dealing with Ill-formed English Text (Chapter 10). In *The Computational Analysis of English: A Corpus-Based Approach*. R. Garside, G. Leach, G. Sampson, Ed. Longman, Inc. New York.

Elman, Jeffrey L., Elizabeth A. Bates, Mark H. Johnson, Annette Karmiloff-Smith, Domenico Parisi, and Kim Plunkett. *Rethinking Innateness: A Connectionist Perspective on Development*. 1996: Massachusetts Institute of Technology

Granger, R.H. 1983. “The NOMAD system: Expectation-based detection and correction of errors during understanding of syntactically and semantically ill-formed text.” *American Journal of Computational Linguistics 9*, 3-4 (July-Dec.), 188-196.

Jones, Michael P. and James H. Martin. “Contextual Spelling Correction using Latent Semantic Analysis.” 1997.

Kukich, Karen. "Techniques for Automatically Correcting Words in Text." ACM Computing Surveys Vol. 24, No. 4, December 1992.

Ramshaw, L. A. 1989. “Pragmatic knowledge for resolving ill-formedness.” Tech. Rep. No. 89-18, BBN, Cambridge, Mass.

Wang, Jin and Jack Jean. "Segmentation of Merged Characters by Neural Networks and Shortest-Path." ACM-SAC 1993.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 16–22
Computer Science Department, Swarthmore College

Part Of Speech Tagging Using A Hybrid System

Sean Finney
Swarthmore College
finney@cs.swarthmore.edu

Mark Angelillo
Swarthmore College
mark@cs.swarthmore.edu

Abstract

A procedure is proposed for tagging part of speech using a hybrid system that consists of a statistical based rule finder and a genetic algorithm which decides how to use those rules. This procedure will try to improve upon an already very good method of part of speech tagging.

1 Introduction

The tagging of corpora is an important issue that has been addressed frequently in computational linguistics for different types of tags. Transformation-Based Learning (Brill, 1995) is a successful statistical method of tagging a corpus. It has been useful in part of speech tagging, word sense disambiguation, and parsing among other things. This paper describes a hybrid method for tagging part of speech. The method employs an implementation of Brill's Transformation-Based Learning (TBL) as the statistical part of the system, and a Genetic Algorithm which tries to modify the transformation ruleset in a way that will produce better results.

Part of speech taggers are very useful in modern Natural Language Processing, and have potential applications in machine translation, speech recognition, and information retrieval. They are usually used as a first step on a block of text, producing a tagged corpus that can then be worked with. Of course, the better the tagger is, the more accurate overall results will be. While TBL is a very successful part of speech tagger, it does still create some errors. It might be possible to tag a corpus perfectly, and it might be possible to use TBL, slightly modified, to acheive that goal.

During Transformation-Based Learning, transformation rules are learned which will correct errors in an incorrectly tagged corpus. The incorrectly tagged corpus is compared to the truth in order to learn these rules. Once the rules are learned, they are applied in an order determined by a greedy search. The rules are applied to the corpus to see how many errors they correct. The rule that corrects the most errors is chosen as the next rule in the ruleset. This process is inherently shortsighted.

cond 1	transform x
cond 2	transform y
cond 3	transform x
cond 1	transform z
cond 4	transform w
cond 5	transform x
cond 3	transform z
cond 2	transform x

Figure 1: A Sample Ruleset for TBL

The issue here is the greedy search. As rules are applied, the number of errors corrected in the text is always the largest. However, the transformation rules do tend to create errors, even as they are fixing errors. This process assumes that those errors are unimportant, or that they will be corrected later on in the ruleset. This assumption is made by TBL, and we are hoping to fix this potential problem. With a careful reordering of the rules, it might be possible to create a tagged corpus without any errors.

2 Related Work

Other workers in the field have produced non-statistical systems to solve common NLP problems. There appears

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

to be a number of methods that produce varying results. The Net-Tagger system (Schmid, 1994) used a neural network as its backbone, and was able to perform as well as a trigram-based tagger.

A group working on a hybrid neural network tagger for Thai (et al., 2000) was able to get very good results on a relatively small corpus. They explain that while it is easy to find corpora for English, languages like Thai are less likely to have corpora on the order of 1,000,000 words. The hybrid rule-based and neural network system they used was able to reach an accuracy of 95.5(22,311 ambiguous word) corpus. It seems that the strengths of rule-based statistical and linguistic methods work to counteract the inadequacies of a system like a neural network, and vice versa. It seems logical to combine the learning capabilities of an AI system with the context based rule creation capabilities of a statistical system.

3 Genetic Algorithms

The Genetic Algorithm (GA) takes its cue directly from modern genetic theory. This method entails taking a problem and representing it as a set of individual chromosomes in a population. Each chromosome is a string of bits. The GA will go through many generations of chromosomes. One generation consists of a fitness selection to decide which chromosomes can reproduce, reproduction itself, and finally a chance for post reproduction defects. The GA generates successive populations of chromosomes by taking two parent chromosomes selected by a monte carlo approach and applying the basic genetic reproduction function to them. The chromosomes are ranked by a fitness function specific to the problem. The monte carlo approach assures that the two best chromosomes are not always chosen.

The reproduction function is *crossover*, in which two chromosome parents are sliced into two pieces (See Figure 5 in the appendix). The pieces each take one of the pieces from the other parent. The resulting two chromosome children have part of each of the parents[1].

After reproduction, there is a chance that *mutation* will occur, in which some part of the child chromosome is randomly changed (see Figure 6 in the appendix).

When selection, reproduction, and mutation have finished, the generation is completed and the process starts again. The genetic algorithm is a good way to try many probable solutions to a problem. After a few generations of a genetic search, it is likely that the solution generated will already be very good.

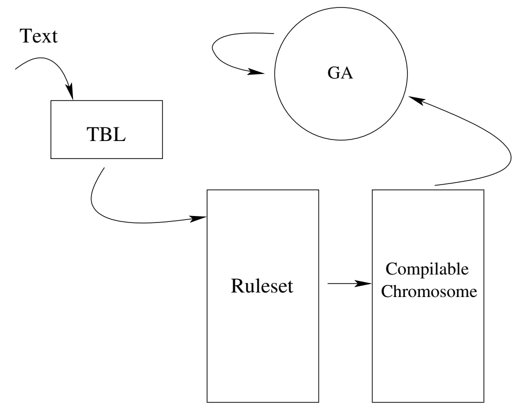


Figure 2: A Sample Flowchart for Our System

4 Representing the Problem

Given that we are trying to produce the optimal set of rules, and the order in which to use them, our chromosomes are best represented as a set of rules in order. The first parent from which all successive generations grow is collected directly from TBL. We let TBL find the rules that it finds useful, trim that set of rules to a satisfactory length, and run that through as the first parent of our GA.

One issue that we ran across was how we would represent each TBL rule as a bitstring. A rule in TBL consists of a series of predicates acting on the words before and after the word in question, and a trigger that changes the tagged part of speech on the word if all of the predicates are met. A predicate can be either a matching of a part of speech, or one of a number of string matching rules. The first of these is a pre or postfix matcher. The second adds a pre or postfix and checks to see if the resulting string is a separate word. The third removes a pre or postfix, and checks if the resulting string is a word. The fourth checks to see if a certain string is contained within the word. The fifth checks if the word in question appears to the left or right of a specified word in a long list of bigrams.

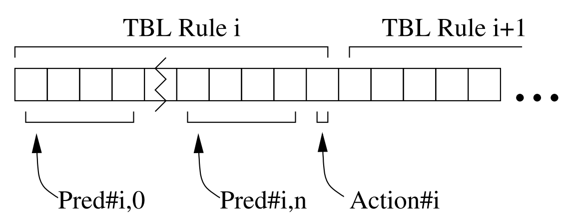


Figure 3: A sample TBL ruleset embbedded in a chromosome

In our actual implementation, every C structure con-

[1]an exception is where the crossover consists of an entire chromosome from one parent and none from the other

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

index	lexical meaning	contextual meaning
1	type of predicate	type of predicate
2	extra info for predicate type	whether or not the predicate is a range
3	not used	if a range, the start and stopping bounds
4	index into a table of observed strings/POS's	index into a table of observed strings/POS's

Table 1: Meaning of chromosome values to a predicate

tained member variables that could be easily converted to and from a genetic bitstring. In our population of individual chromosomes, each chromosome represented a lexical and contextual ruleset, such that any individual could be output via a helper function into the files necessary for fnTBL to run.

Each chromosome is first divided in half, with one half representing the lexical portion (prefixes, suffixes, et c.) of the fnTBL tagging rules, and the other representing the contextual (ngram) portion of the rules.

Both of these halves are then divided into several smaller parts of equal length, each part representing a single rule (see Figure 3). If the reader will recall, a rule consists of a series of predicates and a resulting action to execute if the predicates are all matching. The action is an integer which is computed via a modulo operation against the number of possible actions, such that it will always evaluate to the value of an enumerated part of speech.

A predicate is in fact one of two kinds of data structures, depending on whether it is found in the lexical or in the contextual portion of the gene sequence. They are of the same length (16 bytes), but but each 4 byte sequence may have a different meaning (See Table 1).

Our genetic ”fitness” heuristic takes a chromosome, and converts it into a structure that conforms to the heirarchy discussed above. this structure is then used to output the necessary data files, which are then used to determine the error rate, which is then returned as the fitness to the genetic algorithm.

5 The Process

Our GA process works in series with TBL. We start with a corpus which gets passed through TBL, but only far enough to output the ruleset to a file. We then run a script that generates a compilable chromosome representation from the TBL ruleset file. That chromosome representation is then passed to a GA program written in C by Lisa Meeden of the Swarthmore College Computer Science Department, and severely hacked by us. In the original implementation, a chromome ”bitstring” was an array of integers, where each integer was set to either 1 or 0. The author hopes that the reader can understand how this will become prohibitively large for any kind of complex representation. We therefore modified Meeden's code to have each integer be a random value, and therefore saved the memory resources required by this program by a factor of 32^2.

The GA then loops on itself for as many generations as we deem necessary. The final ruleset chromosome is then converted back into a file in the form that TBL likes, and the results are also written to a file.

The TBL implementation we used is Florian and Ngai's fnTBL(Florian and Ngai, 2001). We worked hard to integrate our scripts and code with their implementation of TBL, which trying to keep their code as intact as possible.

As it is implemented, fnTBL has two parts, the first being a script that trains the ruleset, and the second being the ruleset applicator. It was fortunate for us that fnTBL was broken up in this way, because we were able to take the learned ruleset and run it through our GA, producing another ruleset which could then be given to the fnTBL rule applicator. The results are placed in another file which can then be tested for accuracy.

6 Annealing the Mutation Rate

In order to make sure that we would get enough variation in our populations , we used a bit of the methods of annealing on our mutation rate. For the first few generations of running, the mutation rate is very high so that the children will show a lot of difference from the parents. Over time, we bring this mutation rate down so we converge on a good solution without jumping around as much. We feel that this method further reduces the risk of the solution being the greedy path to the goal.

One potential point of contention arises here. It is impossible for us to guess what the best place to begin our mutation rate is. We also need to pick the decay function, and we cannot choose whether a linear or logarithmic or exponential decay function would be better without trying them all first. We did speculate that the rate should start high and follow an exponential decay function. We want the chaos in the system to start high and come down fast, so the system has a lot of time to work out all of the chaos that was introduced in the first few generations of the GA. We chose to try a linear decay first, with a high starting mutation rate. The idea here is that much chaos over a long period of time will produce a ruleset that is radically different from the parent.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

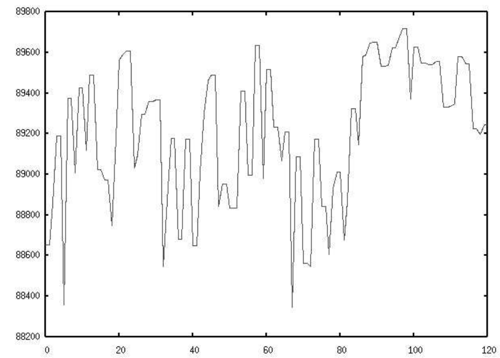


Figure 4: If only we had a month of cpu time...

7 Results and Discussion

After 100 generations without an annealed mutation rate, we had a success rate of 84.45%. When compared to fnTBL's 94.57% success rate, we actually took a loss on the performance of the system. This does make sense, seeing as how the mutation rate was very low, about 0.001. Essentially the only process being run was crossover, and when a set of rules is crossed over, more that likely the better rules at the beginning of the ruleset will be moved to the end. It certainly does not make sense to run the less successful rules first.

We experienced much better results once we annealed the mutation rate and ran the GA for 100 generations. Our success rate on this run was 89.96better results. One problem with this approach was the way the mutation rate was annealed, a strictly linear decay. We hope with an exponential decay rate, our results will be even better.

Another run with an annealed mutation rate used a starting mutation rate of .1 instead of the .8 used in the result above. 100 generations produced a success rate of 89.7even reached the level acheived already by TBL. The inherent nature of the GA approach is to need many generations, and a large population size. Unfortuantely, these runs both took close to 8 hours to complete. We believe that when taken further, and with more time and generations, the hybrid approach could allow for a continued increase across the board in NLP problems. With our minimal settings, the system took 8 hours to run. Increasing the population size and the generations would increase the running time significantly on today's computers. We hope that some day this work can be tested on a parallel or distributed system powerful enough to handle the large memory and processor requirements.

[2]there are 32 bits in an integer, so to represent an integer as a "bitstring" in Meeden's code would require 32 real integers!

8 Future Work

The original goal of the project was to try to improve TBL using artificially intelligent techniques. We feel that we have adhered to that goal. We are of the mind that language and intelligence are very closely related, and that in order for us to create systems that can use language effectively, we will first need to have a better model of intelligence. While the tools we have today in the AI field are not powerful enough to be considered fully intelligent, they can be used to approximate what we might be able to accomplish in years to come.

Work from others such as (Yang, 1999) tries to concentrate on the modelling of development, and argues that a child's developing language pattern cannot be described by any adult's grammar. Yang also states that development is a gradual process, and cannot be described by any abrupt or binary changes. The same sort of developmental thinking is encompassed by the AI approach, with a gradual (albeit hopefully faster) learning process based on human intelligence itself.

As far as our project goes, there are many variables that we did not have the time to explore. Firstly, our fitness function for the GA was simply based upon the percentage correct. Another possible idea would be to have a fitness function which takes the number of errors, either present or created by the ruleset, into account. One way to do this would be to return a fitness of the number of correctly tagged words minus a fraction of the errors. This would discourage the GA from having and causing errors.

There are also variables associated with TBL for which we chose values that made sense to us. The number of rules in a ruleset and the number of predicates in a rule could both be modified by a weight learning program like a neural network which would watch over the entire process. This way, we would have a more focused idea of what the optimal size of these variables would be.

9 Acknowledgements

We would like to acknowledge the efforts of our professor Rich Wicentowski of the Swarthmore College CS Dept. His encouragement and feedback were invaluable to our project.

Thanks also go out to Lisa Meeden of the Swarthmore College CS Dept. for her genetic algorithm code. Our task would have been much more complicated had we needed to impliment a genetic search ourselves.

We would also like to extend our gratitude to Florian and Ngai for their contribution to our project with fnTBL. Without regard to the countless days, nights, and following mornings spent trying to get fnTBL to work for us, the task at hand would have been prohibitively more complicated had we needed to implement TBL ourselves.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

Thanks!

Finally we would like to thank the entire robot lab crew for being friends comrades, and cohorts throughout the entire process. Konane!

References

Eric Brill. 1995. *Transformation-based Error-driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging*.

Qing Ma et al. 2000. *Hybrid Neuro and Rule-Based Part of Speech Taggers*. Communication Research Laboratory, Iwaoka, Japan.

Radu Florian and Grace Ngai. 2001. *Fast Transofmration-Based Learning Toolkit*. Johns Hopkins University.

Helmut Schmid. 1994. *Part of Speech Tagging with Neural Networks*. Institute for Computational Linguistics, Stuttgart, Germany.

Charles D. Yang. 1999. *A Selectionist Theory of Language Acquisition*. Massachusetts Institute of Technology, Cambridge, MA.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

A Appendix

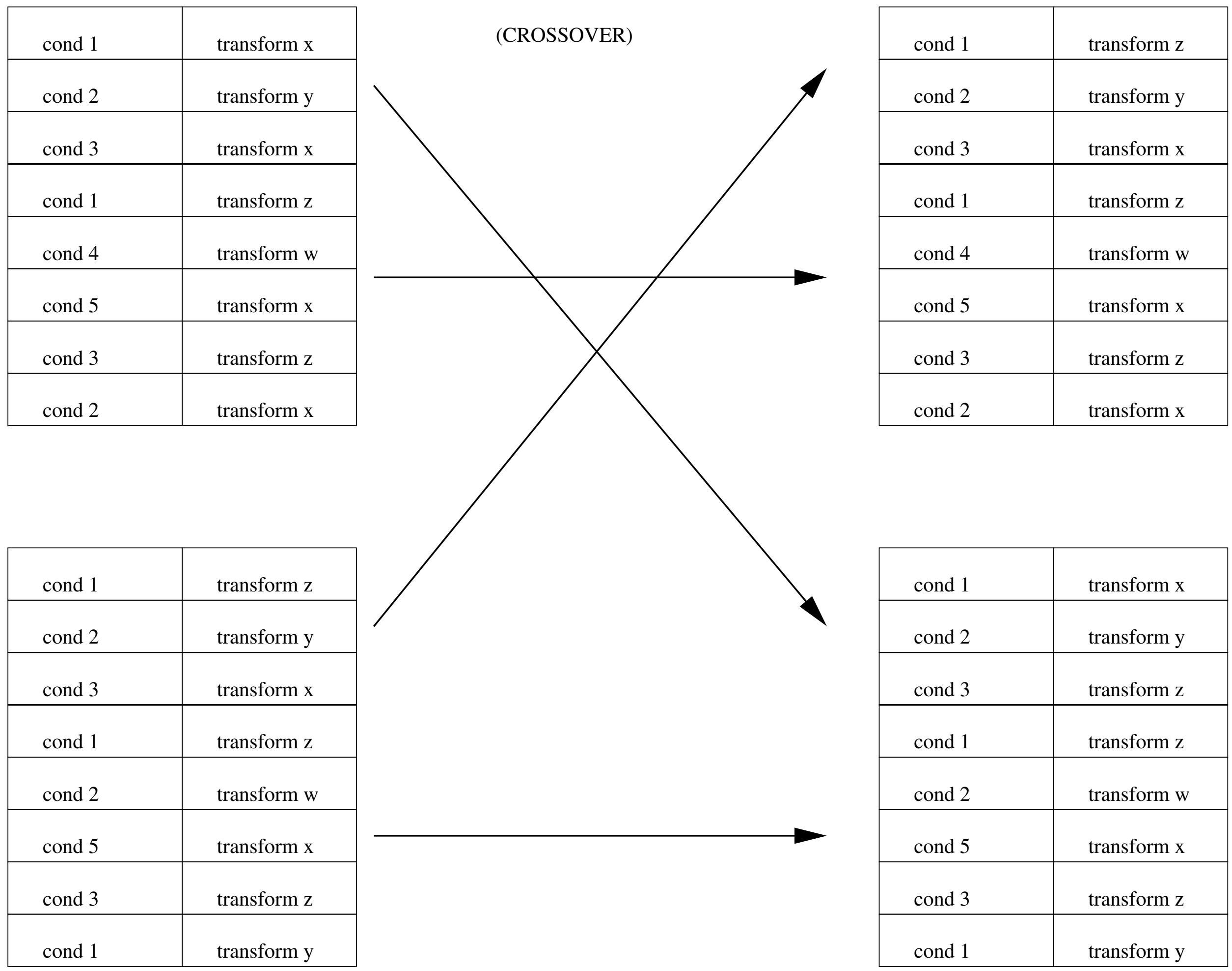


Figure 5: An example of a crossover operation

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

cond 1	transform x
cond 2	transform y
cond 3	transform x
cond 1	transform z
cond 4	transform w
cond 5	transform x
cond 3	transform z
cond 2	transform x

(MUTATION)

cond 1	**transform z**
cond 2	transform y
cond 3	transform x
cond 1	transform z
cond 2	transform w
cond 5	transform x
cond 3	transform z
cond 1	**transform y**

Figure 6: An example of a mutation operation

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 23–29
Computer Science Department, Swarthmore College

Disambiguating between ‘wa’ and ‘ga’ in Japanese

Yoshihiro Komori
500 College Avenue
ykomori1@swarthmore.edu

Abstract

This paper attempts to distinguish when to use ‘wa’ and ‘ga’ in Japanese. The problem is treated as one of word sense disambiguation, regarding both ‘wa’ and ‘ga’ as a prototype particle that indicates the subject of the sentence. Various statistical and linguistic techniques are employed to disambiguate the sense, such as ngram and syntactic analysis. The program scored 100% recall rate and 83.8% using the syntactic model.

1 Introduction

The distinction between ‘wa’ and ‘ga’ in Japanese has been notoriously hard for both Japanese linguists and those who attempt to learn Japanese as a foreign language. Both ‘wa’ and ‘ga’ are particles to indicate the subject but with slightly different connotations. For example, the English sentence

I am Yoshi.

can be translated to Japanese as

watashi	***wa***	*yoshi*	*desu.*
I	(null)	Yoshi	am.

or

watashi	***ga***	*yoshi*	*desu.*
I	(null)	Yoshi	am.

Whether we should use ‘wa’ or ‘ga’ cannot be determined locally in many cases, such as in this example. Those two Japanese sentences are syntactically valid and commonly used. To determine which particle to use, we need first determine the semantics of the sentence from its context.

There are several areas where having a machine that distinguishes ‘wa’ and ‘ga’ can be helpful. Those include translation to Japanese from various languages (given a sentence in a foreign language, should we use ‘wa’ or ‘ga’), Japanese sentence generation (given what we want to say, should we use ‘wa’ or ‘ga’), and Japanese linguistic theory (what are the conditions that require the use of ‘wa’ or ‘ga’?).

2 Linguistic Theory

Karita Shuji, a Japanese linguist, summarized the works of recent linguists on the usage of ‘wa’ and ‘ga’. According to Karita, the usage of ‘wa’ and ‘ga’ can be categorized in the following manner:

2.1 Substitution by ’no’

In some cases ‘wa’ can be substituted by ’no’ without changing the meaning of the sentence, but ‘ga’ can never be. This is due to the fact that the noun preceded by ‘wa’ does not have to be the actor in the sentence.

2.2 Novelty of the subject

One sense of ‘wa’ and of ‘ga’ is dependent on whether the subject is novel to the listener. While ‘wa’ is used when the subject is novel, ‘ga’ is used when the subject is known, previously mentioned or implied. Even when the subject is a novel topic in the discourse, ‘ga’ might be used if the information is implied by information outside of the discourse. An instance of such cases is:

watashi	***ga***	*senjitsu*	*email*	*shita*
I	(null)	the other day	email	did
gakusei	*desu.*			
student	am.			

(I am the student that emailed you the other day.)

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 30–34
Computer Science Department, Swarthmore College

The fact of the subject's emailing the listener makes the subject a familiar topic even though it may be the first time it is introduced in the discourse.

2.3 Description vs Judgment

Karita argues that in some cases 'ga' is used for describing a phenomenon, while 'wa' is used for expressing a judgment. For example,

ame	***ga***	*futte*	*iru*
rain	(null)	raining	is

(Rain is falling.) (*description*)

are	***ga***	*ume*	*da.*
that	(null)	plum	is.

(That is a plum.) (*judgment*)

In the first case we use 'ga', and in the second case 'wa'. The difference, however, is slight and is hard even for a native speaker to distinguish.

2.4 Sentence Structure

We use 'wa' if the subject specified is the subject of the entire sentence, and we use 'ga' if the subject is only the subject of a phrase in the sentence. so, for example:

tori	***ga***	*tobu*	*toki*	*ni*	*wa*
bird	(null)	fly	when	(null)	(null)
kuuki	*ga*	*ugoku.*			
air	(null)	move.			

(When a bird flies, the air moves.)

tori	***wa***	*tobu*	*toki*	*ni*	
bird	(null)	fly	when	(null)	
hane	*wo*	*konna*	*fuu*	*ni*	*suru.*
wing	(null)	like this	way	(null)	do.

(A bird moves its wings like this when it flies.)

The bird in the first sentence is a subject only in a phrase, where the second bird is the subject of the entire sentence. Note that being the subject of an entire sentence is not a necessary condition for using 'wa'. However, being a subject inside a phrase is a necessary condition for using 'ga'. Therefore, if 'wa' or 'ga' is to be used inside a phrase, 'ga' must be used all the time.

2.5 Contrast and Exclusion

Karita argues that we use 'wa' to indicate contrast and 'ga' to indicate exclusion. Two exampler sentences:

ame	***wa***	*futte*	*iru*	*ga*
rain	(null)	fall	(-ing)	but
yuki	***wa***	*futte*	*inai.*	
snow	(null)	fall	(not -ing)	

(Rain is falling but snow isn't)

yoshi	***ga***	*seito*	*desu*
Yoshi	(null)	student	is.

(Yoshi is the student.)

In the first sentence, 'wa' is used to express the contrast between 'rain' and 'snow,' while in the second sentence 'ga' is used to imply that Yoshi is the only student in the context.

2.6 Specimen

Two sentences:

chou	***wa***	*mushi*	*da.*
butterfly	(null)	insect	is.

(A butterfly is an insect.)

kore	ga	*kimino*	*chou*	*da.*
this	(null)	your	butterfly	is.

(This is your butterfly.)

In the first sentence 'wa' is used, and 'ga' is used for the second case. The difference between the two cases is that in the first sentence, a butterfly is a specimen of the class insect, where in the second case 'this' and 'butterfly' are in the same class.

2.7 Implication for the project

Karita's linguistic analysis on the usage of 'wa' and 'ga' has two implications for this project. First, these characterizations imply that the usage of 'wa' and 'ga' are determined by a mixture of syntactic and contextual information. Therefore, in order to capture the usage of 'wa' and 'ga', both syntactic and contextual approach must be employed. Second, from these characterization one could argue that both 'wa' and 'ga' have several different senses. This implies that in order to achieve the competence of a native speaker, the problem has to be understood as disambiguating the sense of the prototype subject indicator

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 30–34
Computer Science Department, Swarthmore College

into a dozen senses. However, such a project would require a huge tagged corpus where each instance of 'wa' and 'ga' is disambiguated from several senses. Employing humans to hand-tag such a corpus would be expensive. Thus we will attempt to disambiguate the prototype subject indicator into only two senses, 'wa' and 'ga'.

3 Related Works

The task of word sense disambiguation has been tackled by many NLP researchers, such as Yarowsky (1992, 1993 and 1995). However, the two key assumptions often made in the task of word sense disambiguation do not hold in this particular task. The assumption of 'one sense per discourse' clearly does not hold here because both 'wa' and 'ga' occur in discourses with any topic and style. The other key assumption, 'one sense per collocation,' does not hold here as well as it does in other places, since both 'wa' and 'ga' can follow any noun. On the other hand, the idea of 'one sense per collocation' can be useful if we take the collocation to be mainly syntactic and use it loosely to aid other algorithms.

4 Task Description

The input to this program consists of Japanese copra tagged with the POS. The tags are produced by a GPL engine "mecab" developed by Kudo Taku, which claims to achieve 90% accuracy. At the preprocessing stage we replace every instance of 'wa' and 'ga' that are particles to indicate the subject, determined by the POS, with a prototype particle *prt*. The output of the program is 'wa' or 'ga' for each instance of *prt*. The performance is measured by the ratio of correct determination of 'wa' and 'ga'.

The training corpus consists of three novels by a Japanese novelist Dazai Osamu, *NingenShikkaku*, *Joseito* and *Shayou*. The testing corpus consists of two other short stories by the same author, *Haha* and *Hashire Merosu*. The size of each corpus was about 130,000 words and 11,000 words respectively.

5 Algorithms

The algorithms employed in this project can be broadly divided into three parts: word based, syntactic based and context based. For word based analysis, simple ngrams are used to get as much information out of words that surround *prt*. For syntactic analysis, both ngrams with POS and sentence-level syntactic analysis are employed. Finally for context, we will test whether the word preceding *prt* is novel in the discourse.

5.1 Word Ngrams

First we used unigram on 'wa' and 'ga' on our training corpus to obtain the ratio between the occurrence of 'wa' and 'ga'. This ratio was used as the baseline in the determination of the particles. That is, if there are no further information available on the particular instance of *prt*, we will put whichever particle that has the higher ratio of occurrence.

We also use word based bigrams to get information as to whether 'wa' and 'ga' are likely to be preceded by certain words. Upon each instance of *prt*, we see what the preceding word is, and check how many times 'wa' and 'ga' have occurred in the particular context. If there is a difference in the number of occurrences, we will pick the one with the higher occurrence.

5.2 Syntactic Ngrams

Similar to the word based ngrams, we first compile POS based ngrams on training copra. Then for each instance of *prt* in the testing corpus, find the ratio of 'wa' and 'ga' in that particular POS surroundings. So far we have only considered the word preceding and the word following *prt*. A wider ngram may be employed in the future work.

5.3 Threshold for ngrams

In combining these two ngram methods, we used a threshold to increase precision and also to minimize the effect of infrequent ngrams. The algorithm used for the threshold is the following:

```
if ( countwa + 3 > 2 * ( countga + 3) )
   return wa
else if ( countga + 3 > 2 * ( countwa + 3 ) )
   return ga
else
   return (no answer)
```

(countwa and countga are the counts of the particular contexts for 'wa' and 'ga' in the corresponding ngram data.)

We first added 3 to both the count of 'wa' and 'ga' so that contexts with low counts will not produce extreme ratio. For example, while the ratio between 0 and 1 is infinity, by adding 3 to both we get a more reasonable ratio of 3/4. For either ngram method to return an answer, we required that the count of the more frequent ngrams has to be greater than twice the count of the less frequent ngrams.

5.4 Syntactic Analysis

From Karita's work we know that if the subject is the subject of a phrase but not of the sentence, then 'ga' is always to be used and not 'wa'. We will implement this model by

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 30–34
Computer Science Department, Swarthmore College

	recall	precision
wa	67.5%	86.2%
ga	60.4%	60.5%
total	65.7%	80.4%

Table 1: Performance with word based bigram analysis

	recall	precision
wa	92.2%	87.0%
ga	94.0%	63.5%
total	92.6%	81.1%

Table 2: Performance with syntactically based trigram analysis

sentence level syntactic analysis. Finding sentence structures requires a full blown syntactic analyzer, which is difficult and complex enough to dedicate a whole project. Therefore, instead of designing a thorough syntactic analyzer, we will use a simple heuristic to determine a phrase in a given sentence. The heuristic exploits the fact that a phrase in Japanese is often segmented by a comma and always contain a verb phrase. Therefore, a prototype is considered to be inside a phrase if and only if a verb phrase occurs after the prototype and before a comma.

5.5 Contextual Analysis

One sense of 'ga' indicates the fact that the subject is not a novel topic. Thus by searching through the corpus and testing to see whether the subject has been mentioned previously, we can bias the output towards 'ga'.

6 Results

We counted the occurrences of 'wa' and 'ga' to obtain the baseline for this project. In our corpus of 130,000 words, 'wa' occurred 3439 times and 'ga' occurred 2138 times. Therefore, if the program guessed 'wa' for all instances of *prt*, we can expect it to be correct 62% of the time. The baseline in this project is thus considered to be 62%.

The word based bigram model yielded results with poor recall of 65.7% but precision at 80.4% which is significantly better than the baseline. The syntactically based trigram analysis achieved slightly better precision of 81.1% and huge improvement on recall of 92.6%. Guessing was not allowed for these tests. Therefore, if the context of *prt* did not match any ngram in the data, the program did not return an answer. Thresholds are not used for these tests, either. The recall rate here is calculated as the ratio between the count of guesses for 'wa' and 'ga' and the count of the occurrences of either particle in the corpus. The precision rate is the ratio between the count of correct guesses and the count of total guesses. For example, if 'wa' occurred 10 times in the corpus, the program returned 6 guesses for 'wa', and 4 of them were correct, the recall rate would be 6/10 and the precision would be 4/6. These results are summarized in Table 1 and Table 2 respectively.

Two models gave the same answer 88.4% of the time. When the answers were in conflict, the syntactically model was correct 55.9% of the time.

	recall	precision
wa	82.6%	88.4%
ga	65.7%	76.1%
total	78.4%	85.9%

Table 3: Performance with both syntactically and word based ngram analyses

These two ngram methods combined with the threshold algorithm described above yielded results that are better in precision but worse in recall compared to the results from syntactic ngrams alone. The improvement on the precision rate on 'ga' is significant, changing from 63.5% in the syntactic ngrams approach to 76.1% in the combined methods. When two models gave different answers, the answers given by the syntactic method was always chosen. The results are summarized in Table 3.

The same algorithm but with random guesses produced results only slightly poorer in precision. Note that the precision rates for the models with and without random guesses are exactly the same. This is due to the fact that all random guesses were 'wa' since 'wa' generally occurs more frequently. The results are in Table 4.

The syntactic method based on the analysis of phrases in a sentence gave poor results. When used alone, it predicted correctly the instances of 'ga' 56.2% of the time. When used to disambiguate the cases where the word based and the syntactic based gave conflicting answers, the precision dropped to 28%.

The contextual method was a complete failure. It turned out that in almost all cases, the word preceding *prt* is introduced prior in the corpus in other contexts. In the testing corpus, only one word preceding *prt* was a novel word in the discourse. Because of this unexpected result, no further analysis was possible concerning the distinction between a novel and a familiar topic.

	recall	precision
wa	100%	85.2%
ga	100%	76.1%
total	100%	83.8%

Table 4: Performance with both syntactically and word based ngram analyses with random guesses

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 30–34
Computer Science Department, Swarthmore College

7 Discussion

The poor recall rate of word based bigram model can be attributed to the fact that the bigram data compiled from the training corpus did not contain most of the proper nouns that occurred in the testing corpus. This is an irredeemable flaw with the word based model. Because both ‘wa’ and ‘ga’ can follow any proper noun, it is inherently impossible to capture a significant portion of them. The precision rate for the case of ‘wa’ was surprisingly high. A closer look at the bigram data revealed that ‘wa’ uniquely followed two common particles, ’de’ and ’ni’, both of which combined with ‘wa’ indicate topic introduction. The precision was high due to the fact that if the preceding word were ’de’ or ’ni’, the prediction of ‘wa’ yields almost 100% accuracy.

The higher recall rate for the syntactic model was expected, since the part of speech tags are vastly more general than words. It was interesting to see that its precision rate was also higher than the word based model, which is contrary to the expectation regarding its generality. We can attribute the unexpected precision rate to the fact that this simple trigram model embodies many of the characterizations put forth by Karita. First, the rule of substitution by ’no’ is reflected in the trigram model because ’no’ in this sense happens only between two nouns. Second, it is seen in the trigram data that ‘ga’ is much more likely to be followed by a verb, which is perhaps due to the fact that ‘ga’ happens inside a phrase where ’noun-ga-verb’ phrase is common.

The precision rate of 56.2% for the sentential syntactic analysis is statistically significant even though its absolute value is low. If we recall that the percentage of the occurrence of ‘ga’ among all occurrences of *prt* is only 25%, answering correctly 56.2% implies that the implementation captured at least some part of what Karita argues about ‘ga’ inside a phrase.

It is also worth noting that the testing corpus had an unusual distribution of ‘wa’ and ‘ga’. Where the distribution in the much larger training corpus was about 3 to 2, the distribution in the testing corpus was 3 to 1. This unusual trend might have affected the result one way or the other. Further testing with a different corpus is required to examine the effect.

8 Conclusion

With the combination of word ngrams, syntactic ngrams and phrase analysis alone, we have achieved 83.8% precision with 100% recall. This is promising considering the fact that we did not use a syntactic analyzer outside of our heuristics. With such an aid, we can perform a complete analysis of sentential structure, which will probably boost the precision to the high 80’s. With further work with a syntactic analyzer, we will perhaps be able to disambiguate all instances of ‘wa’ and ‘ga’ that have distinct syntactic contexts.

The project did not succeed in disambiguating the cases where deeper contextual analysis is required. The problem of contexts and semantics beyond pure statistics of words is a notoriously difficult one across all NLP fields. Thus we do not expect that we will be able to solve the problem without employing an entirely novel method yet undiscovered in the field of NLP. However, we do believe that using implementations similar to the current one can contribute to practical applications such as machine translations and grammar checking in Japanese. Even though by word based ngrams and syntactic analysis alone cannot capture all occurrences of ‘wa’ and ’ga,’ they can give correct answers most of the time for most of the cases.

9 references

David Yarowsky, 1992. *Word-Sense Disambiguation Using Statistical Models of Roget’s Categories Trained on Large Copra*

David Yarowsky, 1993. *One Sense Per Collocation*

David Yarowsky, 1995. *Unsupervised Word Sense Disambiguation Rivaling Supervised Methods*

Karita Shuji, *‘wa’ to ‘ga’ wo meguru shomondai ni tsuite*, (http://bakkers.gr.jp/ karita/report/report$_k anyou - j.html$)

$KudoTaku(http : //cl.aist - nara.ac.jp/\ taku - ku/software/mecab/)$

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 30–34
Computer Science Department, Swarthmore College

Machine Translation Evaluation by Document Classification and Clustering

Feng He and Pascal Troemel
Swarthmore College,
500 College Avenue,
Swarthmore PA 19081, USA
{feng,troemel}@cs.swarthmore.edu

Abstract

We propose a Machine Translation evaluation system which does not require human-translated reference texts. The system makes use of a comparison between the performance of a computer's execution of NLP tasks on source text and on translated text to judge how effective the translation is. The difference in accuracy of the NLP task exectutions is used as a gauge for judging the competence of the Babelfish online translation system.

Keywords: *Machine Translation Evaluation, Document Classification, Clustering.*

1 Introduction

1.1 Machine Translation Evalutation

Machine translation research has been going on for several decades, and there are a number of systems available for use, mostly between English and a European or Asian language. Notable examples are products from Systran, which are used in Altavista's Babelfish online translation service. Machine translation evaluation has long been an extremely arduous task which requires much human input; more recently, the BLEU evaluation system [3] has made use of a much more automated, and thus more practical, approach. However, the BLEU system still requires the presence of several correct, human-translated reference texts (see Section 2.1 for an overview on the BLEU system). We propose a system which does not have this requirement, a system that is capable of judging the competence of a translation simply by comparing the source and target texts. We believe that this freedom from human input is important; human translation is a time-consuming and costly task in the MT evaluation process, and to cut it out alltogether will undoubtedly save resources.

We attempt to either prove or disprove the notion that although a machine translation may seem ineffective to a human reader, it still holds sufficient correct information to allow a computer to adequately perform the NLP tasks of text classification and clustering on it. If this is indeed the case, then even though machine translations may not yet be acceptable as accurate representations of works of literature in their original language, they may be far from useless to a computer capable of interpreting ("understanding") them.

Ultimately, a translation should be judged on how much information it retains from the original text. Following this notion, we judge the effectiveness of a translation system by comparing the accuracy results of a computer's NLP task execution on the source text and the target text. We expect a drop in performance that can then be interpreted as "acceptable" or "unacceptable," which serves as an evaluation of the system. Indeed the drop in performance gives us a quantitative measure of the translation's effectiveness.

In Section 2, we discuss a few relevant examples of previous work in the area of machine translation evaluation. Section 3 serves to describe how we collect data. The various experiments we performed are discussed in Section 4. In Sections 5 and 7 we share our results and conclusions.

1.2 Text Classification and Clustering

Text classification and clustering are two common NLP tasks that have been shown to obtain good results with statistical approaches. Classifications refers to the assigning of new documents to existing classes. The models for existing classes are built from documents known to belong to the classes. Usually a document is assigned to a single class, but it is also possible that a document has multiple class-tags.

Clustering refers to the dividing of a collection of documents into groups (clusters). These clusters are not pre-defined, although the number of clusters can be specified. It is also possible to create a hierarchy

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 35–39
Computer Science Department, Swarthmore College

of clusters, in which a cluster is further divided into sub-clusters.

Classification and clustering have long been studied, and there are many effective toolkits for both tasks. These two NLP tasks are natural choices for our experiments because they can be effectively performed on our data sets.

2 Related Work

2.1 MT Evaluation

The BLEU machine translation evaluation system [3] proposed in 2002 produced very respectable results, effectively emulating a human translation judge. The system produced a score for a given translation by comparing it to a group of "perfect" human-translated reference texts using n-gram precision values. After a few necessary details such as a brevity penalty had been added, the BLEU system's scores were found to correlate closely with those given by a variety of human judges on a group of test data. The main weakness of this system is its dependency on human-translated reference texts. Although it is far more automated than older, completely human-dependent "systems," which relied completely on human evaluation, the BLEU method still requires several correct translations. This means that, for every new text that the MT system is tested on, the BLEU evaluation system must first be presented with good reference texts, which must be produced by a group of humans. This can get expensive when a machine translation system is tested on numerous documents, a case that is clearly possible during the production of a truly effective MT system.

2.2 Tools

The Bow toolkit [1] was designed for statistical language modeling, text retrieval, classification and clustering. It provides a simple means of performing NLP tasks on a body of newsgroups, and was thus a very useful tool for us. We produced our results for text classification and clustering with the help of this system.

2.3 Other Related Works

In [4], Weiss et al. showed that newsgroup postings can be reasonably classified using statistical models.

Palmer et al. [2] investigated the effect of Chinese word segmentation on information retrieval. This work suggests that well-segmented Chinese text will improve performances of NLP tasks. Chinese segmentation is an active area of research, partly because current systems produce very poor segmentations. As we do not have a working segmenter for Chinese text, we expect our results to be accordingly affected.

Finally, Yang [5] gives a good overview of statistical text classification.

3 Data

The Internet is a rich resource for both English and Chinese texts. Chinese text on the Internet is encoded using Chinese character sets. There are several existing Chinese character sets. The most popular ones are: GB (simplied Chinese), Big5 (traditional Chinese) and UTF-8. Both GB and Big5 use two bytes to encode one character; UTF-8, a more recent encoding standard, uses three bytes. The differences between these encoding standards complicate data collection, as a data set must have uniform encoding to be usable. Character set detection and conversion are required. In addition, the character boundaries are often misaligned in online documents, producing corrupted Chinese text. These need to be removed from the data set.

For our experiments, we downloaded newsgroups articles as our data. Newsgroups are online communities where people send posts covering many different topics. There are newsgroups in both Chinese and English covering similar topics, providing parallel corpora. Postings are self-organized into topics, which serve as natural labels for document classification. The following data sets were downloaded:

- Data Set 1: Postings from 5 Chinese newsgroups were downloaded. The newsgroups and their topics are:
  - talk.politics.china – (Chinese politics)
  - cn.bbs.rec.movie – (movie reviews)
  - cn.sci.medicine – (Chinese medicine)
  - cn.culture.buddhism – (Buddhism)
  - cn.comp.software.shareware – (software/shareware)

  These newsgroups are chosen such that they are terminal groups (they are not further divided into subgroups), and they cover very different topics. About 800 postings were downloaded. The postings that contained corrupted Chinese or were too short (fewer than 50 Chinese characters) were removed, leaving about 400 to 700 usable postings from each newsgroup. The total number of postings is around 2000.
- Data Set 2: English translations of data set 1. We used Altavista's online Babelfish translation.
- Data Set 3: To create a parallel corpus to data set 1, we downloaded articles from 5 English newsgroups. The newsgroups are:

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 35–39
Computer Science Department, Swarthmore College

- talk.politics.usa – (American politics)
- rec.arts.movies.reviews – (movie reviews)
- misc.health.alternative – (alternative medicine)
- soc.religion.christian.bible-study – (bible study)
- comp.softawre.shareware.announce – (software/shareware)

These newsgroups were chosen such that they cover similar topics as data set 1. About 3500 postings in all, roughly 700 from each group.

- Data Set 4: Chinese translations of data set 3 using Babelfish

4 Experiments

4.1 Experiment 1: Classification on Chinese Source

This experiment serves to compare the accuracy in performance of text classification on Data Sets 1 and 2: Chinese as source text and English as target text. We expected a significant drop in accuracy between the source and target performances, marking a loss of information in the translation. A typical member of Data Set 2, the target English, follows:

> *Perhaps in the very many movies, the audience already was used to it the good Lai shipyard -like violence and the love. But truly could attract the audience or has the male is positive just the charm actor, they usually could initiate audience's favorable impression even respect. Below is one good Lai shipyard cinema world first ten very male ranks announcement.*

Clearly the translated text is not good English, but it is also relatively clear that the topic of the posting is the movies, and that the correct newsgroup classification is *cn.bbs.rec.movie*, and not one of the other candidates: *cn.comp.software.shareware, cn.culture.buddhism, cn.sci.medicine,* or *talk.politics.china.* The purpose of this experiment is to discover whether a classification system is able to correctly classify documents such as this one.

We used the rainbow toolkit to classify the documents. To get a more rounded and reliable precision average for each data set, we performed classification using each of three different methods: Naive Bayes, TFIDF, and Probabilistic Indexing. Each data set was partitioned into training and testing sets. Either 80% or 60% of the documents from each class was randomly selected as traning data to build the class models. The remaining 20% or 40% was used as testing data and were classified. This procedure was repeated for 5 times each time different subsets was selected as training and testing data, and results averaged. The average results from data set 1 and 2 were compared. Note that the Chinese documents were not segmented, meaning each character was treated as a token, instead of a word, which usually consists of several characters. We expect this to lower classification performance on the Chinese documents.

4.2 Experiment 2: Classification on English Source

This experiment serves to compare the accuracy in performance of text classification on Data Sets 3 and 4: English as source text and Chinese as target text. Again, we expected a drop in accuracy between the source and target performances.

4.3 Experiment 3: Clustering on Chinese Source

In this experiment, we performed clustering on data sets 1 and 2 using *crossbow*, which is part of the rainbow toolkit. The number of clusters was specified to be 5, in accordance with the number of newsgroup topics. Note that, since no cluster topics were provided, the resulting clusters may or may not correspond to the original newsgroup topics. Indeed it is possible that articles from one newsgroup be divided into two clusters, while several newsgroups be merged into one cluster. However, there is usually a clear correspondence between the clusters and the original topics.

4.4 Experiment 4: Clustering on English Source

Experiment 4 is a repeat of experiment 3 on data sets 3 and 4.

5 Results

5.1 Experiment 1 and 2: Accuracy Results

	Classname	0	1	2	3	4	Total	Accuracy
0	software	318	19	1	22	.	360	88.33%
1	health	5	291	.	34	.	330	88.18%
2	movies	2	3	44	281	.	330	13.33%
3	religion	2	10	.	268	.	280	95.71%
4	politics	3	33	.	114	19	169	11.24%

Table 1. Classification with Probabilistic Indexing results example for Chinese as TARGET

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 35–39
Computer Science Department, Swarthmore College

Method	Test-Set Size 20%	40%
Naive Bayes	90.45	90.16
TFIDF	92.35	91.79
Probabilistic Indexing	86.85	86.14

Table 2. Classification accuracy, Chinese as SOURCE

Method	Test-Set Size 20%	40%
Naive Bayes	92.76	93.07
TFIDF	94.32	93.35
Probabilistic Indexing	90.40	90.39

Table 3. Classification accuracy, English as TARGET

Table 1 shows a typical classification result using documents from the English newsgroups. The rows represent the original newsgroup topics. The columns represent the number of documents assigned to each class. For example, of the 360 documents from *comp.software.shareware.announce*, 318 were assigned to class 0 (the correct class), 19 were assigned to class 1 *(mis.health.alternative)*, and so on.

Tables 2 and 3 summarize results from experiment 1. Each row records results using a specific modelling method. The size of the testing set was set to be either 20% or 40% of the total number of documents, and the results are tabled accordingly.

Method	Test-Set Size 20%	40%
Naive Bayes	97.38	97.74
TFIDF	97.60	97.97
Probabilistic Indexing	95.26	95.18

Table 4. Classification accuracy, English as SOURCE

Tables 3 and 4 summarize results from experiment 2, in which original English documents and their Chinese translations were classified.

5.2 Experiment 3 and 4: Clustering Results

Tables 6 and 7 summarize results from experiments 3 and 4. In each of the experiments, each data set was divided into 5 groups, which often corresponded to the original newsgroups. The clusters were matched with the newsgroups so that the total number of documents in wrong clusters was minimized. The second column in each table shows the number of correctly clustered documents out of the total number of documents. The third column gives the percentage accuracy.

Method	Test-Set Size 20%	40%
Naive Bayes	90.74	89.75
TFIDF	96.08	96.05
Probabilistic Indexing	65.42	65.38

Table 5. Classification accuracy, Chinese as TARGET

Texts	Accuracy	
Chinese as SOURCE	1385/1994	69.46%
English as TARGET	1431/1994	71.77%

Table 6. Clustering Accuracy

Texts	Accuracy	
English as SOURCE	1961/3674	53.38%
Chinese as TARGET	1817/3674	49.46%

Table 7. Clustering Accuracy

6 Discussion of Results

The results of Experiment 1 are unexpected: the target text actually performs better on the classification than the source text. This should obviously never occur, unless the machine translation system is so effective that it actually *corrects* errors instead of creating them. Since it is extremely unlikely that Babelfish is such a system, we need an alternate explanation. We propose two hypotheses, namely that either (1) the task of classifying Chinese text is somehow inherently more difficult than classifying English text, or (2) the lack of any segmentation in our Chinese mapping scheme is causing words to be incorrectly interpreted by the classification system. These two factors could easily be the reason that the results in Table 1 are actually lower than those of the target English in Table 2.

The results of Experiment 2, on the other hand, are more as expected. The source English performs much better than the target Chinese, as can be seen in Tables 3 and 4. These results suggest that the translation system did not perform very well; the accuracy average dropped from 96.68% to 82.86%, a 13.82% loss, which amounts to an error increase of about 500%. It is evident from Table 4 that the target Chinese results suffer greatly from the tests performed with the Probabilistic Indexing method. It is likely that information somehow key to this particular method was lost in the translation, and that this loss greatly hampered the classification of the documents. Table 5 shows

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 35–39
Computer Science Department, Swarthmore College

the results of a typical classification test-run using the PI method, and it is very interesting to note that the great majority of postings in the "movies" and "politics" newsgroups were incorrectly placed into the "religion" newsgroup. Reasons for this misplacement could be any of several possibilities, such as the all-encompassing nature of religion.

The same trend was observed in Experiments 3 and 4. Clustering result improved from Chinese documents to English translations, but deteriorated from English documents to Chinese translations. One interesting observation is that, clustering performed better on data sets 1 and 2 than 3 and 4. One possible reason is that data sets 3 and 4 are a lot bigger (contain almost twice as many documents).

It is difficult, however, to come up with a concrete measure of "acceptability" from these numbers. How do we know what is an acceptable drop in accuracy, or an unacceptable error increase? The answer to these questions may depend on the specific purpose of the MT system under evaluation: if its purpose is simply to provide a general idea of the original text, perhaps a 13.82% drop in accuracy is a perfectly adequate performance; but if its purpose is to provide an accurate, well-organized text fit for publication (which may be the purpose of future MT systems), a drop of even 2% may be unacceptable.

7 Conclusion and Future Directions

In this paper we proposed a machine translation evaluation system that is not bound to human-translated reference texts, instead making use of a text classification and clustering performance comparison. We described our experiments in which we evaluated the Babelfish translation system on newsgroup postings. The results were mixed. The Chinese to English translation actually improved classification and clustering performances, while the English to Chinese translation lowered performances. We hypothesize that this is either because Chinese text inherently does not fit well with the built-in language models in the Bow toolkit, or that the lack of segmentation hampered performance.

There are some interesting extensions to the experiments described in this paper. It will be interesting to see how much segmentation will improve task performances on the Chinese documents. We could also compare performances from other NLP task such as information retrieval. Finally, given that there are many NLP packages for English, and relatively few for Chinese, it is of practical value to see if it is possible to combine NLP packages with some machine translation system to obtain NLP packages for other languages.

References

[1] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

[2] D. D. Palmer and J. D. Burger. Chinese word segmentation and information retrieval. *AAAI Spring Symposium on Cross-Language Text and Speech Retrieval*, 1997.

[3] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia*, 2002.

[4] S. A. Weiss, S. Kasif, and E. Brill. Text classification in usenet newsgroups: A progress report. *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[5] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, vol 1:pp 69–90, 1999.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 35–39
Computer Science Department, Swarthmore College

Constructing a Lossless and Extensible Part-Of-Speech Tagger

Jeffrey Regier
jregier1@swarthmore.edu

Abstract

This paper describes the design of a lossless and extensible part-of-speech tagger, with the intent of illuminating general principles underlying part-of-speech tagging. To this end, the described tagging program has been designed with form emphasized above performance and even above completeness. A key design premise is that a tagger is most naturally constructed through the implementation of largely independent methods of inferencing tags (e.g. analyzing suffixes, analyzing context). However, occasionally communication between these methods is necessary, as is communication of each method's output to a component that ultimately decides each word's part-of-speech. It is argued that a "stochastic-tag", implemented as a class and defined roughly as a set of pairings between each part-of-speech tag and its hypothesized likelihood, is an appropriate and perhaps optimal vehicle for communication. Finally, several techniques of combining the stochastic tags returned by methods are evaluated.

1 Introduction

Deducing the correct part-of-speech (POS) of the vast majority of words in a testing corpus requires little more than recalling the POS most frequently assigned to the same word type in a training corpus of reasonable size. More sophisticated POS taggers attempt to exceed this level of performance by correctly tagging 1) words types not found in the training corpus, and 2) words types with multiple parts-of-speech (POSes).

The arrangement of POSes is typically viewed as governed either by a set of simple rules with many exceptions or by distributions which deviate from randomness. These views correspond to two paradigms between which most tagging programs are divisible. Rule-based taggers generate a set of rules for choosing a tag, often including rules that are exceptions to other rules. Choice of the tag in the testing corpus is simply a matter of applying the rules invented during the training section of the corpus.

Stochastic taggers, on the other hand, have much similarity to Markov models. They too learn from the training corpus, but not merely by creating rules to be used during the testing phase. Their learning takes the form of improving the accuracy of rules' parameters. While there are many taggers that claim to be hybrids, it is perhaps more natural to view the set of rule-based taggers as the subset of stochastic-taggers with parameters that may only hold binary values.

2 Research Goals

This research is motivated by the search for a deeper understanding of the principles that underlie POS tagging. To this end, a POS tagger has been constructed from scratch with the goal of learning as much as possible from the aspects of the document which have been selected for exploration. Restated, a key design goal is to model language such that POS deduction can occur without any loss of information due to processing failure. While the categories "loss of information due to collection failure" and "loss of information due to processing failure" are perhaps not disjoint, an example may clarify this key distinction. Information available from a word's prefix, context, etc. can be lost if no attempt is made to process that aspect of a word. Information can also be lost when an aspect of a word is collected, but either for reasons of efficiency, ease, or oversight some of the available information is not fully incorporated into the ultimate prediction of that word's POS.

In order to minimize the loss of information due to processing, the program's form is emphasized in the design process above all else. Thus, some features which could

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

have improved accuracy were omitted because they were not central to building a framework for lossless POS tagging. For one part of the program, for which programming the optimal method proved beyond the scope of this research, several suboptimal methods were suggested and clearly marked as suboptimal.

It has already been suggested that designing a tagger that minimizes loss of information due to processing failures is a appropriate approach to understanding POS tagging and language. Additionally, if the framework to which components are added is readily extensible, then it is likely the framework has captured some truth about language. Conversely, if the framework designed captures the complexities of language without using techniques that sacrifice a full understanding either due to obscure structure (e.g. neural networks) or full consideration (e.g. rule-based POS taggers), then it will be simple to building new considerations into the existing framework.

[perhaps use this paragraph instead of some of the previous sentences] The introduction of rule-based taggers is one way to counteract the incredible complexity of creating a stochastic-based tagger. The motivation behind this project is that another method of overcoming the complexity of creating a stochastic model of language is by placing an explicit focus on structure and simplicity, and admission of areas of the tagger which will require further work, rather than garbling ideas together to avoid acknowledging the need for future research and to get higher accuracy measures at the expense of a consistent methodology for addressing considerations.

3 Program Structure

3.1 Parsing

This research's emphasis form is natural to accommodated by parsing the training corpus and the testing corpus into separate document objects containing sentence objects which in turn contain word objects.

Each document is ultimately a series of word tokens (including punctuation) but the sentence is nonetheless a relevant unit since it is the largest unit of inferencing. Because a human reader has no trouble figuring out words' POSes without looking beyond the current sentence, it is reasonable that the program not do so either. Of course, this is assuming that the reader is not using other sentences to learn about the language. The equivalent to a reader with knowledge of language is the program running on the testing phase. The exception to this rule is that words used near a sentence are likely to be used with the same POS in later sentences. This exception could be dealt with via horizontally accessed variables. [horizontal?]

Each Word object contains a type, which is the text made lowercase of the word (case is another field). Just as the sentence method handle context sensitive rule deduction and application, the word class handles context free inferencing. Context free inferencing includes operating a dictionary and set of stochastic rules, and will be discussed later.

3.2 Stochastic-Tag

In other taggers, each word is associated with one or at most a few tags. Ultimately, tagging is applying a series of methods to a given word, and combining the output of these methods. If the output of each method is merely a tag or set of tags, then combining is probably best performed by simply picking the POS returned by all methods. This severely limits the abilities of taggers, since most methods of inferencing are not totally confident about and equally confident in the tags in the set that they report. Naturally, there are ways to work around an unwillingness of methods to return both the absolute and relative likelihood of every POS. However, if abstraction barriers are to be respected (and doing so is key to dealing with the large amount of complexity likely inherent to dealing with language), then each method must describe its confidence in its each tag. The stochastic-tag provides is the vehicle for doing so.

The data portion of a stochastic-tag associates each POS with a number representing the likelihood that it is the tag. Accessor methods to return the most likely tag, the percent likelihood of each tag, and the confidence in the most likely tag. Additional modifier methods make automatic some types of adjustments of the likelihood of a POS. Finally, some more complex methods allow the averaging and summing of multiple stochastic-tags. Finally, a product method the confidence in every POS according to a vector taken as a parameter.

[should be proofread]In whole, the stochastic-tag provides a convenient format for relaying the information required of each component. The proportion of the sum of the numbers associated with each of the POSes made up of each number relates the relative likelihoods of each part. Greater deviation from 1 divided by the number of POSes indicates greater amounts of information provided by the component. Similarly, the total sum of the numbers provides confidence in the measure. The design of the components will show how the stochastic-tag's format for relaying information is convenient and ultimately enables inter-component comparison and combination.

4 Context Free Methods

Context free inferencing refers to a set of methods used to deduce POS that do not consider a word's context within a sentence. There are several methods for doing this.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

4.1 Past Word Type Usage

A single Dictionary object is created to serve as the store for all the words encountered during the training phase. During the testing phase the dictionary is used to retrieve statistics about the uses of the word. Word that have not been encountered in the training section will not be found in the dictionary. Similarly, words that have been encountered only a few times may take on novel POSes in the testing corpus. Actually, there is a chance of a word type taking on any formerly unseen POS in the testing corpus if the word type has only been encountered a finite number of times in the training corpus. More generally, given a finite number of encounters with a word type in the training corpus the sample distribution of POSes will differ from the true population distribution for a given word type. There is never a case where dictionary lookup is certain to be correct, and thus more information is always potentially helpful to choosing a tag. Since the dictionary tends to be the premere source of information about a word token's POS, in the case of words not found in the dictionary other methods become especially influential. In isolation of other methods, the dictionary correctly tags approximately 85% of words when given a large training corpus. The addition of a few simple rules such as guessing noun when no instances of the word are found increases accuracy 92%. However, this type of modification has been commented out of the program's code because it is inconsistent with the stochastic approach underlying this research.

4.2 Past POS Usage

A crude method of guessing POSes is to always guess the POS that most frequently occurs in the training corpus without even considering the specific word type. However, appropriately filling a stochastic-tag requires including information other than that the most likely POS has 100% chance of correctness while the other POSes have a 0% chance of correctness. Rather, accurate information concerning the likelihood of each POS must be included to avoid unfairly undermining the estimates of other methods. It is thus more appropriate to fill each field with the percent of tags in the training document with that POS. By being truthful about its certainty in each POS, this component can hope to have its information accurately weighted upon combination with other POSes.

Applied in isolation, this method will guess that every word is a noun, with accuracy of approximately 21%. An example will make clear the value of avoiding the rule-based method of guessing noun when a word type is not found in the dictionary, even without consideration of methods other than "Past Word Type Usage" and "Past POS Frequency". Obviously, when "Past Word Type Usage" reports that it has seen the word type 0 times, it is more reasonable to follow the suggestions from "Past POS Usage". Less obviously, when "Past Word Type Usage" reports that it has encountered a certain word token a small but nonzero number of times, then there still is some information about smoothing that could potentially be obtained from "Past POS Usage".

4.3 Suffix Identification

Some words have suffixes which provide useful hints as to their POS. For example, words ending in "ly" are likely to be adverbs while words ending in "ing" are frequently verbs. There are also many endings that provide less certain mandates, but nonetheless slightly increase the likelihood of a word taking on a certain POS or decrease the likelihood of a word having another POS. More generally, shared word endings influence the distribution of probabilities.

Words can share suffixes of different lengths with multiple classes of words. How are multiple shared suffixes to be reconciled? With a stochastic-tag, of course. The stochastic-tag features an accumulate function, used to sum the contents of multiple stochastic-tags. If a word has one suffix that occurs many times in the testing corpus but provides no clear mandate, then it should be like adding unbiased white noise to the stochastic-tag. The number associated with each POS in the stochastic-tag returned by this component is the number of words with a matching suffix in the train corpus. If a suffix of length 3 is matched then a suffix of length 2 will also be matched, bringing up questions of double counting. This is a recurring theme, which tragically is beyond the scope of this project. Nonetheless, even in isolation this method performs much better than randomness, choosing the correct tag for approximately 55% of words when recall is set to 35%. Ultimately, there will be no need to manually set the level of recall, since doing so is inconsistent with this research's objectives.

4.4 Affix Transformation

Some words, when stripped of a prefix or a suffix, may match a word type in the dictionary. However, just because a word shares a root word with a word in the dictionary does not mean that it has the same POS. In fact, languages frequently rely on affixes to change a word's POS. While recall will be lower for this method than for "suffix identification", it will be more accurate for the words it finds, as is illustrated by an example. While "ly" may be highly correlated with adverb, if the root word is not itself a verb then "ly" is likely be a red herring (or at least to be over confident).

In practice this method is useful only when recall is set at less than 12%.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

5 Context Sensitive Methods

There is also information available about a word's POS based on its location and surroundings within a sentence.

5.1 Preceding Word's Type

Certain words tend to immediately precede words of certain POSes. For example, "the" tends to precede a noun. By identifying these words in the testing corpus a stochastic-tag can be returned indicating all information relevant to each word.

5.2 Surrounding Words' POSes

Words with certain POSes tend to be located in the same position relative to other words with certain POSes. For example, articles tends to precede nouns, either directly or perhaps with an adjective in between the article and the noun. By identifying these words in the testing corpus a stochastic-tag can be returned of the suspected POS distribution of the current word. However, there is one major difficulty with this method that was not relevant to the "Preceding Word's Type" component. The type of the preceding token is certain, so applying the data collected from the training corpus is not difficult. During the training phase it is not permissible to use the truth about the previous words POSes to deduce the current word's POS. An estimate must be substituted for the truth, and for simplicity this estimate comes from the previous word's dictionary-based tag. Again, unlike the truth, the dictionary-based tag is a stochastic-tag, and so it does not clearly specify one POS. Using the "product" function of stochastic tag it is possible to use this tag as a vector for scaling distribution for each POS, and subsequently accumulating the resulting stochastic-tags.

6 Techniques for Combining Methods' Opinions

For each word, each method must not only suggest the most likely POS, but must suggest the likelihood of each element of the set of POSes. This approach largely isolates the methods, such that additional code is needed to combine the methods' outputs. Since the return type of each of the methods is a stochastic-tag, the most natural method of combining the information they contain is to create one "meta-stochastic-tag", and only then pick the POS deemed most likely by the "meta-stochastic-tag". Incidentally, leaving a "meta-stochastic-tag" associated with each word instead of just its most likely POS allows for more accurate context sensitive inferencing of neighboring words.

6.1 First Technique: Use Case Statements

A crude technique of combining the stochastic-tags returned by each component can be implemented using a series of case statements. Let the internal stability of a stochastic-tag be the maximal deviation of the estimated probability of any POS from a value denoting no information (i.e. $\frac{1}{|POSes|}$). Then, if the internal stability of the stochastic-tag returned by the "Past Word Type Usage" method is greater than 0 (i.e. the word type was encountered in the training corpus), let the meta-stochastic-tag be the "Past Word Type Usage" tag. Otherwise, use the suffix identification method's return value as the meta-stochastic-tag if the internal stability claimed by the suffix is over 50%. Precede in this fashion through the remainder of the methods, assigning the tag returned by "Past POS Usage" as a last resort.

Clearly this method is quite imperfect. The most glaring problem is that so little use is made of the information provided by the stochastic-tags returned by each methods. While with this technique each method's return value could potentially influence the final selection of a word's tag, the way this is achieved is far from optimal.

6.2 Second Technique: Pick the Most Confident Opinion

A superior technique lets the meta-stochastic-tag equal the method's output that makes the strongest claim to correctness. Without combining tags, even in theory no better choice can be made than choosing the method's output with the greatest internal stability (previously defined).

While conceptually this technique is an improvement over the first technique, it is still lacking. Recall that the motivation for introducing stochastic-tags was not to directly pick the method claiming to be the most certain. Rather, it was to allow the unbiased combination of information returned by various methods. These first two techniques are similarly suboptimal in that they both discard all of the returned tags except for one. Assuming that the components are accurately reporting the likelihood that they are correct, this does not preclude their being valuable information in the tags returned by the other components. Even given credible evidence, one is still aided by the collection of less credible pieces of evidence. Perhaps multiple pieces of less credible evidence all point mainly towards the same POS, thus overwhelming the most credible evidence.

6.3 Third Technique: Sum POS Histories

For this technique combination is performed by summing the number associated with each POS. Recall that this number as a proportion of the sum of the numbers associate with all of the POSes is the likelihood that this POS is correct, at least from the perspective of a particular method. However, this number is not a likelihood in isolation of the other POSes. This number is generated by seemingly similar processes across methods; when an instance is encountered a stochastic-tag's noteOccurrence

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

method is called, which increases the count of the appropriate tag by 1. However, the tags returned by "Past POS Usage", for instance, will be called many times yet contains little information not accounted for by the dictionary-lookup method. However, with so many calls made to noteOccurrence, the stochastic-tag returned by "Past POS Usage" method will overwhelm other methods, effectively always choosing one of the least accurate tags. It was eventually realized that this number was not necessarily comparable among tags returned by different methods. This technique is discussed because its failings highlight a key difficulty in combining stochastic output from methods that generated their output in isolation.

6.4 Fourth Technique: Sum or Multiply POS Likelihoods

The simplest attempt at remedying the very fundamental problem raised by the third technique is to sum or multiply the likelihood of each POS instead of combining the number of calls to noteOccurrence. While dealing with percentages clearly improves on the unworkable model suggested by the third technique, in a way it is a step backwards. Certainly by using the percent rather than the number of calls to noteOccurrence the use of some of the most important features of the stochastic-tag are preserved: Summing two stochastic-tags that represent the same distribution will result in the same distribution of percentages. Multiplying a stochastic-tag that is 100% confident in one POS by another tag typically also gives returns a tag that is 100% in the appropriate POS. Both of these results are desirable. However, switching the tags that were summed in the previous example with the tags the were multiplied in the previous example provides counterexamples to the correctness of both methods.

Moreover, by dealing with likelihood instead of the absolute number, the ability to cope with small sample size is limited. Given the large number of English word types and the even larger number of bigrams, small sample size will be an issue even with the largest of corpora. Moreover, even if sample size was finite but not small, combining the stochastic-tags without loss of information would still require knowledge of the sample size.

6.5 Fifth Technique: Use Information External to the Methods

Before learning from a sentence of the training corpus, the tagger attempted to tag the sentence based on the data collected from the sentences that preceded it. It then preceded to judge its accuracy at tagging that sentence. Using a procedure which simulated the operations of a Kalman filter, a prediction of the current accuracy of each method was maintained throughout training. (A Kalman filter is a procedure which adjusts for the Gaussian distribution of noise.) By the time the tagger had finished learning it knew the optimal estimate of how each method would perform throughout the testing phase of the document, during which time it would be unable to improve itself since it did not have access to the truth.

Using a Kalman filter-like procedure generates the optimal estimate of how the tagger will perform on the training corpus. Many other methods would have resulted in worse performance. Method's accuracy from earlier is not a good indication of its current level of performance, since the method become more accurate as it trained. Also, a method's performance on the previous sentence would have been a poor way to gauge accuracy, since the previous sentence may have been unusually difficult or unusually simple to tag. Waiting until the tagger has processed the whole training corpus to gauge the abilities of the tagger also would not have been a good alternative; measures obtained from testing on the document on which training has occurred are largely meaningless. Finally, reserving part of the training corpus for testing prior to the actual testing is wasteful and suboptimal.

Based on the judgment of each method's accuracy, those methods consistently performing closer to their predictions were given appropriately more weight, using the stochastic-tag's accumulateByMerit function. Note that accuracy for a stochastic-tag is rarely a simple correct or incorrect, but rather the degree to which it is correct. A stochastic-tag's accuracy is the percent with which it votes for the correct tag. While it was heartening that accuracy could be so aptly defined, second guessing the internal measures of correctness entailed overwhelming complexity; further attempts along this trajectory were abandoned.

6.6 Suggestions for Future Research

Further development of the stochastic-tag is necessary if the output of methods is to be combined losslessly. Specifically, each method must accurately report more than the likelihood of each POS. Rather, each method must also adjust its output to account for the size of its sample. Modifying the "Past Word Type Usage" method such that it will adjust for sample size could be done perhaps by using the t-distribution to generate confidence intervals surrounding each estimate of POS likelihood. Generating confidence intervals around the unbiased estimates of several other methods would also be possible. Modifying operations on stochastic-tags to accommodate confidence intervals is well beyond the scope of this paper.

The covariance between methods also needs to be controlled. For example, when a word token has occurred often in the training corpus, "Past POS Usage" offers nothing that is not accounted for "Past Word Type Usage". Similarly, "Suffix Identification" is largely redundant when "Affix Transformation" is highly relevant.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

Controlling for covariance is a standard method used when performing regression analyses, so perhaps those methods could be incorporated.

References

Brill, Eric. 1994. "A Report of Recent Transformation-Based Error Driven Learning." *http://www.cs.jhu.edu/brill/dissertation.html*

Carlberger, J. & Kann, V. 1999. "Implementing an Efficient Part-Of-Speech Tagger." *Software Practice and Experience, 29(9), 815-832.*

Charniak, Eugene. Statistical Language Learning. The MIT Press, Cambridge, MA (1996)

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 40–45
Computer Science Department, Swarthmore College

A hybrid WSD system using Word Space and Semantic Space

Haw-Bin Chai, Hwa-chow Hsu
Dept. of Computer Science
Swarthmore College
{chai, hsu}@cs.swarthmore.edu

Abstract

We describe a hybrid Word Sense Disambiguation (WSD) system using the context-based frameworks of both Word Space and Semantic Space. We develop confidence measures for the results generated by each model. To solve a WSD task, each classifier is run independently and the results combined using the confidence measures. The result is a more robust solution to the disambiguation task.

1. Introduction

Word Sense Disambiguation (WSD) remains a difficult problem in natural language processing. The source of the problem lies in the ambiguity of language – many words, in many languages, have different meanings in different contexts and situations. An often used example is the English word 'bank', which has the institutional sense (as in 'National Bank') and the shore sense ('river bank'), among others. Native speakers develop an intuitive ability to recognize these distinctions, but it is difficult to translate this ability into computational algorithms or models.

A variety of approaches have been attempted, ranging from statistics-based to connectionist methods. We focus on the Word Space and Structured Semantic Space approaches to WSD, two methods that develop the idea of modeling a language as a vector space. In the case of Word Space, a vector space is constructed for a particular word we wish to disambiguate, and spans all possible context words with which that word appears in a training corpus. Each instance of the ambiguous word is represented by a single point in this space. Structured Semantic Space is constructed similarly, but uses semantic information about context words rather than the words themselves, and models an entire language as opposed to a single ambiguous word.

Given the two methods' mutually independent information domains, we hypothesize that a hybrid system using both methods can take advantage of the strengths of each method while compensating for their weaknesses. One explicit way we hope to achieve this is by developing confidence measures for the results of each method and taking these into account when forming our final result.

The remainder of the paper is organized as follows: Section 2 explains the concepts of Word Space and Semantic Space in more detail. Section 3 describes our hybrid system and confidence measures for the results produced by Word Space and Semantic Space. Section 4 describes our implementation of this system. Section 5 presents our results, of which a discussion follows in Section 6. Finally, Section 7 describes possible future work.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

2. Related Work

2.1. Word Space

A Word Space (Schütze, 1992) is an n-dimensional space of contexts of a particular word w, where n is the total number of unique words that co-occur with w in a training corpus, and each dimension of the space corresponds to one such unique word. Co-occurrence is determined by considering a window of a fixed number of characters before and after each instance of the word w in the training corpus. For example, a window of 1000 characters would include all words within 500 characters of an instance of w, both before and after.

A Word Space is built by taking every instance i of w in the training corpus. A vector representing i can be generated by considering all of the words in the context window of w along with their frequencies; the vector is non-zero in those dimensions which correspond to the words in the context window.

If the context of an ambiguous word is a good indicator of which sense it carries (this assumption is the basis for many WSD techniques), then the vectors associated with similar senses of w should have spatial locality in the Word Space for w. Vectors which are close to each other can be grouped into clusters, and the centroid of a cluster (the average of all vectors in the group) can be thought of as the "sense" for the cluster. Therefore, a small number of centroid vectors representing senses of w also exist in the Word Space of w.

WSD is accomplished by comparing the vector representing an instance of w in the test set to each of the centroid vectors determined through clustering, and assigning it the sense of the nearest vector, using cosine of the angle between the vectors as a metric. However, unless we assign a dictionary definition to each centroid vector as well, that w has been determined to carry the sense of a certain cluster means very little; we don't know what the cluster itself means! Schütze allowed assigning of real-world definitions to clusters by hand in his work with Word Space. Sense-tagged corpora can be used to automate this process of assigning definitions.

2.2. Structured Semantic Space

The Structured Semantic Space approach to WSD (Ji and Huang, 1997) is reminiscent of Word Space insofar as it involves creating context vectors and clustering them according to a similarity metric within an n-dimensional space. In the "sense space", however, similarity is measured with respect to the semantic categories of the context words rather than the context words themselves. Each of the n dimensions of the sense space corresponds to a semantic category, as defined in a dictionary resource such as Roget's Thesaurus. Additionally, a corpus tagged with semantic senses is required to construct the context vectors of monosense words, which outline the sense clusters in the space. The relevance of each particular semantic category c to the sense of the monosense word w is captured by a *salience* value, given by the formula:

$$Sal(c, w) = \frac{|\{w_i \mid c \in NC_i\}|}{k}$$

where NC_i is the set of all semantic codes for neighboring words of instance i of word w, and k is the total number of occurrences of w. Each unique w that appears in the corpus is therefore represented by a context vector of length equal to the number of semantic categories, where the c'th element of the vector is equal to the salience of semantic category c with respect to w:

$$cv_w = \langle Sal(c_1,w), Sal(c_2,w), \ldots, Sal(c_k,w) \rangle.$$

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

The similarity (distance) metric between two context vectors is defined as (1 - cos (cv1, cv2)), where cos(cv1, cv2) is the cosine of the angle between the two vectors. A tree-based algorithm that iteratively merges the most similar context vectors together is then used to partition the set of context vectors into a number of sense clusters. A sense cluster is characterized by its centroid.

Actual disambiguation of a word takes place in two steps. First, the words within the context window of an instance of an ambiguous word are used to create a context vector consisting only of 1's and 0's. This vector is compared to all sense clusters in the space using the distance metric described above, and all clusters within a certain threshold distance are "activated", or selected as candidates for the next step. Second, a context vector is created for each dictionary sense of the ambiguous word, based on the contents of a collocation dictionary. The distance between each dictionary sense vector and each activated cluster is calculated, and the sense minimizing the distance (maximizing similarity) is selected for the ambiguous word.

3. A Hybrid Approach

We describe a hybrid system combining features of the above mentioned methods with the following two goals in mind: 1) automatic assignment of real-world senses to sense clusters in Word Space, and 2) increased performance by employing Word Space and Semantic Space in parallel and combining the results of both methods, taking into account available confidence measures.

3.1. Automatic Tagging of Word Space Clusters

We present a method for implementing unsupervised tagging of Word Space clusters. The method requires a sense-tagged corpus, and simply involves assigning the sense with the highest representation in a cluster to that cluster. For any ambiguous word, the more consistent the mapping between senses and clusters in the Word Space, the more confidence we have in the disambiguation result. If the sense-tagged corpus is small, the clusters can first be generated from a larger, untagged corpus. The cosine similarity between context vectors for instances of each sense of the ambiguous word in the sense-tagged corpus and the cluster centroids is then computed, and the vectors are assigned to their closest clusters. We tally the number of times each particular sense was assigned to each cluster, and expect the tally to be high for only one sense per cluster, indicating that the cluster is representative of that sense.

We define a *representativeness* measurement for each sense s of ambiguous word w in cluster c, given by

$$R(c,s) = \frac{s_c / n_c}{s_t / n_t}$$

where s_c is the number of occurrences of sense s in cluster c, n_c is the total number of sense occurrences in c, s_t is the total number of occurrences of sense s in the corpus, and n_t is the total number of occurrences of w in the corpus. The numerator describes the ratio of sense s in cluster c, while the denominator normalizes according to the number of times s appears in the corpus. For word w_0, a representativeness value of 1 for sense s_0 indicates that the distribution of s_0 with respect to all senses of w_0 in cluster c_0 is the same as the distribution of s_0 with respect to all senses of w_0 in the entire corpus. Given that vectors are clustered by similar contexts, we assume that the more similar a cluster's sense distribution is to the sense distribution of the corpus, the less "unique" the context represented by the cluster is to its senses.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

Under this assumption, cluster c_0 provides no information for disambiguating sense s_0. Thus, representativeness estimates how reliable disambiguation based on a particular cluster will be. (For convenience's sake, we take the natural log of representativeness values in our system, shifting the value of neutral representation from 1 to 0. Positive representativeness will always mean a sense is well represented in a cluster, and negative representativeness will always mean the opposite. The characteristic representativeness of a cluster is its largest positive representativeness value. We test the utility of this measurement in our experiments.)

3.2. Improving Performance by Employing Word Space and Semantic Space in Parallel

The orthogonality of different WSD techniques suggests that using multiple methods will improve overall performance. Our approach is to apply both Word Space- and Semantic Space-based disambiguation in every disambiguation event. As Ji and Huang point out, a confidence rating is implicit in the Semantic Space distance calculated between word senses and activated clusters; if this distance is large, it is very unlikely that the system will select the correct sense.

Our intuition is that an analogous confidence rating is implicit in Word Space distance calculations as well. If the distance between a word's context vector and its potential sense clusters is large, or if the sense clusters are all more or less equidistant from the context vector such that no single sense is strongly preferred over the others, we should put less faith in the system's determination.

Intelligent consideration of these confidence measures in conjunction with the results of both disambiguation methods should allow the hybrid system to show improvement over each individual system.

4. Implementation

Our implementation of Word Space involves the following steps: First, we parse a corpus, searching for occurrences of words in a list of target ambiguous words. We build context vectors for each occurrence. Second, we reduce the dimensionality of the context vectors to 100 by means of Singular Value Decomposition in order to facilitate clustering by AutoClass, a program for clustering data by modeling the data as mixture of conditionally independent classes. (For comparison, the average unreduced context vector length in our experiments was 18145.) Next, we run AutoClass to generate clusters from the vectors of reduced dimensionality. The results tell us which vectors belong to which clusters; we use this information to compute the centroids of the clusters in the original space. Finally, to perform WSD on an instance of an ambiguous word we construct its context vector, find the cluster with the highest cosine similarity to it, and assign the most representative sense of that cluster to the word.

In order to test the performance of the Word Space classifier in the absence of a sense-tagged corpus, we use *pseudowords* (Schütze, 1992). A pseudoword is a set of two or more monosense words having different senses which we consider to be a single word with multiple senses. For testing purposes, pseudowords may then substitute for a sense-tagged corpus: for example, we can pick two words, 'house' and 'dog' and treat them as one word with two senses, a 'house' sense and a 'dog' sense.

To evaluate the Word Space component, we ran four different experiments, with (CITY, HOME), (FACE, WAR), (CHILDREN, HAND), and (EYES, SYSTEM) as our pseudowords. We selected only nouns under the assumption that they possess the most distinct context vectors. Our context window of choice was 1000 characters wide,

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

which Schütze found to be ideal in his experiments. We trained on the Brown Corpus and tested on a 1 million word WSJ corpus. Since word frequencies between the two corpora are significantly different, we also test using the Brown corpus, with the idea that it can hint at the "best case" performance potential of our system. The distribution of the pseudowords in the corpora is given in Table 1:

Pseudoword	Brown corpus	WSJ corpus
CITY/HOME	415 / 547	316 / 795
FACE/WAR	314 / 310	167 / 167
CHILDREN/HAND	372 / 419	220 / 103
EYES/SYSTEM	394 / 404	36 / 479

Table 1. Frequencies of pseudowords in corpora

We test the usefulness of the representativeness measure (section 3.1) by disregarding clusters in order of increasing representativeness value and noting the effect on precision. Finally, we look for a correlation between correctness of disambiguation and the distance from the ambiguous words' context vectors to the closest cluster centroids.

We implement Semantic Space in the following manner: First, we parse the public domain 1911 version of Roget's Thesaurus and create a database of semantic categories. Second, we map word-POS pairs that appear only under a single semantic category to the headers under which they appear, discarding all word-POS pairs which appear in multiple categories. These are the monosense words we are able to identify in a POS-tagged corpus. We then semantically-tag the Brown Corpus.

To build the Semantic Space, we follow the same procedure as described in section 2.2, with the exception that we choose to try AutoClass as our clustering method instead of the trie method described by Ji and Huang. To test disambiguation, we construct a pseudoword by selecting two or more words with distinct senses from the tagged, unambiguous words. Next, we can generate the equivalent of a collocation from the context of the selected words. Because this collocation is generated from a corpus which may not be representative of all contexts in which the words may appear, it may not be as general as a collocation taken from a collocation dictionary. However, we hope it adequately reflects the semantic nature of most contexts. If the content of the test corpus is of the same genre as that of the training corpus, we expect this to be the case. A larger and more representative training corpus may obviate this problem.

To simulate disambiguation of the pseudoword in a test corpus, we follow the same procedure as described in section 2.2. We search for occurrences of the pseudoword, using the semantic-category thesaurus to tag context words, and from these build either normalized Boolean or frequency vectors. We activate nearby sense clusters in Semantic Space with this context vector, and determine which word in the pseudoword set is closest to one of the activated clusters, according to the collocation for each such word found earlier.

Unfortunately, due to time and computer hardware limitations, we have to date been unable to obtain useful data from the Semantic Space component of our system.

5. Results

Tables 2, 3, 4, and 5 summarize the results of our experiments with regard to the representativeness measure. The first row shows the precision of the system taking into account all clusters; each successive row drops the cluster with the lowest representativeness value until only one cluster is left. The second column shows the results using the WSJ corpus; the last column shows the result using the Brown corpus.

The recall value for all experiments is 100%, because our system in its current form returns an answer for every instance of the

# clusters dropped	WSJ Corpus	Brown Corpus
0	.455856	.587929
1	.455856	.551509
2	.35045	.547347
3	.35045	.569199
4	.410811	.578564
5	.384685	.573361
6	.317117	.562955
7	.501802	.57232
8	.547748	.619147
9	.363063	.526535
10	.361261	.539022
11	.284685	.430801
12	.284685	.430801

Table 2. Precision results for Test #1

# clusters dropped	WSJ Corpus	Brown Corpus
0	.462462	.659905
1	.477477	.677804
2	.477477	.71957
3	.468468	.720764
4	.459459	.626492
5	.435435	.610979
6	.45045	.613365
7	.465465	.713604
8	.513514	.554893
9	.498498	.557279

Table 3. Precision results for Test #2

ambiguous word – no thresholding or other means of filtering results are currently employed.

Figure 1 shows the results with respect to the distance values for the experiments using the WSJ corpus in graphical form, while Figure 2 shows the results for the same experiments using the Brown corpus. Note that the values on the vertical scale are cosine similarity values; thus, a low cosine similarity value indicates a large distance.

# clusters dropped	WSJ Corpus	Brown Corpus
0	.313665	.635572
1	.31677	.631841
2	.319876	.646766
3	.60559	.656716
4	.590062	.661692
5	.590062	.655473
6	.639752	.619403
7	.649068	.61194
8	.680124	.468905
9	.680124	.468905
10	.680124	.468905

Table 4. Precision results for Test #3

# clusters dropped	WSJ Corpus	Brown Corpus
0	.680934	.818293
1	.363813	.858537
2	.363813	.859756
3	.344358	.858537
4	.392996	.868293
5	.441634	.871951
6	.929961	.510976
7	.929961	.510976
8	.929961	.510976
9	.929961	.510976
10	.929961	.510976
11	.929961	.510976
12	.929961	.510976

Table 5. Precision results for Test #4

6. Discussion

Generally, our results do not match the reported performance in Schütze's paper. We believe that this may be due to training data sparseness. Another reason for the low performance on the WSJ tests is the fact that we are testing on a different corpus than the one we are training on; the Brown and WSJ corpora might have sufficiently different types

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

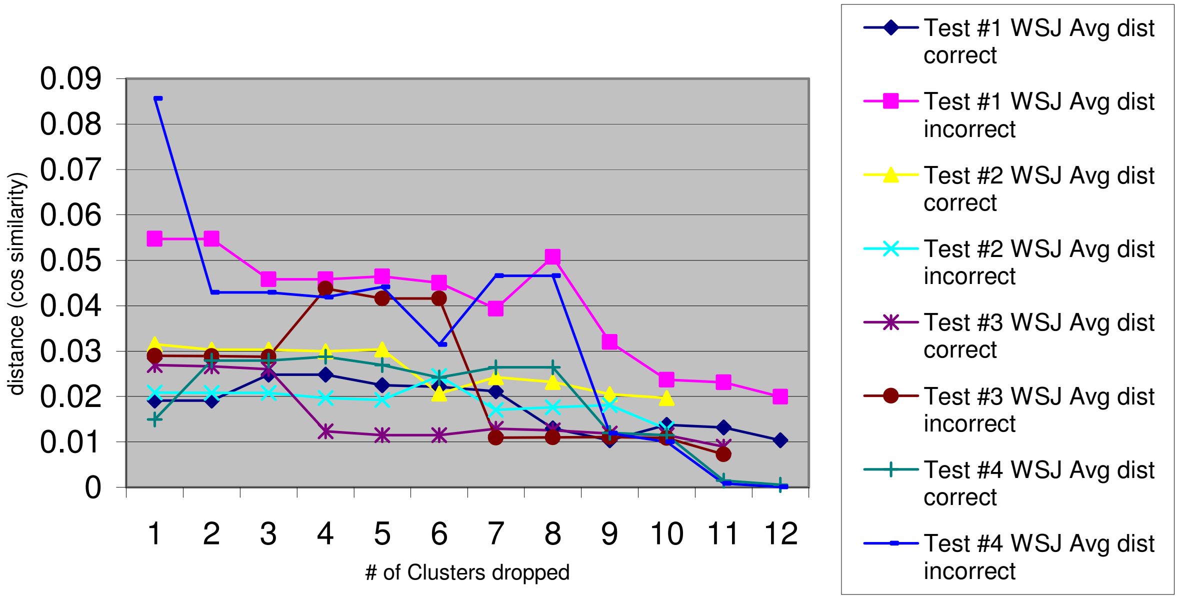


Figure 1. Distances for correct and incorrect classifications (WSJ corpus)

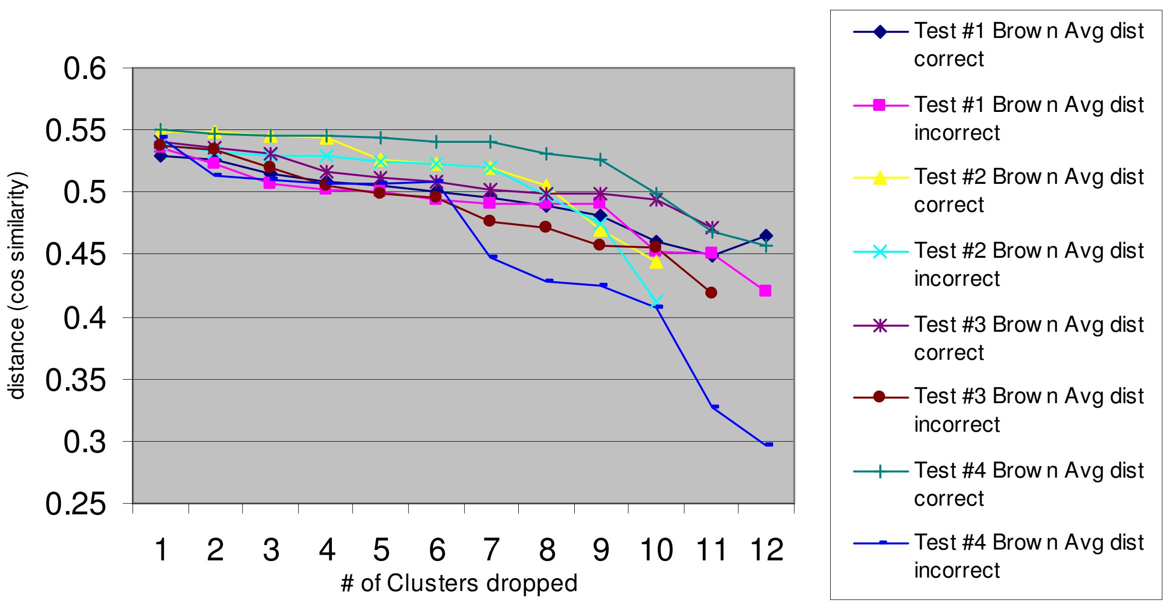


Figure 2. Distances for correct and incorrect classifications (Brown corpus)

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

of information that the context vectors are too dissimilar to produce good results. Nevertheless, despite the overall low performance, we wish to discuss several trends that we observed.

6.1. Representativeness

An interesting trend we observed is that in all four tests and with both testing corpora (with the exception of test #4 using the WSJ corpus; we provide an explanation of this anomaly later), the precision of our system is never at its peak with all clusters used. Instead, as we drop the first several clusters, a general trend of increasing precision sets in, leading up to the peak performance. A possible explanation is that because the dropped clusters have low representativeness values, they contribute little to word sense disambiguation. In fact, allowing these clusters to remain in the system impairs performance by "attracting" context vectors in their vicinities that otherwise would be assigned to sense clusters with higher representativeness values. As we drop even more clusters, we begin to lose important clusters and the system performance degrades.

Note that towards the end of each column, the precision values have a tendency to remain constant. This indicates that all remaining clusters lean towards the same sense; all instances of the ambiguous word are automatically assigned that sense, and the precision value obtained is identical to the ratio of that sense in the testing corpus. In the case of test #4, this leads to an absurdly high precision value of 93% when using the WSJ corpus for testing. Of course, no attention should be paid to these values.

As mentioned earlier, Test #4 using the WSJ corpus performs best when all clusters are considered. However, during the clustering phase of the Word Space, the most populated cluster turned out to be representative of the SYSTEM pseudosense. At the same time, this cluster's representativeness value was the lowest. We also recall that the WSJ corpus has a SYSTEM/EYES ratio of 479/36. Thus, the high initial precision can be attributed to the fact that the SYSTEM cluster described above very likely attracted many context vectors during the testing phase (since it attracted the most context vectors during the training phase), and since there were many more SYSTEM instances than EYES instances in the testing corpus, these taggings turned out to be correct. Once we dropped that cluster, some of these context vectors were assigned incorrectly to EYES clusters, thus lowering the performance.

Another exception to this trend is found in Test #3 using the WSJ corpus, which fails to exhibit the behavior of dropping precision as more clusters are dropped. This can be explained by two facts: that the highest representativeness clusters were all of the CHILDREN pseudosense, and. that CHILDREN appeared twice as many times as HAND in the test corpus.

Another interesting trend is that there seems to be some correlation between the points of highest precision when using the Brown corpus and when using the WSJ corpus. This suggests that the training corpus used to generate a Word Space can also be used to find the optimum cutoff point for dropping clusters and thus optimize actual disambiguation.

6.2. Distance value as confidence measurement

Figures 1 and 2 show that there is very little consistency in the cosine similarity values between correctly and incorrectly classified instances. The average cosine similarity of the correctly classified instances was greater than incorrectly classified instances in some cases (for example, test #2 in Figure 1), whereas in the other cases, surprisingly, the opposite was true (test #1 in Figure 1). In the Brown corpus results, the values were generally too close

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

together for any distinction between correct and incorrect classifications to be reasonable. We conclude that distance to the closest cluster is not a good confidence measure for results obtained from Word Space.

7. Future Work

Future work would of course entail completing our proposed hybrid system, followed by implementing a voting system between the Word Space and Semantic Space components of the system. Although we found the distance to the closest cluster in Word Space to be an unreliable confidence measure, perhaps the representativeness measure can in some way be used instead.

Performing additional experiments using new pseudowords would allow us test the validity of our interpretation of the relationship between the optimal cutoff point for dropping clusters in the training corpus and testing corpus.

Another way of using the representativeness measure could be to perform screening on disambiguation results. Instead of dropping clusters, we could set some minimum representativeness threshold. For disambiguation attempts that do not break that threshold, we do not return an answer, thus lowering the recall rate of the system. But since clusters with low representativeness values in general do not disambiguate well, we expect the precision to increase as a result of the thresholding.

We would also like to experiment on different languages with the hybrid system. Ji and Huang claim that the Semantic Space approach is language independent; we expect Word Space to be as well. We currently have the resources to perform tests in Chinese.

We have also discovered that the AutoClass clustering package generates some weight measures for each class found during the clustering process. This information can possibly be used to supplement existing confidence measures.

8. References

Hasan, Md Maruf; Lua, Kim Teng. Neural Networks in Chinese Lexical Classification. http://citeseer.nj.nec.com/8440.html

Ji, Donghong; Huang, Changning. 1997. Word Sense Disambiguation based on Structured Semantic Space. *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics.

Schütze, Hinrich. 1992. Dimensions of Meaning. In *Proceedings of Supercomputing '92, Minneapolis,* pages 787 – 796.

Yarowsky, David. 1992. Word-sense Disambiguation using Statistical Models of Roget’s Categories Trained on Large Corpora. In *Proceedings of COLING ‘92,* pages 454 – 460.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 46–54
Computer Science Department, Swarthmore College

A Minimally-Supervised Malay Affix Learner

Yee Lin Tan
Swarthmore College
Swarthmore, PA 19081
yeelin@cs.swarthmore.edu

Abstract

This paper presents a minimally-supervised system capable of learning Malay affixation. In particular, the algorithm we describe focuses on identifying *p*-similar words, and building an affix inventory using a semantic-based approach. We believe that orthographic and semantic analyzes play complementary roles in extracting morphological relationships from text corpora. Using a limited Malay corpus, the system achieved F-scores of 36% and 86% on prefix and suffix identification. We are confident that results would improve given a larger Malay corpus. In future work, we plan to extend our algorithm to include automatic discovery of morphological rules.

1 Introduction

1.1 Overview

There are over 18 million speakers of Malay in United Arab Emirates, the US, and southeast Asian countries such as Malaysia, Indonesia, Brunei, Singapore, Thailand, and Myanmar (Ethnologue, 2002). Malay uses both Roman and Arabic scripts, and belongs to the Western Malayo-Polynesian group of languages in the giant Austronesian family of over 1200 languages (Ethnologue, 2002).

1.2 Malay Morphology

Malay morphological processes include affixation (prefixal, suffixal, and infixal) as well as reduplication; however, prefixation is one of the most productive of these processes. There is a total of 21 prefixes in Malay (Tatabahasa Dewan, 1993) and the more common ones include *men-*, *pen-*, *ber-*, *se-*, *ter-*, and *di-*. (See Appendix for the full list.) With the exception of *men-* and *pen-*, prefixes typically do not result in changes to the stem. However, prefixes *men-*, *pen-*, and their allomorphs (which we will denote as *meN-* and *peN-* respectively), take on different forms depending on the initial letter of the stem. The allomorphs of the prefix *meN-* are *me-*, *mem-*, *men-*, *meny-*, *meng-*, and *menge-*. Similarly, the allomorphs of *peN-* are *pe-*, *pem-*, *pen-*, *peny-*, *peng-*, and *penge-*. The use of the allomorphs of *meN-*, which parallels that of *peN-*, is illustrated as follows:

(a) *me-* is typically used with stems that begin with the letters l, m, n, ng, ny, r, w or y. For example, *me-* + 'nanti' (wait) = '*me*nanti' (to wait).
(b) *mem-* is typically used with stems that begin with the letter b, or cognate verbs that begin with f, p, or v. For example, *mem-* + 'beri' (give) = '*mem*beri' (to give), and *mem-* + 'proses' (process) = '*mem*proses' (to process).
(c) *men-* is typically used with stems that begin with the letters c, d, j, sy, z or cognates that begin with the letters t or s. For example, *men-* + 'cari' (search) = '*men*cari' (to search), and *men-* + 'sintesis' (synthesis) = '*men*sintesis' (to synthesize).
(d) *meng-* is typically used with stems that begin with vowels, the letters g, gh, kh, or cognates that begin with k. For example, *meng-* + 'ambil' (take) = '*meng*ambil' (to take), and *meng-* + 'kritik' (critique) = '*meng*kritik' (to criticize).
(e) *meny-* is typically used with stems that begin with the letter s, which is dropped in the inflected form. For example, *meny-* + 'sumpah' (swear) = '*meny*umpah' (to swear).
(f) *menge-* is used with monosyllabic stems. For example, *menge-* + 'cat' (paint) = '*menge*cat' (to paint), and *menge-* + 'kod' (code) = '*menge*kod' (to encode).

Unlike prefixation, suffixation in Malay never results in a change to the stem. *-i*, *-an*, and *-kan* are the only three suffixes in the language. These suffixes can be attached to words to form nouns, verbs, and adjectives. For example, 'air' (water) + *-i* = 'air*i*' (to irrigate), 'lukis' (draw) + *-an* = 'lukis*an*' (drawing), and 'kirim' (send) + *-kan* = 'kirim*kan*' (to send). Suffixes can also be combined with prefixes. *peN-* and *ke-* are frequently combined with *-an* to form nouns, while combinations like *meN-...-i*, *meN-...-kan*, *di-...-i*, *di-...-kan*, *ber-...-kan*, and *ber-...-an* generally form verbs.

In addition to prefixes and suffixes, Malay has four infixes, namely *-el-*, *-em-*, *-er-*, and *-in-*. Compared to the other two affix categories, infixes are used relatively

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

infrequently since only a very small subset of words in Malay take infixes. The following are examples of such words: 'tunjuk' + *-el-* = 't*el*unjuk' (index finger), 'gilang' + *-em-* = 'g*em*ilang' (splendid), 'gigi' + *-er-* = 'g*er*igi' (serrated), and 'sambung' + *-in-* = 's*in*ambung' (continue).

Nested affixation occurs in Malay as well. Fortunately, unlike some agglutinative languages, no more than three layers of affixation is allowed in Malay. For example, the stem 'orang' (person) can be prepended with the prefix *se-* to form the word '*se*orang' (alone), followed by the layer *ke-...-an*, resulting in '*kese*orang*an*' (loneliness), and finally the prefix *ber-* to form the word '*berkese*orang*an*' (to suffer from loneliness). Similarly, the word 'kesinambungan' (continuity) can be decomposed to *ke-* + s + *-in-* + ambung + *-an*, in which the stem 'sambung' (continue) undergoes two layers of affixation.

Aside from nested affixation, reduplication is also common to Malay. Reduplication is the process of repeating phonological segments of a word. There are three types of reduplication in Malay, namely full and partial reduplication, and reduplication that results in a certain rhythmic phonetic change (Tatabahasa Dewan, 1993). The first two processes typically produce indefinite plurals and words that convey a sense of resemblance or homogenity, while the latter usually results in words that describe repetitive or continuous actions, heterogenity, and level of intensity or extensiveness. For example, 'pulau-*pulau*' (islands) results from the full duplication of the word 'pulau' (island) while '*se*siku'(triangle/drawing tool) results from the partial reduplication of the word 'siku' (elbow). Partial reduplication is not limited to the front segment of a word, as duplicated phonetic segments can be added to the end of a word as well (e.g. 'berlari-*lari*' and 'kasih-*mengasihi*'). In rhythmic reduplication, the entire stem is repeated but with phonetical changes that can either be a free phonetic change or involve rhythmic vowel and consonant repetition. The following examples illustrate the different types of rhythmic reduplication (Tatabahasa Dewan, 1993).
Vowel reduplication: 'sayur-*mayur*' (vegetables)
Consonant reduplication: 'gunung-*ganang*' (mountains)
Free reduplication: 'saudara-*mara*' (relatives)

1.3 Motivation and Goals

Automated morphological analysis can be incorporated into information retrieval systems as well as grammar and spell checkers. With the increase in the number of computer and internet users in southeast Asia, performance of such systems is becoming increasingly important. According to a 1999 International Data Corporation (IDC) report, internet users in the Asia Pacific region show preference for viewing the World Wide Web in their native language, especially when English is not their native tongue. Nevertheless, a recent check on Google revealed that there is still no option to limit a search query to only webpages written in Malay. Furthermore, while a Malay grammar and spell checker is currently available on Microsoft Word, a quick check showed that it does not catch errors that pertain to word order or incorrectly inflected words.

In this paper, we propose an algorithm that automatically induces a subset of Malay morphology. In particular, this algorithm takes as input a text corpus and produces as output an affix inventory of prefixes and suffixes. This system ignores infixes since they are not productive in modern Malay and their use is limited to a very small subset of words (Tatabahasa Dewan, 1993).

Although Malay is a reduplicative language, word reduplication will be ignored here as well since the goal of this system is to obtain an affix inventory for a highly prefixal language, not to perform a complete morphological analysis of Malay. The proposed algorithm can be used as part of the design of a complete morphological analyzer. Since Malay morphology is similar to that of Indonesian, this algorithm is likely to be portable to Indonesian as well.

2 Related Work

Most of the existing morphological analyzers focus on suffixal languages. With the exception of Schone and Jurafsky (2001), whose work we will describe in Section 2.1, few have considered prefixes, circumfixes, infixes, or languages that are agglutinative or reduplicative. Previous unsupervised morphology induction systems can be divided into two main categories based on whether the goal is to obtain an affix inventory or to perform a more comprehensive morphological analysis.

2.1 Morphological Analysis

Gaussier (1999) uses an inflectional lexicon to analyze derivational morphology. His system automatically induces suffixes by splitting words based on p-similarity, that is words that are similar in exactly the first p characters. Schone and Jurafsky (2000), on the other hand, extract affixes by inserting words into a trie, and observing places in the trie where branching occurs, an approach similar to identifying p-similar words. Using only the 200 most-frequent affixes, they generate a list of pairs of morphological variants (PPMVs). Their system then determines the semantic relationships between word pairs via Latent Semantic Analysis. Word pairs with high semantic correlations form conflation sets. Schone and Jurafsky (2001) extended their semantic-based algorithm to include orthographic and syntactic cues, and applied their algorithm to induce more extensive morphological relationships (prefixes as well as circumfixes) in German, Dutch, and English.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

2.2 Affix Inventories

Brent *et al* (1995), uses a Minimum Description Length approach to obtain suffixes that result in the maximum compression for any given corpus. DéJean (1998) uses an algorithm that exploits the entropy of the next character in a word. His algorithm decomposes a word into stem and suffix when the number of possible characters following a sequence of characters in a word exceeds a certain threshold. Like Brent *et al*, his goal, too, was to obtain an affix inventory using statistical methods.

2.3 Previous Work in Malay Morphology

Very little work on morphology induction has been done in Malay. The most recent work with regard to Malay morphology is an automated stemmer proposed by Tai *et al* (2000) as part of the design of an information retrieval system for Malay. In addition to a set of heuristics, their system is given a list of prefixes and suffixes along with an explicit set of rules under which affixes may be removed from words. Their overall goal is different from ours: Tai *et al* seek an efficient, but highly supervised stemming system, while we seek a minimally-supervised system that is capable of inducing affixation in Malay via semantic-based analysis. The output of our system may be used to eliminate the need for an explicit affix list that is required by their stemming algorithm.

3 Current Approach

We propose to extend Schone and Jurafsky's semantic-based approach to analyzing a highly prefixal, agglutinative language. Like Schone and Jurafsky (2000), our algorithm can be decomposed into four phases, namely (1) building an initial affix inventory, (2) identifying pairs of potential morphological variants (PPMVs), (3) computing semantic correlation of PPMVs, and finally (4) identifying valid affixes by selecting morphological variants with high semantic correlation. We use a text corpus consisting of news articles from an online Malaysian newspaper, and words from an online Malay dictionary.

3.1 Phase 1: Selecting Potential Affixes

In the first phase of analysis, we build two tries via in-order and reverse order insertion of words from the corpus along with their frequencies. Before describing the algorithm that extracts potential prefixes from these tries, we define the following terms:

(1) type count: Each distinct word in the corpus is considered a unique type. Hence, type count refers to the frequency of occurrence of a unique word.

(2) branching factor: When two or more p-similar words are inserted into a trie, branching occurs at the p-th node. For instance, in the reverse trie shown in Figure 2, branching occurs at the fourth node from the root since

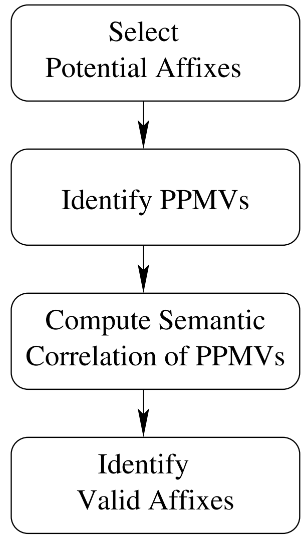


Figure 1: System architecture.

'cuba' (try), 'dicuba' (tried), and 'mencuba' (trying) are similar in exactly the last four characters. The branching factor is the number of branches that hang off a node in the trie. In the previous example, the branching factor at 'c' is 3.

To extract candidate prefixes from the reverse trie, the system needs to identify p-similar words. However, we believe that a constant p value is unsuitable for this task since errorneous splitting points may be proposed. Hence, we try to automatically induce an appropriate p value for different sets of words. To do this, we observe places in the trie where τ, the ratio between the branching factor and the type count is exactly 1. We call these places potential breaking points (PBPs). Splitting words into stem and affix when the τ ratio is 1 gives us an estimate of a suitable p value for any given subtrie.

Once a potential breaking point is identified, each candidate prefix that hangs off that PBP is checked for its overall frequency in the corpus. Only the T most frequent candidate prefixes, as determined by their frequencies in the forward trie, are selected as potential prefixes, and thus, added to the potential prefix inventory.

A reverse selection process is performed to determine potential suffixes. That is, candidate suffixes are identified from PBPs in the forward trie, and depending on their overall frequencies in the reverse trie, the system decides whether or not to add these candidate suffixes to the potential suffix inventory.

3.2 Phase 2: Identifying PPMVs

Pairs of potential morphological variants are constructed from words that descend from the same root node in the trie, share a common PBP, and contain a potential affix

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

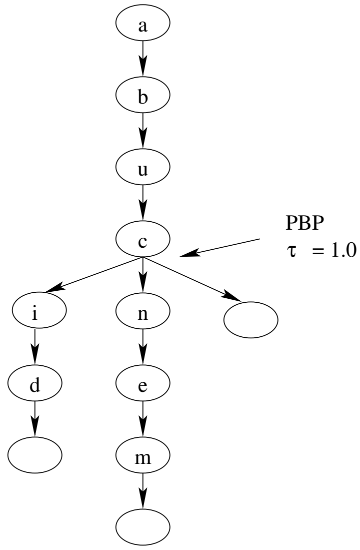


Figure 2: Structure of reverse trie with the words 'dicuba', 'mencuba', and 'cuba' inserted. The empty nodes represent end-of-word markers.

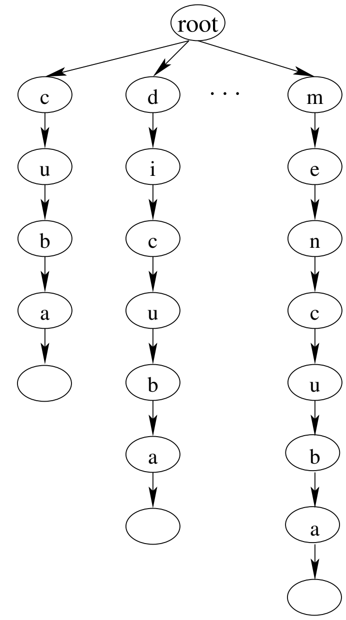


Figure 3: Structure of forward trie with the words 'dicuba', 'mencuba', and 'cuba' inserted.

in the initial inventory. For instance, if *di-*, *men-*, and *NULL* were candidate prefixes that were added to the inventory at the PBP shown in Figure 2, then the pairs of morphological variants would be ('dicuba', 'mencuba'), ('dicuba', 'cuba'), and ('mencuba', 'cuba'). The three affixes {*di-*, *men-*, *NULL*} form what we call the affix set for the stem 'cuba'. The same construction process is repeated to obtain PPMVs for words containing candidate suffixes.

3.3 Phase 3: Computing Semantic Correlation of PPMVs

Morphologically-related words frequently share similar semantics. Accordingly, we determine the validity of candidate affixes in the potential affix inventory by computing the semantic correlation of the PPMVs. The correlation score of each PPMV gives us an estimate of the validity of the two affixes it contributed to the initial inventory. For this purpose, we construct a co-occurrence vector with a ± 5-word window for each word in the PPMV using the corpus from Phase 1. We then compute the cosine of the angle between the two vectors using the standard formula:

$$\cos(\vec{v}_1, \vec{v}_2) = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1||\vec{v}_2|}$$

The dot product is the projection of one vector onto the other, and is thus a measure of the similarity, or more accurately, the co-directionality of two vectors. In view of this, the cosine of the angle between two co-occurrence vectors is commonly used as a measure of the semantic correlation of two words. Ideally, a pair of morphologically-related words would have a large dot product, and thus, a high cosine score.

3.4 Phase 4: Identifying Valid Affixes

Using the cosine scores of the PPMVs computed in the previous phase, we determine the "goodness" and "badness" of each candidate affix in the initial inventory. For every PPMV with a cosine score above the cosine threshold, *C*, we increment the "goodness" of the affixes corresponding to that PPMV by 1. Likewise, for every score below the threshold, the "badness" of the corresponding affixes is incremented by 1. For instance, assuming a cosine threshold of 0.2, the goodness of *di-* and *men-* corresponding to the PPMV ('dicuba', 'mencuba') from Table 1 will be incremented by 1 each. Similarly, the goodness of *men-* and *NULL* is incremented since the pair ('mencuba', 'cuba') has a cosine score greater than the threshold we defined earlier. However, the cosine score for ('dicuba', 'cuba') is below the threshold; consequently, the affixes *di-* and *NULL* will have their

PPMV	Cosine score
('dicuba', 'mencuba')	0.22
('dicuba', 'cuba')	0.19
('mencuba', 'cuba')	0.32

Table 1: Cosine scores of PPMVs formed from the stem 'cuba' and affixes in the set {*di-*, *men-*, *NULL*}.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

badness scores incremented by 1. The goodness and badness scores of each candidate affix in the affix set $\{di$-, *men*-, $NULL\}$ corresponding to the stem 'cuba' are summarized in Table 2.

A new inventory is constructed from candidate affixes in the initial inventory whose goodness scores are greater than or equal to their badness scores. From the previous example, both *di*- and *men*- would be considered valid affixes for the stem 'cuba', and hence, added to the new inventory.

Affix	Goodness	Badness
di-	1	1
men-	2	0
NULL	1	1

Table 2: Scores of affixes in the set $\{di$-, *men*-, $NULL\}$ corresponding to the stem 'cuba'. These scores were obtained using the validity heuristic described in Phase 4.

4 Results

4.1 Prefixes

Before Semantic Analysis: In order to determine a reasonable value for *T*, frequency thresholds were varied between 0 and 500 in increments 5 (with the exception of the interval between 35 and 40 in which *T* was incremented by 1), and proposed affix inventories were evaluated for recall and precision. Figures 4 and 5 summarize the results of this evaluation. Since we valued recall over precision in the initial phase, and did not wish to lose any correctly identified affixes prior to semantic analysis, we fixed *T* at 36. The later phases would serve to increase precision by eliminating incorrectly hypothesized prefixes. Thus, with a *T* value of 36, the system achieved 100% recall, 7.95% precision, and 14.74% on F-measure.

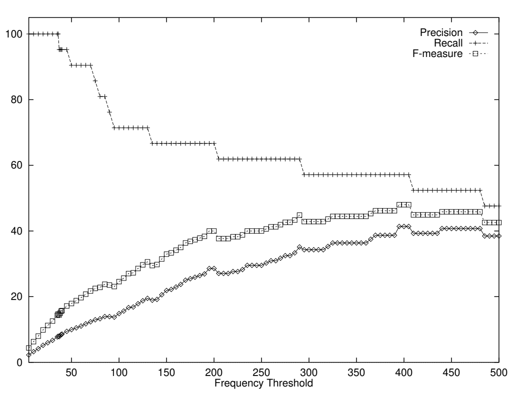


Figure 4: Precision, recall, and F-measure as a function of the frequency threshold, *T*, in the initial phase of prefix identification. Recall is highest for $0 \leq T \leq 36$.

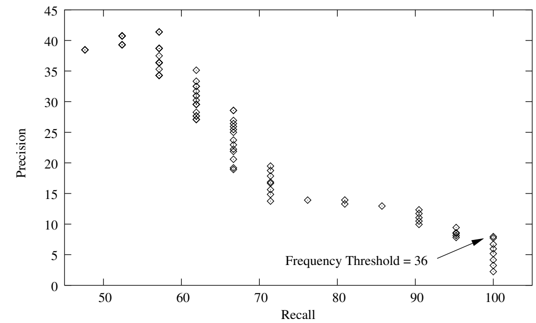


Figure 5: Precision versus recall for prefixes. At 100% recall, precision is highest at a frequency threshold *T* of 36.

After Semantic Analysis: Cosine thresholds were varied between 0.2 and 1.0, and the new prefix inventories were re-evaluated. Figure 6 shows that, at a cosine threshold *C* of 0.45, our system obtained 150.62% relative increase in precision but suffered 19.05% relative decrease in recall. The F-measure climbed 170% after semantic analysis.

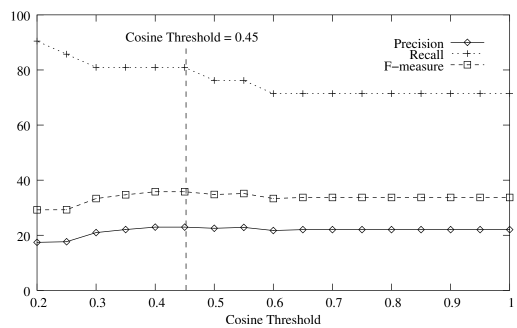


Figure 6: Precision, recall, and F-measure as a function of the cosine threshold, *C*, in prefix identification. F-measure is highest at $C = 0.45$.

4.2 Suffixes

Before Semantic Analysis: As a consequence of the low precision in prefix identification, our system did not attempt to remove prefixes from words in the corpus before they were reinserted into the tries for potential suffix selection, as suggested by Schone and Jurafsky (2001). To identify candidates for the initial suffix inventory, we employed the method described for prefixes, that is, we varied the value of the frequency threshold *T* between 0 and 2000 in increments of 5, and evaluated the proposed inventories for recall and precision. Figure 7 shows the evaluation results. The system achieved 100% recall, 60% precision, and 75% on F-measure for *T* values between 1500 and 1745.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

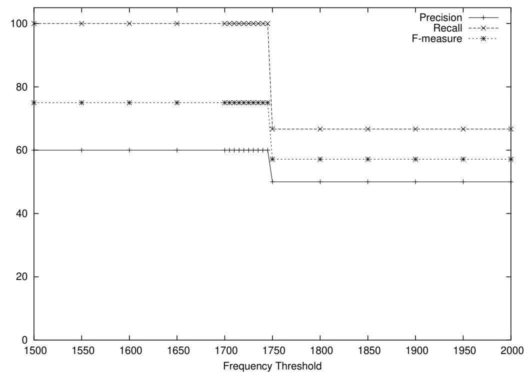


Figure 7: Precision, recall, and F-measure as a function of the frequency threshold, T, in the initial phase of suffix identification. At 100% recall, precision is highest for $1500 \leq T \leq 1745$.

After Semantic Analysis: The new suffix inventories that were obtained with cosine thresholds varied between 0.2 and 1.0 were re-evaluated as described in section 4.1. At a cosine threshold of 0.65 (see Figure 8), the system was able to achieve 80% precision and 10.71% increase in F-measure while maintaining recall at 100%. The identified suffixes were -*i*, -*an*, -*kan*, and -*a*. Of these, the first three were correct.

Table 3 provides a summary of the precision, recall and unweighted F-scores obtained by the system before and after semantic analysis.

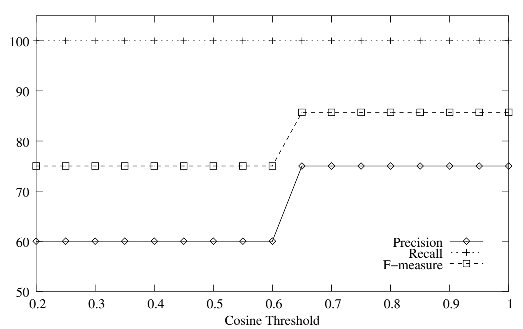


Figure 8: Precision, recall, and F-measure as a function of the cosine threshold, C, in suffix identification. F-measure is highest for $0.65 \leq C \leq 1.0$.

		Prefix	Suffix
Recall	Before	100.0	100.0
	After	80.95	100.0
Precision	Before	7.95	22.97
	After	22.97	75.0
F-measure	Before	14.74	75.0
	After	35.70	85.71

Table 3: Summary of precision, recall, and unweighted F-measure before and after semantic analysis.

5 Discussion

5.1 Corpus Size

The results from prefix identification were rather disappointing. However, we believe that this is not a shortcoming of our algorithm, but rather of the training corpus itself. While it is typically easy to find unlabelled corpora in English, building a large corpus in Malay proved rather difficult. Our corpus contains just over 22,000 unique words, compiled from two online sources: a Malaysian newspaper and a Malay dictionary. Since English is widely spoken in Malaysia, English words frequently find their way into Malaysian news articles, and thus our corpus. Extending the news corpus to include entries from the dictionary increased the number of prefixes found in the initial phase, but doing so presented a significant problem in the semantic analysis phase because many of the words from the dictionary were not found in the news articles.

Although the results from suffix identification could improve with a larger corpus as well, the size of the training corpus was not an issue in the case of suffixes. This is because the size of the corpus relative to the number of suffixes the system had to identify was approximately 7,000 to 1, while the ratio was only 1,000 to 1 in the case of prefixes.

5.2 Potential Issues with Validity Heuristic

While the validity heuristic employed in Phase 4 generally works well, it has potential issues that are not addressed in current work. A problem arises when there are many spurious PPMVs associated with a given stem. Consider the addition of *s*- to the affix set {*di*-, *men*-, *NULL*} in our earlier example. We present the corresponding PPMVs and their cosine scores in Table 4.

PPMV	Cosine score
('dicuba', 'mencuba')	0.22
('dicuba', 'cuba')	0.19
('mencuba', 'cuba')	0.32
('dicuba', 'scuba')	< 0.01
('mencuba', 'scuba')	< 0.01
('cuba', 'scuba')	< 0.01

Table 4: Cosine scores of PPMVs constructed from the affix set {*di*-, *men*-, *NULL*, *s*-} and the stem 'cuba'.

As mentioned earlier, our corpus has the potential of containing English words, thus there is a chance that a

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

word like 'scuba' may appear in the training corpus. Because of the presence of spurious PPMVs (due to the addition of *s*-), the hypothesized badness of each affix in the original set {*di*-, *men*-, *NULL*} has increased. In Table 5, we list the new goodness and badness scores with the addition of candidate prefix *s*-.

Affix	Goodness	Badness
di-	1	2
men-	2	1
NULL	1	2
s-	0	3

Table 5: Validity scores of candidate affixes in {*di*-, *men*-, *NULL*, *s*-}, assuming a cosine threshold of 0.2.

Although few instances like this were encountered in our current work, it is conceivable that such a problem would be significantly detrimental to our system's performance, especially in future work when a larger corpus is used. A more robust solution would be to compute the goodness and badness of each candidate in the affix set, remove any affix with a goodness score of 0, and then recompute the validity of each affix in that set by decrementing each of their badness scores by 1.

With *s*- removed, and the badness scores recomputed, the validity of *di*-, *men*-, and *NULL* would be restored to their original values as shown in Table 2.

This method of determining affix validity suffers from another drawback in that it would incorrectly identify affixes as invalid if there is a partition within an affix set associated with a given stem. A partition exists in an affix set if the PPMVs that are constructed from those affixes belong to two disjoint, morphologically-unrelated sets. Although we did not find an example like this in the Malay corpus, such a phenomenon occurs in languages like French. Consider the two verbs 'fonder' and 'fondre' whose simple past forms are {'fondai', 'fonda'} and {'fondis', 'fondit'} respectively. On seeing these four inflected words, our system would propose {*-ai*, *-a*, *-is*, *-it*} as the affixes associated with the stem 'fond'. The problem arises from the fact that the four words 'fondai', 'fonda', 'fondis', and 'fondit' belong to two morphologically-unrelated sets. Consequently, our validity heuristic would propose the scores shown in Table 6. Since none of the affixes have goodness scores greater than or equal to their badness scores, all of them would be erroneously discarded by our algorithm.

Fortunately, this phenomenon rarely occurs in most languages. Even in cases where it does, it is highly likely that the affixes, which are mistakenly discarded by the system, would be associated with other stems that do not suffer from the same problem.

Affix	Goodness	Badness
-ai	1	2
-a	1	2
-is	1	2
-it	1	2

Table 6: Validity scores of suffixes from the example in French.

5.3 Unsupervised Selection of Thresholds

Although the values of the frequency and cosine thresholds in this experiment were hand-picked to obtain the best results, these values can be obtained automatically. The following is a potential algorithm for doing so:

(1) Set the frequency threshold T to a reasonably small number, say, 5, in order to eliminate potential typos as well as the possibility of foreign words in the corpus.
(2) Run Phase 1 with T = 5 to obtain an initial affix inventory, I.
(3) Build a vocabulary of all distinct words in the corpus. Attach each affix $a \in I$ to each word in the corpus. Check if we still have a valid word in the vocabulary. If we do, add a to the new inventory, I'.
(4) Next, run Phase 2 for each affix in I'.
(5) Now, run Phase 3 with varying cosine thresholds, starting at 0. With each different threshold, check to see if we have lost any affix in I'. Increase the threshold as long as we have 100% recall on the affixes in I'. Save the cosine threshold C' prior to the drop in recall on I'.

C' should give us a good estimate of the optimal cosine threshold for the initial inventory I. Since I' is a subset of I, we are guaranteed that recall on the affixes in I would drop before C'. Having estimated the value of the cosine threshold, we can now return to running Phase 2 with I, and Phases 3 and 4 with a cosine threshold of C'.

6 Conclusion

Despite relatively disappointing results, we are confident that this algorithm would be more successful on a larger corpus. Being one of the first systems built to analyze Malay affixation, this system shows promise of analyzing highly prefixal languages. More importantly, this system provides a starting point for future work in Malay morphological analysis.

Acknowledgements

Many thanks to Richard Wicentowski and five anonymous reviewers for useful suggestions and comments on a first version of this paper.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

References

Michael R. Brent, Sreerama K. Murthy, Andrew Lundberg. 1995. Discovering Morphemic Suffixes: A Case Study In MDL Induction. *Proceedings of 5th International Workshop on Artificial Intelligence and Statistics*.

Hervé DéJean. 1998. Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora. *Workshop on Paradigms and Grounding in Natural Language Learning*.

Ethnologue Report for Language Code: MLI. 2003. Ethnologue. *http://www.ethnologue.com/*.

Éric Gaussier. 1999. Unsupervised Learning of Derivational Morphology from Inflectional Lexicons. *ACL '99 Workshop Proceedings: Unsupervised Learning in Natural Language Processing*, University of Maryland.

Asia/Pacific's Internet Users Demand Localized Web Content, IDC Finds. 1999. Techmall. *http://www8.techmall.com/techdocs/TS991104-4.html*

Nik S. Karim, Farid M. Onn, Hashim Hj. Musa, Abdul H. Mahmood. 1993. *Tatabahasa Dewan* Edisi Baharu. Dewan Bahasa dan Pustaka, Kuala Lumpur, Malaysia.

Patrick Schone, Daniel Jurafsky. 2000. Knowledge-Free Induction of Morphology Using Latent Semantic Analysis. *Proceedings of the Computational Natural Language Learning Conference*.

Patrick Schone, Daniel Jurafsky. 2001. Knowledge-Free Induction of Inflectional Morphologies. *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Sock Y. Tai, Cheng S. Ong, Noor A. Abdullah. 2000. On Designing an Automated Malaysian Stemmer for the Malay language. *Proceedings of the 5th International Workshop International Retrieval with Asian Languages*.

Appendix: Affix List

Affix Type	Nouns	Verbs	Adjectives
Prefix	pe- pem- pen- peng- penge- pel- per- ke- juru-	me- mem- men- meng- menge- memper- di- diper- bel- ber- ter-	ter- se-
Suffix	-an	-kan -i	
Prefix and Suffix Combination	pe-...-an pem-...-an pen-...-an peng-...-an penge-...-an per-...-an pel-...-an ke-...-an	me-...-kan mem-...-kan men-...-kan meng-...-kan menge-...-kan me-...-i mem-...-i men-...-i meng-...-i menge-...-i memper-...-kan memper-...-i di-...-kan di-...-i diper-...-kan diper-...-i ber-...-kan ber-...-an ke-...-an	ke-...-an
Infix	-el- -er- -em-		-el- -er- -em- -in-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 55–62
Computer Science Department, Swarthmore College

code-name DUTCHMAN: A Text Summarization System

Erik Osheim
Swarthmore College
osheim@sccs.swarthmore.edu

Daniel Sproul
Swarthmore College
sproul@sccs.swarthmore.edu

Abstract

Text summarization is an interesting and challenging problem in natural language processing, and one that has numerous potential applications in the realm of data-mining, text searching, and information retrieval. We have implemented a summarization system appropriate for articles and technical texts.

The system, code-named DUTCHMAN, attempts to identify which sentences in the document are most appropriate for inclusion in a summary based on analysis using key noun-phrases. The system employs WordNet in order to extend the notion of key phrases to key concepts.

The system, in its current instantiation, only achieves mediocre results, but our work does suggest some promising avenues for future research in text summarization.

1 Introduction

The general problem of text summarization is very broad and difficult: Given some document of arbitrary length, can we produce a second document of roughly constant length (ie. a few sentences) that conveys the general meaning of the original? How can we process a text to determine what the text is about, and then reformulate that information to produce a viable summary? Certainly, humans are able to do this, but this is typically contingent on our ability to not only parse and read, but also *understand* the document in question. Thus, to fully address text summarization in general, we would need to first solve a large number of difficult and currently unresolved natural language processing and artificial intelligence problems.

One option would be to use a knowledge base to identify semantic facts and topics in a document; without fully solving things like the symbol grounding problem, we can still hope to *understand* the text's subject. Summarizers which take this approach are known as symbolic summarizers. However, there are some difficulties with taking a heavily symbolic approach. First, since it requires a large knowledge base in order to function, the results do not generalize across languages well. Second, symoblic approaches are especially vulnerable to the depth vs. robustness trade-off. Simply put, systems that are created to analyze a certain type of document can restrict themselves to that domain, allowing them to make more assumptions about the source texts and thus perform better, at the expense of generality (Hovy, 2000). Since symbolic summarizers have to make a lot of assumptions about the text's content, they tend to do especially well when they can specialize; however, this makes a general summarization tool difficult to implement symbolically. Theme and topic recognition (two common methods of symbolic summarization) are staggeringly complex in the most general cases (Mani, 1998).

Fortunately, in certain domains, documents tend to contain sentences which are self-summarizing. For example, journal and newspaper articles, and technical documents, tend to begin with, and, more generally, contain sentences which address the purpose and nature of the document as a whole. We cannot expect this sort of sentence to be found in certain other domains, for example fiction, where no part of the text can be expected to pertain to the text as a whole. Many text summarization systems (eg. (Barker, 1998), (Szpakowicz, 1996)) choose to adopt such a restricted domain, and thus are able to exploit the self-summarizing nature of such documents.

Within this restricted domain, we can reformulate the problem of text summarization as follows: How do we select the best sentences for inclusion in a summary, and what do we do with these sentences after we have selected them? We have based our work on the work of the Text Summarization Group at the University of Ottowa

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

Department of Computer Science (Barker, 1998). Their general method involves identifying key noun phrases within a document, and then applying various heuristics to weight sentences based on key phrase frequency, then just concatenating the sentences in order to produce a summary.

Our text summarization system, code-named DUTCHMAN, is structured similarly, but we have extended the key phrase analysis to a form of conceptual analysis based on WordNet, allowing us to increase the emphasis placed on certain key phrases which are representative of general concepts in which other key phrases in the document participate. For example, in a paper about engines, given that *engine*, *camshaft*, and *piston* are all key phrases, the salience of the word *engine* will be increased, because *camshaft* and *piston* are both parts of engines.

2 Related Work

Due to renewed intereset in text summarization, several conferences have recently addressed the problem. From these talks, it is obvious that researchers somewhat divided over the best methods of text summarization. While many researchers favor statistical approaches similar to the one pursued in DUTCHMAN, there are also symbolic summarizers, which place more weight on trying to find important topics through world-level concepts (Hovy, 2000). These systems try to identify an underlying topic (or topics) before ranking phrases and sentences on their score. (**?**) In this context, DUTCHMAN is a statistical summarizer which utilizes symbolic information (via WordNet) in an attempt to improve its statistically generated keywords.

Most other projects that use symbolic information do so before their statistical processing, or do the two forms of processing independently and then attempt to integrate the results ((Szpakowicz, 1996), (Mani, 1998)). However, there are many varieites of symbolic summarizers; its unclear what the best use of ontologies is, especially given the depth/robustness trade-off. Some examples of symbolic summarization methods from the TIPSTER conference are:

- use a graph of theme nodes linked via a custom thesaurus (CIR).
- use sentences determined to be about frequently mentioned individuals via co-reference resolution (Penn)
- use morphological analysis, name tagging, and co-reference resolution to weight sentences (SRA)

Ad hoc summaries (undirected summaries like the kind DUTCHMAN generates) only comprise some of the goal of summarization systems. Most systems also support interactive summarization (after being questioned, i.e. (Mani, 1998) and (Szpakowicz, 1996)), and topic-specific summarization (summarization with regard to specific set of interests, i.e. terrorist activity (Mani, 1998)). These systems serve different purposes, but most summarization methods can be used fairly effectively in any of the realms. (Mani, 1998)

3 DUTCHMAN Base System

As noted, our general approach is to identify key noun phrases within a document which indicate that the sentences in which they participate might be relevant summary sentences, ie. might contain content which is relevant to the overall meaning of the text.

Given an input text, our basic algorithm is as follows:

1. Split the document into sentences
2. Apply part-of-speech tagging, text-chunking, and noun-phrase identification
3. Identify key noun-phrases based on frequency
4. Select sentences based on key phrase frequencies
5. Concatenate sentences to generate the final summary

Lacking sophistacated sentence splitting tools, DUTCHMAN currently relies on a simple and imperfect script to handle sentence chunking.

POS tagging, text chunking, and NP identification are accomplished using the pre-trained fnTBL rule sets which are distributed with the fnTBL system (fnTBL is a freely-distributed rule-based machine-learning tool commonly employed in natural language processing tasks; TBL stands for Transformation-Based Learning) (Florian, 2001).

The remainder of the base system was implemented using Python. After sentence chunking and NP identification, we construct a database of frequencies for each noun-phrase in the document. In addtion to noun-phrases identified by fnTBL, we also add to the database individual nouns and noun-phrases connected by prepositional phrases; in this manner, given the string "King of Prussia Mall",rather than merely adding "King" and "Prussia Mall" to the database, we also add "Mall" and "King of Prussia Mall",which are, one might imagine, the truly salient key-phrases for the document in question.

We then identify 10 noun-phrases as "key phrases",ie. phrases whose sentence content is likely to pertain to the overall meaning of the document, and thus make good summary sentences. In the base system, the key-phrases are chosen based purely on which noun-phrases have the greatest frequencies. The scores for noun-phrases which are contained in larger noun-phrases (eg. "King" is contained in "King of Prussia") are discounted somewhat by

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

the scores of their containing phrases. We used 10 keyphrases because, given the length of our test documents, this tended to cover about 10-25implementation might include dynamic selection based on a fixed target percentage, but DUTCHMAN does not currently support this.

Each sentence is then scored based on the summed weights of the key phrases in the sentence. When a key phrase occurs more than once within the same sentence, an interesting issue arises. The most obvious approach would be to simply multiply by the key phrase's count in the sentence, but the problem with this is that we would like to in some manner reward diversity of key phrases for our summary sentences. Thus, a sentence which contains a single key phrase twice ought to, on average, fair poorer than a sentence which contains two distinct key phrases once each. We accomplish this by multiplying by the square-root of the count rather than just the count; thus, additional instances of a key phrase increase the score of a sentence, but always by an amount less than the prior instance. The resulting scoring equation is as follows:

$$\mathrm{score}(S) = \sum_{w \in S} \mathrm{weight}(w) \cdot \sqrt{\mathrm{freq}(w)}$$

The final summary is then generated by selecting the three highest-scoring sentences in the text, and concatenating them in the order in which they occur in the text. We found that since our algorithm tended to pick longer sentences from the text, choosing the best three tended to produce summaries which had fairly varied content, but with brevity. While the longer sentences naturally tend to score higher, it is still a useful result, as longer sentences are often used to link various indepentently occuring ideas within a text.

4 Key-Concept Analysis

We refine our key phrase analysis by generalizing to a notion of key concepts. Within a given text, many of the key phrases will in some manner be related to a similar concept. For example, in an article about engines, both pistons and camshafts are parts of an engine, and thus can be said to participate in the concept *engine*.

In order to implement our key concept analysis, we employed WordNet. WordNet is an ontology; it contains information linking words in the English language. It stores many different types of relationships, such as hypernymy, holonymy, synonymy, and sense. Langauge processing systems which take advantage of WordNet have information about words and language that is fundamentally richer than those that do not (Miller, 1998).

After experimenting with WordNet, and creating a prototype summarization system without it, we found that the only two relations which seemed to provide us with useful information for summarization were hypernymy and holonymy (-hypen and -hholn). Hypernymy identifies hierarchical "is a" relationships (thus a query for "tabby" might return "tabby IS cat IS mammal IS animal IS organism IS entity"), whereas holonymy returns a variety of containment relationships, eg. "is a" or "part of" in a non-hierarchical fashion (thus a query for "tabby" might return "tabby IS A cat", whereas a query for "piston" might return "piston PART OF reciprocating engine"). In order to facilitate interfacing DUTCHMAN with WordNet, we implemented a WordNet subsystem which we termed FERNANDO.

Because our summarizer is not generative, there was no good way to take advantage of noun phrases which WordNet found to be related to the article, but which did not appear in the article. Therefore, we use the WordNet analysis to reconsider the weights in the noun-phrase frequency database, giving added weight to those noun-phrases which represent concepts in which other noun-phrases in the document participate. Thus, if the words "cat", "tabby", and "calico" all appear in a document, the score for "cat" would be increased because both "tabby" and "calico" are identified as being kinds of "cat". We then select the 10 most salient key phrases based on the adjusted weights.

This modified algorithm is reflected by adding an extra step in relation to the base-system algorithm:

3a Use WordNet to modify key noun-phrase values based on key-concept analysis

Our algorithm generates a tree of noun phrases for each keyphrase. It then weights each noun phrase in the tree based on the number of keyphrases in whose trees it participates, and how directly it is linked to them. We wish to favor relationships which are closer to the source words, thus given the example "tabby IS cat IS mammal IS ...", if both "cat" and "mammal" occur in the document, we wish to increase the score for "cat" more than the score for "mammal",because we are seeking to achieve the correct level of generalization which encompasses the salient noun-phrases in the document and yet still addresses the meaning of the document as a whole. An article might contain "tabby","calico", "siamese", etc., and thus is most likely about cats and not about mammals. However, an article which contains not only those words but also "labrador" and "squirrel" is more likely about mammals; here, despite the fact that the score for "cat" was increased more than the score for "mammal" by the various types of cat, all words which are kinds of mammal contribute to "mammal",so in most cases "mammal" will win out over "cat".

For each noun-phrase considered in WordNet analysis, we must now compute a score offset. In essence, we need a decaying distance function for the relevant internode distances, which we achieve with a decaying exponential.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

	Average Score		
Document	*Random*	*no WordNet*	*WordNet*
ADA	1.2	3.6	2.4
SAUDI	1.0	4.4	2.8
GW	2.6	3.6	2.8
ENGINE	1.2	1.8	1.6
PIRATE	1.6	4.0	2.8
Average	1.52	3.48	2.48

Table 1: Human-assigned summary scores

	Random	
Document	*Precision*	*Recall*
ADA	25%	30%
SAUDI	13%	10%
GW	36%	50%
ENGINE	0%	0%
PIRATE	33%	60%
Average	21%	30%

Table 2: Baseline Precision and Recall

	no WordNet		*with WordNet*	
Document	*Precision*	*Recall*	*Precision*	*Recall*
ADA	57%	90%	61%	90%
SAUDI	42%	100%	50%	100%
GW	31%	40%	23%	40%
ENGINE	63%	40%	48%	40%
PIRATE	45%	50%	43%	40%
Average	48%	64%	45%	62%

Table 3: Precision and Recall for test documents

To determine the score offset for each noun-phrase N, we then sum over each considered noun-phrases n, for each adding its originally computed frequency weight times the decaying distance function:

$$\Delta\text{score}(N) = \sum_{n \in \text{noun-phrases}} \text{freq}(n) \cdot \alpha^{\text{distance}(N,n)}$$

where α, a constant, was empirically chosen to be 0.7, a value which helped acheive the aforementioned desired level of concept generalization.

5 Results

One of the inherent difficulties of the text summarization problem is that it is rather difficult to evaluate quantitatively. What makes a summary "good" varies from person to person and from document to document. Nonetheless, some attempt can be made to evaluate the quality of a summary.

We selected a set of five test documents: an article about the Americans with Disabilities Act (ADA), a text regarding a Gulf-War era Iraqi occupation of a Saudi Arabian town (SAUDI), a brief biography of George W. Bush (GW), an excerpt from a text about engines (ENGINE), and a brief article about pirates (PIRATE). For each, we generated summaries both with and without using FERNANDO. In addition, in order to establish a baseline for our system, we generated summaries based on purely random selection of sentences.

Our first evaluation scheme involved getting a group of human evaluators to score each summary on a scale from 1 (bad) to 5 (good). The results of this evaluation are displayed in Table 1. It is sadly apparent that FERNANDO seems more to detract than add to the quality of a summary, but nonetheless both are notably better than the results acheived by random selection.

Our second evaluation scheme involved using precision and recall metrics. For each document, human evaluators identified a list of what they felt were the ten most relevant key noun phrases, and then each summary was scored for precision and recall of this list. We calculated precision as the percentage of nouns in the summary which were in the key phrase list; in this manner, we prevent the possible favoring of gibberish sentences like "Engine engine engine." Recall was simply the percentage of words in the list which were contained in the summary. In the text summarization domain, a good recall score will indicate that a summary addressed the major content elements of the document, whereas a good precision score indicates that a summary is targeted and concise. Arguably, recall is a more important metric than precision in this domain, but both convey meaning regarding the quality of a summary. The baseline (random summary) results are displayed in Table 2 and the actual summary results are displayed in Table 3.

6 Discussion

On the surface, it appears as if incorporating WordNet into our system has made it slightly worse rather than better, as we get the same recall but, on average, slightly worse precision. However, the engine and George W. Bush texts presented unique challenges to summarization, being that the engine article contained many lists of engine parts and very few summary-relevant sentences, and the Bush text was just very short, which meant that there were not enough key words present to make our WordNet analysis particularly meaningful. This suggests that perhaps the size of our keyword set needs to be allocated dynamically based on document length rather than constant. Also, in neither of the two problem cases did the sentences in the article have any real unifying themes, other than a very shallow description ("biography of George Bush", or "Mechanic's Textbook") which was not actually present in the text. Thus, our use of WordNet depends upon the assumption that the general

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

concepts relating the keyphrases actually be relevant to the summary.

On a more qualitative level, the no WordNet vs. WordNet summaries tended to be similar, and in the Saudi and ADA cases the WordNet ones provided more thorough detail, according to the human readers. Thus, despite the disappointing figures, analysis with WordNet did seem to yield some positive results.

7 Future Improvements

It would be interesting to test the human readers to see which documents they believed would be easier or harder to summarize, and compare those figures to our precision and recall figures for summarization with and without WordNet. Based on the articles and summaries that we have seen, we would guess that the articles which were found to be more easily summarizable by human readers would be the ones that the WordNet-aided summarization system would do best on.

DUTCHMAN lacked pronoun resolution, which severely hindered its performance. Since most sufficiently complicated ideas will span multiple sentences, and subsequent references of salient noun phrases are typically substituted for pronouns, pronoun resolution is key to derivative summarization (creating a summary directly out of excerpts from the text). Thus, a system with pronoun resolution could see a signifigant jump in its effectiveness. Additionally, DUTCHMAN lacked robust sentence splitting utility, and was thus often forced to deal with sentence fragments rather than whole sentences. Incorporating a more viable sentence splitter would no doubt increase DUTCHMAN's performance as well.

Another direction would be to use WordNet on all the noun phrases instead of just the statistically signifigant ones. It seems like concept-webs such as were used in the CRI summarizer might be an interesting way to augment our statistical data (Mani, 1998). As was remarked earlier, we did not find examples of summarizers that did symbolic analysis on a statistically selected subset, and this could explain FERNDANDO's confusing inability to help DUTCHMAN.

Future tests would probably have to define a narrower type of text to summarize; as we discovered, ontological assumptions about content which were valid for certain articles were invalid for others– in particular, it doesn't seem like biographies or instructional texts tend to yeild to the same techniques as explanatory articles, which are written with a more specific goal in mind. A larger testing set, with a narrower range of article types, and a broader base of human readers, with statistics on how well the humans believed they summarized the articles, and comparisons of the sets of human-identified keywords, would all aid in evaluating a summarizer.

8 Conclusion

Code-name DUTCHMAN is a fairly reasonable text summarization system, for which further fine-tuning would no doubt produce better results. Our addition of keyconcept analysis using WordNet has proved helpful in some subjective cases, and further refinement of this technique, combined with other uses of WordNet, could facilitate the production of better summaries.

It is not clear whether methods that generate summaries out of excerpts can overcome all difficulties. Since the technique is limited by the quality of summarization-grade sentences in the document, it will never be perfect for certain types of documents. This is a problem that non-productive summarizers have regardless of whether they are statistical or symbolic. Many summarizations, such as the popular Cliff Notes series, are designed to do more than just abbreviate the text, but to paraphrase and explain; it would be desirable to have a summarizer that could do this. However, we do not have any belief that a system like ours could function in this way without radical modifications.

References

Radu Florian and Grace Ngai. 2001. *fnTBL*. Johns Hopkins University, Baltimore, MD

Eduard Hovy, Chin-Yew Lin, Daniel Marcu. 2000. *SUMMARIST: Automated Text Summarization*. http://www.isi.edu/natural-language/projects/SUMMARIST.html

Jade Goldstein, Mark Kantrowitz, Vibhu Mittal, and Jaime Carbonell. 1999. *Summarizing Text Documents: Sentence Selection and Evaluation Metrics*. Carnegie-Mellon University and Just Research, Pittsburgh, PA

Inderjeet Mani, David House, Gary Klein, Lynette Hirschman, Leo Obrst, Therese Firmin, Michael Chrzanowski, Beth Sundheim. 1998. *The TIPSTER SUMMAC Text Summarization Evaluation*. The Mitre Corporation, McLean, VA

George A. Miller, Christiane Fellbaum, Randee Tengi, Susanne Wolff, Pamela Wakefield, and Helen Langone. 1998. *WordNet: a lexical database for the English language*. Princeton University, Princeton, NJ

Ken Barker, Ylias Chali, Terry Copeck, Stan Matwin, and Stan Szpakowicz. 1998. *The Design of a Configurable Text Summarization System*. University of Ottowa, Ottowa, CA

Stan Szpakowicz, Ken Barker, Terry Copeck, J. F. Delannoy, and Stan Matwin. 1996. *Preliminary Validation of a Text Summarization Algorithm*. University of Ottowa, Ottowa, CA

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

Appendix A: Sample Summaries

Here we include four summaries, both with and without using FERNANDO for two documents, SAUDI and ENGINE, chosen to be representative of "good" documents (SAUDI) and "bad" documents (ENGINE) for summarization by DUTCHMAN.

SAUDI - No FERNANDO:

A fierce battle for this deserted coastal town ended today when forces from Saudi Arabia and the emirate of Qatar, backed by American artillery and air strikes, evicted Iraqi troops and tanks, and freed two trapped U.S. reconnaissance teams. Marine commanders explained that Saudi forces had responsibility for the defense of the border area around Khafji, a town of 45,000 people on the Persian Gulf about six miles south of the Kuwait frontier. Marines provided artillery support and air strikes from Cobra gunships, but did not participate in the on-again, off-again ground battle, an occasionally tense confrontation involving close-quarters encounters between tanks and troops in the middle of town.

SAUDI - With FERNANDO:

A fierce battle for this deserted coastal town ended today when forces from Saudi Arabia and the emirate of Qatar, backed by American artillery and air strikes, evicted Iraqi troops and tanks, and freed two trapped U.S. reconnaissance teams. Marine commanders explained that Saudi forces had responsibility for the defense of the border area around Khafji, a town of 45,000 people on the Persian Gulf about six miles south of the Kuwait frontier. But Marine Lt. Col. Garrett, supervising Marine fire teams, supporting the Saudi counter-strikes, met with the U.S. officer serving as liaison with the Saudi and Qatari forces, who checked with Admire and called a meeting to see if the Marine reconnaissance teams could be extracted using a Saudi tank attack as cover.

ENGINE - No FERNANDO:

With the exhaust valve closed and the intake valve open, the piston moves down in the cylinder as the engine crankshaft turns. The operation of the four stroke cycle style of engine depends on the timing of its valves and their condition, on the piston rings, and on the cylinder walls. This is the standard number per cylinder in almost all four stroke cycle engines, with the exception of some aircraft engines and racing car engines which have four valves per cylinder.

ENGINE - With FERNANDO:

The operation of the four stroke cycle style of engine depends on the timing of its valves and their condition, on the piston rings, and on the cylinder walls. This is the standard number per cylinder in almost all four stroke cycle engines, with the exception of some aircraft engines and racing car engines which have four valves per cylinder. This causes a partial vacuum in the crankcase to prevent oil from being forced out of the engine past the piston rings, oil seals and gaskets.

Appendix B: DUTCHMAN's Abstract

Here we have used the DUTCHMAN system to generate an alternate abstract of the DUTCHMAN paper; the original abstract, references, and appendices were excluded from the source document; FERNANDO was used. All things considered, this summary is terrible.

DUTCHMAN - With FERNANDO:

First, since it requires a large knowledge base in order to function, the results do not generalize across languages well. Many text summarization systems choose to adopt such a restricted domain, and thus are able to exploit the self-summarizing nature of such documents. A larger testing set, with a narrower range of article types, and a broader base of human readers, with statistics on how well the humans believed they summarized the articles, and comparisons of the sets of human-identified keywords, would all aid in evaluating a summarizer.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 63–68
Computer Science Department, Swarthmore College

Wordnet Wordsense Disambigioution using an Automatically Generated Ontology

Sven Olsen
Swarthmore College
solsen1@swarthmore.edu

Abstract

In this paper we present a word sense disambiguation method in which ambiguous words are first disambiguated to senses from an automatically generated ontology, and from there mapped to Wordnet senses. We use the "clustering by committee" algorithm to automatically generate sense clusters given untagged text. The content of each cluster is used to map ambiguous words from those clusters to Wordnet senses. The algorithm does not require any training data, but we suspect that performance could be improved by supplementing the text to be disambiguated with untagged text from a similar source. We compare our algorithm to a similar disambiguation scheme that does not make use of automatically generated senses, as well as too an intermediate algorithm that makes use of the automatically generated semantic categories, but does not limit itself to the actual sense clusters. While what results we were able to gather show that the direct disambiguator outperforms our other two algorithms, there are a number of reasons not to give up hope in the approach.

1 Introduction

Word sense disambiguation algorithms are valuable because there are a number of tasks, such as machine translation and information extraction, for which being able to perform effective word sense disambiguation is helpful or even necessary. In order to fully define the task of word sense disambiguation (WSD), we need to know the set of senses associated with a given word. What set of senses ought to be associated with any word almost certainly depends on the context we are working in. In the case of automatic translation from English to another language, the best sense set for each word should be influenced by the set of translations of that word into the target language. Translation between distant languages such as English and Inuit might require much finer sense disambiguation than would be needed when going between related languages such as English and German.

WSD becomes a much more tractable problem when we have some understanding of the semantics of the senses that we are disambiguating. For this reason word sense disambiguation experiments are usually do assuming the sense sets of large ontologies such as Wordnet. Using Wordnet senses gives researchers access to information regarding the semantic relationships of the senses of deferent words, and many WSD algorithms rely on knowledge of these relationships. Using Wordnet senses may also make the act of sense disambiguation more useful. For example, an information extraction algorithm may take advantage of the semantic content implied by Wordnet senses.

However, there are a number of reasons why Wordnet might not be the ideal ontology for any given task. If we try to use Wordnet in an information retrieval task we may find that important technical terms are missing (O'Sullivan, 1995). If we try to use Wordnet for machine translation tasks, we may find that the sense distinctions are too fine. In a perfect world, we would have a separate ontology specifically tailored for each task. However, compiling ontologies tends to be very difficult, and so Wordnet is still the de facto standard for most WSD experiments.

Naturally there is a demand for algorithms that can automatically infer ontologies from text, thus providing researchers with an infinite set of viable alternatives to Wordnet. While no current automatically generated ontology can compete with Wordnet's fine sense distinctions, Pantel and Lin (2002) present an algorithm capable of generative sense groups of a quality similar to those in Roget's thesaurus (2002). Unlike Wordnet, this automati-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

cally generated ontology has no hierarchical information, instead it simply provides groups of related words senses.

In this paper we present and algorithm which automatically generates an ontology given untagged text, and then disambiguates that text into the senses of the generated ontology. Thus we hope to provide researchers with a context sensitive alternative to Wordnet based disambiguation. We also outline a method for converting our senses to Wordnet senses. This allows us to disambiguate text to Wordnet senses by first disambiguating to the automatically generated senses, and then mapping the results to Wordnet. Because we expect the automatically generated sense clusters to be coarser than those of Wordnet, and because the act of generating the senses leaves our algorithm with access to extra information regarding the ambiguous senses, we expect that disambiguating to the automatically generated senses will be easy.

There are ways in which our method of disambiguating to Wordnet senses might have advantages over more direct approaches. Because the senses used by our system are inferred from the text to be disambiguated, we can expect to avoid confusion caused by senses that never appear in our text. Additionally, our system has the advantage of requiring no tagged training data. Mapping the automatically generated senses to Wordnet senses may be complicated by the fact that the generated senses are coarser than Wordnet's, however, we expect that the type mistakes realized because of this to be similar to those mistakes that a human would make when tagging text with the often frustratingly fine Wordnet senses.

2 Related Work

Lin (1994) introduced PRINCIPAR, a broad coverage English parser that works using a message passing model. Among other things, PRINCIPAR can be made to output a set of "dependency triples" given any sentence. Recent work done using MiniPar, PRINCIPAR's publicly available successor, has shown that these dependency triples prove quite useful in the context of a number of different tasks.

Lin (1997) introduces an algorithm for word sense disambiguation based on information from MiniPar's dependency triples.

Lin (1998) includes an excellent articulation of the means through which the syntactic information represented by the dependency triples can be used to infer semantic knowledge. Papers such as our own and Pantel and Lin (2002) tend to rush their descriptions of the methods first outlined in this paper, and readers trying to implement our algorithms for themselves will be well served by referring back to it.

Pantel and Lin (2002) presents an algorithm in which the information from the dependency triples is used to automatically generate sense categories. The same paper also proposes a method for evaluating the similarity between sense categories and Wordnet. Using their own similarity measure, Pantel and Lin found that the categories they created automatically were more similar to Wordnet than Roget's thesaurus.

3 Methods

3.1 Automatic sense clustering

In order to generate our ontology we have implemented the method described in Pantel (2002). The starting point of Pantel's algorithm is the publicity available parser MiniPar. For each sentence in our corpus, we use MiniPar to generate a parse tree, and then from that tree infer a set of dependency triples. Each dependency triple specifies two words and the syntactic relationship between them. For example, one of the triples generated by the sentence "Bob drinks the wine" is [drink, verb/direct object, wine]. We call the construct [drink, verb/direct object,*] a "feature". The frequency of a feature for a given word w is defined as the number of times that we see that feature with w filling in the wildcard slot. Thus, if we see "Bob drinks the wine" in our corpus, one of the frequencies that we increment is $F_{[drink,verb/directobject,*]}(wine)$. The frequency of all features that ever appear with a given word define that word's frequency vector [1].

In order to quickly compile frequency information for a large set of sentences, we use a number of sorted associative containers (STL maps)[2]. We use a map from word-feature pairs to integers to represent a sparse matrix that holds the frequencies of any word/feature pair. We also use maps from strings to integers to store the frequencies of each feature and word. We use yet more maps, these from strings to vectors of strings, to store lists of the features associated with every word, and conversely the words associated with every feature. The map based representation of our data allows us to quickly update frequency information given new sets of dependency triples; $O(log(n))$ string comparisons are required to lookup a value given the string key, and the data structures are such that it is easy to insert information corresponding to novel words and features. But once all the triples have been processed, our map based data structure becomes needlessly inefficient. Therefore we convert as much of the data as possible to an indexed representation, assigning each word and feature an integer label, and collapsing many of our maps to vectors (dropping lookup time from O(log(n)) to O(1), and doing away with the

[1] The concept of feature frequency is explained with more detail in Lin (1998), and with less detail in Pantel (2002).

[2] In order to properly analyze the space/time efficiency of our algorithm, it need to be noted that the version of STL that we use implements maps using red-black binary search trees.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

need for expensive string comparisons)[3].

The basic assumption at the heart of Pantel's algorithm is that semantic similarity will be reflected in the syntactic information inherent in the feature frequency vectors. In other words, if we see [drink, verb/direct object, wine] a lot, and [drink, verb/direct object, beer] a lot, then there is a good chance that beer and wine fit into the same semantic category. We now outline a semantic similarity measure which reflects this assumption.

For each word we compute its mutual information with every feature, and store that information in a sparse vector. The equation for mutual information given a word w and a feature f is

$$mi_{w,c} = \frac{\frac{F_c(w)}{N}}{\frac{\sum_i F_i(w)}{N} \times \frac{\sum_i F_i(w)}{N}}$$

As you can see from the equation, the values that were stored when compiling frequency information were picked to make calculating each word's mutual information vector as fast as possible.

Following the suggestion in Pantel (2002), we multiply our mutual information score by the following discounting factor:

$$\frac{F_c(w)}{F_c(w)+1} \times \frac{min(\sum_i F_i(w), \sum_i F_i(w))}{min(\sum_i F_i(w), \sum_i F_i(w))+1}$$

The theory motivating this discounting factor was not well explained in Pantel (2002), but because we admire his results we follow Pantel's lead.

We define the similarity between two words as the cosine similarity of the associated mutual information vectors.

In order to perform the main clustering algorithm, we create a matrix that caches the similarity between any two words. Taking advantage of the sparse nature of the dataset, we only calculate the similarity of words which share a common feature (preliminary tests show that this strategy allows us to compute the similarity matrix roughly an order of magnitude faster than we could using the naive approach). Had our similarity matrix calculations gone too slowly, we could have further speed up the process by applying the "salient feature" heuristic described in Pantel (2002), however never applied the algorithm to a situation in which the extra speed was necessary. Pantel refers to the processes of setting up the similarity matrix as "phase 1" of the clustering algorithm.

In "phase 2", the words' mutual information vectors are clustered using the "clustering by committee" (CBC) algorithm. The goal of CBC is to create large tight clusters with the additional goal that the centroids of each cluster not be too similar to each other.

The CBC algorithm is recursive. Given a set of elements E, each $e \in E$ contributes a potential cluster composed of between 3 and 12 of the elements most similar to e. Potential clusters are assigned a score equal to the product of their size and their average pairwise similarly. Ideally, we would like to find the highest scoring subset of the set of e's twelve most similar elements and use it as e's potential cluster. Unfortunately, performing an exhaustive search of all possible subsets is computationally expensive. Performing hierarchical average-link clustering seems like one reasonable way to attack the problem, but we suspected that a straightforward greedy search might perform better in practice. Thus our own implementation of CBC uses the greedy search[4].

Potential clusters are sorted according to their score. The sorted list is then traversed, and each potential cluster is added to the set of committees C (initially empty) on the condition that its centroid not have a cosine similarity of more than $\sigma = .35$ with any element of C. If C is empty (which happens in the case that we were unable to create any valid potential clusters in the previous step), we return C.

We then define the set of residual elements R as all e such that e does not have a similarity of at least $\theta = .075$ with any of the committee centroids. Finally the algorithm is recursively called replacing E with R, and we return the union of C and the result.

The algorithms now proceeds to phase 3. Once committees have been created, we generate our ontology by assigning each word to some number of committees; each committee is taken to represent a semantic category. The assignment algorithm works as follows: for each word w, we first select the top 200 committees with centroids most similar to w's mutual information vector. Of those, we consider the committee with centroid most similar to w to be w's first sense. Then we remove the common features of w and the centroid from w's mutual information vector. Now we look for another committee to add w to that has a centroid similar to the new w vector. If at any time the similarity between w and the most similar remaining cluster falls bellow some threshold (in our case .05), we stop assigning senses to w. This method allows us to assign words to clusters that represent very rare senses of that word. Unfortunately, the algorithm is very slow, as the similarity of each cluster to the word vector must be recalculated after every step through the loop.

[3]Our source code is available for the benefit of readers interested in finding out more about the details of the data structures used.

[4]It is unclear what method Pantel (2002) uses to create the potential clusters, our initial interpretation of the paper lead us to believe that Pantel had used the hierarchical approach, but we are no longer certain of this.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

There are a couple of things worth noting about the sense generating algorithm. The committees that a word is assigned to in phase 3 have no immediate connection to the committees that the word helped define, and every word will likely be assigned to some committees that it had played no part is creating. Note also that there is no guarantee a word will be assigned even a single sense.

Given the committees representing its senses, disambiguating a given instance of a word is simple. We calculate the implied feature vector of the instance of the word as if the word instance were a novel word appearing only once in the text. We then find the committee with a centroid most similar to that vector, and say that the sense of the word is the one associated with that committee.

3.2 Disambiguation to Wordnet Senses

Wordnet defines a number of relationships between word senses. Our algorithms only make use of the hypernym and hyponym relations. The hypernyms R of word w are those r for which it is the case that "w is a kind of r". Conversely, the hyponyms of w are the words P such that "p is a kind of w" is true. Thus, 'drink' and 'inebriant' are both hypernyms of a sense of 'wine', whereas 'sake' and 'vermouth' are hyponyms of 'wine'. (According to Wordnet wine is polysemous, 'wine' can also mean a shade of dark red).

In order to link sense clusters created by CBC to Wordnet senses we need to decide what Wordnet sense is associated with every sense of every word in the automatically generated ontology. To do this we search Wordnet for semantic relatives of each ambiguous word's senses in order to find semantically similar words that have a good chance of exhibiting the syntactic features that CBC would have picked up on in the course of its sense clustering. We then find the centroid of that group of words, and decide what Wordnet sense to associate each word's CBC sense with by comparing the centroid of the CBC sense's committee with the centroid of the group of similar words gathered from Wordnet.

As an example, lets say that we some generate the following similarity group for the first sense of wine: {sake, vermouth, champagne, burgundy}. The CBC cluster that we will associate with the first sense of wine will be one based on features that tend to arise around nouns that specify alcoholic beverages. The similarity group for the second sense of wine might look something like {yellow, rose, pink, red}, and thus its centroid vector would be filled with features that are associated with colors. Note that if the text that we are using to generate our senses does not have any instances in which 'wine' is used to describe a color, then we could expect that CBC never added wine to a committee associated with color. In this case we wouldn't map any CBC sense to the second sense of wine (and this is a good thing, as it will force all of our disambiguations of the noun wine to be correct).

Clearly, our mapping method depends on having a good way of creating similarity groups for a given sense. In the case of wine's first sense (vino), the set of hyponyms is very large, and contains nothing but kinds of wine. We would expect kinds of wine to turn up in the same syntactic positions as the word 'wine' itself, so in this case using hyponyms as a similarity set is a good idea. However, the second sense of wine (color), has no hyponyms. What we want in this case for a similarity set are the sisters and uncles of the sense, the hyponyms of the hypernyms of wine (in this case, other kinds of red, and other kinds of colors).

Using the case of wine as our only example, we might conclude that the best way to develop a similarity group for a word sense is to start by collecting its hyponyms, and if they prove too small a set, expand to sister, uncle, and cousin words. We want to avoid using words from too high up in the tree, as 'intoxicant' (one of the hypernyms of wine-*vino*) is not likely to be used in the same sorts of syntactic contexts as 'wine'.

But now consider the problem of creating a similarity group for the word 'intoxicant'. Its hyponyms include things like 'vodka', which will likely have a very different feature vector than 'intoxicant'. The words that we want to see in a similarity group of 'intoxicant' are things like 'barbiturate', 'inebriant', and 'sedative'. These are all sisters of 'intoxicant'.

Because the hyponym favoring approach runs into problems in the case of high level words such as intoxicant, we adopt a method for gathering similar words in which we favor sister words above all else[5], and expand the similarity group to include both daughters and cousin words if we can't find enough sisters to make an informative centroid. Here a similarity group is considered to be "informative" if it contains 15 words for which we have gathered frequency information.

One interesting question is whether or not to limit the allowed members of a similarity group to monosense words. In the case of wine-*color*, two of its sister words are 'burgundy' and 'claret', both of which also hyponyms of wine-*vino*. This example demonstrates a potential problem for our algorithm, if we happen to create a similarity group containing many polysemous words with a shared second sense, the similarity group might create a centroid closer to that second sense than to the one that

[5]It should be mentioned that the first words added to a similarity group are the synonyms. This is a byproduct of the fact that a word's first sister is itself. The Wordnet C library returns its search results in the form of word sets; each set contains a word sense and all of that sense's synonyms. Thus the first search result returned when we look for a word's sisters contains all of that word's synonyms. While we do not consider a word to be part of its own similarity group, we do add all of its immediate synonyms.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

we were trying to represent. We have experimented with limiting similarity groups to monosense words, but found that because most of the words in Wordnet seem to be polysemous, the monosense restriction cripples our algorithm's ability to come up with similarity groups of any significant size.

3.3 Direct and Semi-Direct Wordnet Disambiguation

We have created a direct disambiguation algorithm to compare with our algorithm for disambiguation via CBC senses. Our CBC dependent disambiguation algorithm works by creating a feature vector for the instance of the word to be disambiguated, then finding the CBC sense group closest to that vector, and finally finding the Wordnet similarity group closest to the sense group. The direct algorithm just matches the word instance vector with the closest Wordnet similarity group. Thus, comparing it with our CBC algorithm provides a measure of whether or not mapping to the automatically generated sense is helping or hurting us.

Similarly, we can modify the CBC dependent algorithm by substituting the entire set of committees generated in phase 2 for the set of CBC senses associated with the word. This algorithm allows us to avoid the expensive computation costs inherent in phase 3. Because the "semi-direct" approach has the potential to take advantage of some of the advantages of using an automatically generated ontology (because we are moving first to a coarse sense we can hope that our mistakes will be between senses with similar meanings). However, because of the large number of potential committees, it is likely that the vector that we end up matching with the Wordnet simgroups will be reasonably similar to the vector that we started with, and for this reason our results should tend to be more like those from the direct approach than those achieved using the CBC senses.

3.4 Evaluation Method

We have tested our algorithms using the SEMCOR corpus, our version of which was created by transforming the tags of the Wordnet 1.6 version of SEMCOR to Wordnet 1.7 senses. We comparing our algorithm's disambiguation of polysemous nouns and verbs with the SEMCOR's stated correct senses. All of our word/feature frequency statistics are generated from SEMCOR sentences. To evaluate our performance on a given sentence, we need to align MiniPar's parsing of the sentence with the answers from SEMCOR. This alignment process is necessarily imperfect. Sometimes MiniPar incorrectly identifies the part of speech of a word, and when that happens none of our algorithms have a chance of correctly disambiguating it. In the case that MiniPar incorrectly claims that a word which is not a verb or a noun is one, the extra word makes sentence alignment difficult. We have implemented some fairly simple algorithms that attempt to identify and throw out all such cases. MiniPar will also group words that it feels denote a single concept. For example, "Fulton Superior_Court_Judge" is stored as two words in SEMCOR, but MiniPar treats it is as a single word. In order to make sentence alignment as easy as possible, and avoid many of these kinds of cases, we ignore all proper nouns. Once sentence alignment is complete, we are left with a set of nouns and verbs, their correct Wordnet senses as provided by SEMCOR, and their MiniPar parse tree indices. Those indices are used to gather the related dependency triples, which in turn are feed to our various disambiguation algorithms.

4 Results

4.1 Wordnet Similarity Groups

All of our disambiguation algorithms rely on the Wordnet simgroups. However, a brief investigation of the similarity groups returned by our program demonstrate some worrisome trends. For example, we sometimes fail to find large similarity groups for common senses of words. The simgroup for the first sense of the verb "know" (to be cognizant of information) is:
realize,recognise, recognize.
On the other hand, obscure senses of words can turn up much larger similarity groups. The biblical sense of "know", has the similarity group:
bang, bed, breed, do_it, fuck, jazz, love, make_out, mount, nick, ride, screw, serve, service, tread.
Notice that as feared, many of the words in the similarity group are polysemous, representing relatively obscure senses of other words.

Another nasty case comes up when an obscure sense of a word has a meaning very close to that of a more common sense. For example, the 11^{th} sense of "know" (to perceive as familiar), has a similarity group very close to that of the first sense:
recognise recognize refresh review.

4.2 Disambiguation Performance

For each of the disambiguation methods that we tested: direct, CBC sense based, and semi-direct, we gathered a number of statistics. We store the number of polysemous words which were of sense 1. This allows us to compare our results to the baseline method of assigning each word to its most common sense. We also record the performance of a disambiguator that simply selects a random valid Wordnet sense to assign to each word. Finally we store performance for a third baseline disambiguator, one that uses a "qualified random" approach. The idea here is that we select randomly between the valid disambiguators for which we can find non-empty similarity

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

groups. Such a disambiguator is useful for figuring out how much our success is being influenced by the fact that some word senses are ruled out in the similarity group generation phase.

Because the algorithms used were extremely memory greedy, unix tended to kill the processes after they had run for about an hour. However, one hour was enough time for our experiments to collect a reasonable amount of data, though the trials varied slightly in length depending on what other processes were competing for memory space.

Properly summarizing our results is made more complicated by the problems inherent in word alignment. For example, during our evaluation of the direct disambiguator we successfully aligned 54941 nouns and verbs. 2715 words were discarded because SEMCOR and MiniPar disagreed about whether the word was a noun or a verb, and 8486 more of them were discarded because they were monosense. This information, along with performance statistics for the direct disambiguator and the 3 baseline disambiguators is given in tabular form below (percentages for monosense words and POS error are calculated relative to the total number of aligned words, while percentages for disambiguator performance are calculated relative to the number of attempted words):

	Number of Words	Percent
Monosense	8486	15.5
POS error	2715	4.9
Attempted	43740	79.6
Direct Disambiguator	14591	33.3
Random Choice	10082	23.0
Qualified Random	10244	23.4
First Sense	28392	64.9

Here are results for the semi-direct disambiguator.

	Number of Words	Percent
Monosense	9210	15.3
POS error	3039	5.0
Attempted	47758	76.6
Semi-Direct Dis.	13434	28.1
Random Choice	10941	22.9
Qualified Random	11118	23.3
First Sense	31029	64.9

The CBC based disambiguator often found itself trying to disambiguate words that had no associated CBC senses. Thus we recorded compiled two scores for this disambiguator, one in which we only recorded success when a CBC senses existed and we used it to successfully disambiguate the word, and "augmented" score in which we had the disambiguator return sense 1 in all cases where no sense cluster was associated with the word. We also have data for the precision and recall values of the CBC disambiguator data, though they don't fit nicely into our chart. Recall was 27.2%, and precision was 35%.

	Number of Words	Percent
Monosense	3570	15.4
POS error	1015	4.4
Attempted	18639	80.3
CBC Dis. (augmented)	10149	54.5
CBC Dis. (pure)	1778	9.5
Random Choice	4447	23.9
Qualified Random	4515	24.2
First Sense	11973	64.2

5 Conclusions

5.1 Wordnet Similarity Groups

All of our algorithms depend heavily on the similarity groups for the sense of each word. Given the problems we saw in simgroup generation, it is surprising that any of our algorithms performed better than chance. In our future work section we speculate on some ways that the similarity groups could be improved, and we imagine that all our algorithms would perform significantly better given better similarity groups.

5.2 CBC

The constant terms that we used in our implementation of CBC were taken from one of Pantel's later implementations of the algorithm. There is always a chance that the algorithm might have performed better given different parameters, but in this case it seems more likely that the problem lies in the size of our corpora. Pantel (2002) uses a 144 million word corpora to generate the frequency information provided to CBC, the SEMCOR data that we used contains slightly under a million words. It is also worth noticing that the corpora used in Pantel (2002) was a composition newspaper texts, while the Brown corpus data that makes up SEMCOR comes from a wide range of sources, including press releases and a number of different fictional genres. The heterogenous character of SEMCOR probably increased the number of different word senses used in the text, therefore making the sense clustering task more difficult.

5.3 Comparison of the Algorithms

The most comforting number in the performance statistics is the very low percent of part of speech errors. This indicates that MiniPar is doing its job reasonably well, providing a solid foundation for our algorithms to work off of. The best performance that we ever see comes from the "always pick the most common sense" baseline. This is disheartening, but given the poor quality of the similarity groups and the problems we encountered applying CBC to a dataset as small as SEMCOR, it is impressive that any of our algorithms we do better than random

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

chance. The fact that the qualified random disambiguator performs about as well as the random disambiguator is also heartening, as it implies that the gaps in our similarity sets are not making the disambiguation problem significantly harder or easier. Thus, what success the algorithms do achieve beyond the random baseline is solely the function of their ability to use the syntactic information inherent in the dependency triples to infer semantic relationships between words.

The direct and semi-direct algorithms both solidly outperform random choice, and this gives us cause to hope that if the issues with similarity group creation could be worked out, we would be left with a complete system capable of outperforming the "most common sense" baseline.

The results for the CBC based disambiguator look rather miserable at first glance, as the pure version performs worse than random chance. However, it is worse noticing that most of the errors are due to the failure of our application of CBC to create sense cluster, and that problem is a result of the small dataset size. So we can hold out hope that given a large corpus to supply supplementary frequency information, the CBC algorithm might achieve much higher performance. It is worth noticing that in those cases where it has sense clusters available to it, the CBC based algorithm has a precision higher than either of the previous two algorithms. We had hoped that the low precision CBC algorithm could be combined with the highly successful "most common" baseline. While this project didn't pan out (the "augmented" version of CBC is less effective than the "most common" baseline), we can always hope that given larger corpora and better similarity groups, we could have achieved better results.

The fact that the direct disambiguator outperforms that semi-direct disambiguator does not necessarily mean that the semi-direct disambiguator is in all ways worse than the direct disambiguator. Remember that one of the advantages that we hoped to see in the semi-direct disambiguator was errors which had a higher tendency to be mistakes confusing semantically similar senses of a word. However, we had no way of adjusting our results to take into account semantic similarity. While unencouraging, the performance scores alone are insufficient to disprove our hypotheses.

6 Future Work

There are a couple of ways in which the generation of simgroups for Wordnet senses could be improved. At the moment, we have only experimented with methods for generating senses which have a fixed "profile". That is to say, all word senses prefer to have their similarity groups filled with some predefined set of relatives. As we have implemented our algorithm, sisters are preferred over everything else, first order children over cousins, and the most distant allowed relatives are third order cousins. One could imagine changing the set of preferred relations in hopes of getting better results. However, it seems to us that the *right* thing to do would be to build an adaptive algorithm that first inferred a word's position in the synsnet, and then used that information to define the appropriate profile. Designing such an algorithm would be a reasonably large project, we would attack the issue by coming up with a loose parametrization for a family of such adaptive algorithms, and searching that parameter space for the values that maximized the performance of our direct disambiguator.

One of the problems that we observed in our similarity groups was the tendency for rare senses of a word to have simgroups very similar to those of much more common senses. Wordnet contains sense frequency information for each of a word's senses, and we would imagine that our disambiguation methods could be improved by taking advantage of that information when mapping a word instance vector to a Wordnet simgroup; the algorithm could be designed to only return a rare sense in the case that the match was very good, other wise a more common sense of the word would be preferred.

In the course of implementing the CBC algorithm, we saw a couple of ways in which it might be interesting to modify the algorithm. For example, in phase 3, CBC completely removes feature elements included in the centroid of a matched committee. However, it might be more reasonable to subtract the centroid from the word vector, then set all the negative terms to 0 (we thought this seemed like a better way of defining the "residue" of a vector). We also suspected that it might be interesting to enforce the "distance from all other committees" restriction in phase 2 of the algorithm relative to all previously generated committees, instead of just those committees generated in that iteratation of the algorithm. Both of these modifications to CBC would be easy to implement, and we would like to see how these changes to the algorithm would effect both the generated senses clusters and the performance of our disambiguation algorithms.

If the problems hampering our system could be overcome, it would be interesting to compare our results to those achieved by the disambiguator presented in Lin (1997). It represents another "direct approach" that works on principles rather different that those that we used, though like our own algorithms functions based only the dependency triple information.

It is interesting to note that unlike Wordnet, the ontology generated in Pantel (2002) had high coverage of proper nouns, which could make it more suitable for MT tasks. Al-Onaizan (2000) describes a case in which being able to guess whether an unknown proper noun is more likely naming a town or a militia group can improve MT

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

performance. We did not test the performance of our disambiguators on proper nouns, though the only thing that prevented us from doing so was a set of relatively minor technical concerns involving word alignment. If those concerns were overcome, it would be very interesting to see how our algorithms performed in the limited case of proper noun disambiguation.

If we could prove that the CBC based disambiguation system was making different sorts of mistakes than the direct disambiguation system, it would almost certainly be worth trying to create a hybrid model in hopes of combining the advantages of both approaches. Such a system could be implemented by a voting algorithm as in Florian and Wicentowski (2002). It also worth considering ways in which CBC could be modified to produce clusters that are more appropriate for mapping to Wordnet senses. One obvious modification on the current system would be to repeatedly run CBC on SEMCOR in order to find the similarity thresholds for sense clusters that imply sense distinctions most like Wordnet's. If we found that the CBC algorithm was lumping two common Wordnet senses together, we would try increasing the degree to which CBC demanded tight clusters.

Another idea would be to generate our ontology using a clustering algorithm in which sense clusters are initially seeded in a way that reflects Wordnet senses. The simplest way to do this would be to run a k-means clustering algorithm with initial clusters created from Wordnet simgroups. One might try to incorporate some of the ideas of CBC into such an algorithm by forcing clusters to have a certain degree of difference from each other.

What we have done in our experiments is to measure the extent to which disambiguations using CBC senses clusters can be effectively mapped to disambiguations for Wordnet senses. However, it might be interesting to design an experiment that tries to go the other way: we could run an off the shelf Wordnet disambiguation algorithm and then map the results to sense tags in the automatically generated ontology. If Wordnet-to-CBC worked well, but CBC-to-Wordnet worked less well, then we would be able to speculate that CBC was creating senses using a notion of semantics similar to Wordnet's, but with uniformly coarser sense distinctions. However, if Wordnet-to-CBC worked less well than CBC-to-Wordnet we might start to wonder if Wordnet was missing some interesting relationships between words that CBC was somehow picking up on (given the results in Pantel (2002) it seems likely that this wold be the case for proper nouns).

One of our hypotheses was that the sorts of mistakes that might be made by our system would be the result of confusion between two similar senses of a word. We further hypothesized that a human attempting to disambiguate words to Wordnet senses would be likely to make mistakes on the same cases that give our system the most trouble. It would be very interesting if these hypotheses were true, however we lack the funding to properly test them. If we had a couple of graduate students at our disposal, we could set them on the task of hand tagging chunks of SEMCOR. Then we could directly compare the humans' errors with our system's, as well as with other more direct WSD systems. Instead of merely paying attention to overall correctness, we would attempt to determine, as in Pedersen (2002), which cases were found most difficult by which systems, and whether or not our system made mistakes that were more "human-like" than those of the other statistical systems. If gradstudents proved to be unavailable, a large amount of word aligned text from different languages could be used as in Chugur and Gonzalo (2002) to develop a notion of sense similarity. Our hypothesis would be supported if most of our system's errors came from labelings with high similarity to the proper label, while a more conventional system exhibited errors with varied similarity.

References

Al-Onaizan, Y., Germann, U., Hermjakob, U., Knight, K., Koehn, P., Marcu, D. and Yamada, K. 2000. Translating with Scarce Resources. *The 17th National Conference of the American Association for Artificial Intelligence (AAAI-2000),* Austin, Texas.

Irina Chugur and Julio Gonzalo. 2002. A Study of Polysemy and Sense Proximity in the Senseval-2 Test Suite. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 32-39.

R. Florian and R. Wicentowski. 2002. Unsupervised Italian Word Sense Disambiguation using Wordnets and Unlabeled Corpora. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 67-73.

Denkang Lin. 1998. Automatic Retrieval and Clustering of Similar Words. *COLING-ACL98*, Montreal, Canada. August, 1998.

Denkang Lin. 1997. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. In *Proceedings of ACL-97*. Madrid, Spain. July, 1997.

Denkang Lin. 1994. PRINCIPAR—An Efficient, broad-coverage, principle-based parser. In *Proceedings of COLING-94*. pp. 42-488. Kyoto, Japan.

D. O'Sullivan, A. McElligott, R. Sutcliffe. 1995 Augmenting the Princeton WordNet with a Domain Specific Ontology. In *Proc. Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint*

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada.

Patrick Pantel and Dekang Lin. 2002. Discovering Word Senses from Text. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2002*. pp. 613-619. Edmonton, Canada.

Ted Pedersen. 2002. Assessing System Agreement and Instance Difficulty in the Lexical Sample Tasks of Senseval-2 Appears in the Proceedings of the Workshop on Word Sense Disambiguation: Recent Successes and Future Directions. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 40-46.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

A Connectionist Approach to Word Sense Disambiguation

Andy Zuppann
Swarthmore College
CS97: Senior Conference on Natural Language Processing
zuppann@cs.swarthmore.edu

Abstract

Effective word sense disambiguation can play a crucial role in several important computational linguistic tasks. Problems such as information retrieval and machine translation rely upon accurate tagging of word senses. This paper will present an English word sense classifier based upon connectionist models of learning and behavior. Results perform comparably with state of the art statistical approaches in finely grained word sense disambiguation.

1 The Problem of Word Senses

One interesting feature of language, at least from a computational linguistic standpoint, is its inherent ambiguity. Native speakers of a language have very little problem adjusting to potentially ambiguous statements, but both non-native speakers and computers face the difficulty of extracting a specific semantic meaning from statements that could have several.

An archetypical example of this lexical ambiguity is found in the word 'plant.' Given a sentence: *The plant lived in the chemical plant*, a computer attempting, say machine translation, should be aware that each usage of plant in the sentence represents a different sense of the word - in this case, the difference between a living plant and an industrial plant. It is important to correctly identify these difference because the ambiguity is unlikely to be exactly duplicated in the target language. For instance, the French word for the living plant is *plante*, while the word for the factory plant is *usine*. Clearly, a correct translator needs to be able to resolve any sense ambiguity. This paper will describe one such approach for untangling this problem based around neural networks and connectionist models.

2 Previous Work

Standard approachs to this problem have been developed using statistical methods. Various approaches include utilizing assumptions about one sense per discourse and one sense per collocation (Yarowsky, 1993), (Yarowsky, 1995). More recent work challenges and develops these assumptions into complicated statistical models based on topicality and locality of context surrounding a target word to be disambiguated. These models all rely on explicit calculations of the relevance of given context.

One major exercise in disambiguating word senses has been the SENSEVAL project. By preparing corpora in English and several other languages, the program's designers hope to create a forum for comparing the performance of several approaches to the same problem. By specifying exactly the training and testing data for the classifier systems to use, discrepancies between data and results across experiments should be ameliorated and there should be a fair comparison of all the system's disambiguating capabilities. The results of this approach have been promising, and it appears that the state of the art for word sense disambiguation is 75-80% success both in precision and in recall (Kilgarriff, 1998). Furthermore, by making the training and testing corpora used in the exercise widely available, SENSEVAL allows researchers to test and compare new methods against a solid baseline of other systems' performances.

The hypothesis of my work is that, instead of relying on human generated statistical models, a connectionist, developmental approach can yield as good, if not better, results. The foundations of this approach are strongly motivated by a desire to base learning and development in machines on our understanding of our own developmental process and root the learning in biological plausibility. Additionally, studies suggest that this approach can be as successful as other, more traditional approaches to problem solving such as Markov chains and decision trees (Quinlan, 1994).

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

Although most of the previous work has been focused on resolving sense ambiguity using statistical methods, there still exists substantial evidence that a connectionist approach can lead to comparable results within this particular domain. For instance, Mooney (1996) compares several available word sense classifiers, and out of 7 possible classifiers, a neural net approach tied for best. In this paper, Mooney used a simple perceptron network to disambiguate instances of the word 'line.' The network performs comparably with a Naive Bayesian approach to disambiguation and signficantly better than five other methods, achieving a 70% precision rate. In addition, the neural network approach both trained and tested faster than the Naive Bayesian system.

A separate study also reported a neural net having a high success rate in identifying the meanings of the words 'line,' 'serve,' and 'hard' (Towell and Voorhees, 1998). The study created topical and locational information networks and combined their output to create effective sense classifiers. The topical approach used general context information surrounding a target word. Each word surrounding the ambiguous word in the testing set is given an input into the node, but there is no encoding of any words relation to the target, just that it appears in a similar context.

The locational encoding used by Towell *et al* is a more intricate approach which, when encoding words, affixes locational information. Using their example, the sentence "John serves loyally" becomes the set [-3zzz -2zzz -1John 0serves 1loyally 2. 3zzz]. This affixation massively expands the vocabulary of context words around a target word to contain locational information for each word. Every word within this expanded vocabulary is given its own input node to the network. The locational approach permits a network to uncover for itself not only what context is important, but whether relative location matters as well for disambiguating words. This approach worked extremely well for its three target words, averaging an 86% success rate. This is not altogether surprising, given the rather coarse senses used in their experiment.

My research reported here, to a large degree, is an attempt to reproduce Towell *et al*'s topical neural network model and apply it to a different set of training data. In doing so, I plan to provide two important contributions. One, I will put a neural network model for word sense disambiguation in the context of a previously implemented general word sense exercise comparing different attempts at disambiguation. This will permit accurate comparisons of a neural network to other approachs within a broad framework. Secondly, I hope to test the general connectionist framework for sense tagging in a relatively fine-grained sense environment.

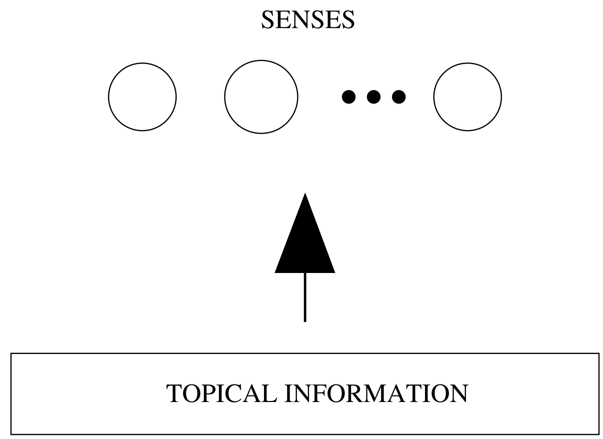


Figure 1: Network Model

3 The Classifier

This section will describe the specifics of my approach. First, the architecture of the network model will be described. Second, the data for training and testing my classifier will be covered. Finally, I will describe the learning method for the network.

3.1 Model

The classifier presented is a very simple neural network comprised of only perceptrons linking topical input to sense output. This idea is based on Towell and Voorhees, who describe a similar system performing - somewhat surprisingly - best without any hidden nodes in the network (Towell and Voorhees, 1998). Indeed, other research into this subject reveals that the poor performance of networks with hidden layers is pervasive (Mooney, 1996)

Given that I will be testing neural networks on their performance on several different potentially ambiguous words, a separate network is required for each. The reason for this is clear when one considers the nature of a single network's inputs for this task. Each network must be able to disambiguate a word's meaning based around that word's particular context and choose out of the word's available senses. This requires the network to have unique inputs and outputs for any target word. To disambiguate a new word, a new network with its own unique parameters must be created and trained.

The general architecture of this model is graphically depicted in Figure 1. A given network will consist of a fixed number of input nodes. The number of input nodes will correspond to the size of the context vocabulary found in the corpus. Any word that appears in a sentence along with the target ambiguous word will have an associated node.

When confronted with an ambiguous word, the net-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

work will collect all the words in the surrounding sentence and create an associated vector. This vector will have the length of the vocabulary, and the topical words will have their associated nodes set to 1.0. Other nodes, i.e., words that do not occur in topical context of the current target, will have their activation set to 0.0.

The output of the network will simply be one node per available sense of the word. The node with the highest activation after the network has analyzed the topical input should correspond to a given sense. This sense, then, is the network's classification of the presented instance of the target word.

One important feature of the network is that its structure almost necessitates that recall on tests will be 100%. Although a perfect word sense disambiguator would certainly have recall that high, current efforts have a much lower recall(Kilgarriff and Rosenzweig, 2000). In my network, the precision-recall trade-off can be approximated by setting a threshold of certainty on the output nodes. In other words, during testing, the network only reports results if the highest activated node is greater than all other nodes by a certain margin. Clearly, if two output nodes have similar activation then the system is having a difficult time choosing between the two senses and precision could be improved by not having to tag that instance.

3.2 Data

The data used for training and testing my model comes directly from the SENSEVAL exercise - now referred to as SENSEVAL-1 (Kilgarriff, 1998), (Kilgarriff and Rosenzweig, 2000). The exercise was intended to compare the effectiveness of several different word sense disambiguators on similar data. For the first SENSEVAL, words were seperated lexically before being disambiguated, an approach that fits nicely with my necessity of having one network model per word. The dictionary and corpus for the exercise come from a project called HECTOR, which involved the creation of a comprehensive hand-tagged sense corpus concurrently with the development of a robust dictionary of word senses.

Although SENSEVAL included several different words to disambiguate, I focus on only five of them. Their selection was based primarily on the relative abundance of both training and testing data. My model attempts to disambiguate *accident*, *band*, *brilliant*, *sanction*, and *slight*. Although the HECTOR sense tags are too fine to recreate here in great detail, Table 1 presents a few examples of ambiguities in the target words. A more complete analysis would undoubtedly test on the entire set of SENSEVAL words. It is unfortunate that, given the large training time for a network, I was unable to test my system on the entirety of the SENSEVAL data.

For each word, the SENSEVAL exercise provided training data, a dictionary, testing data, and a gold standard for answers. My system uses all of these directly from the experiment. The resulting network inputs for the training data corresponds to a varied vocabulary range, from a little over 1000 for *sanction* to over 8000 for *band*.

word	*example meanings*
accident	by chance
	collision
band	musical group
	ring
brilliant	showy
	vivid
sanction	allow
	economic penalty
slight	tiny
	least (superlative)

Table 1: Ambiguities in Target Words

word	*Vocabulary Size*	*# of senses*
accident	6129	11
band	8111	22
brilliant	3783	11
sanction	1125	5
slight	3344	8

Table 2: Data Attributes

One final and important note about the SENSEVAL taggings is that they are extremely fine. In particular, *band* had over 20 different possible senses defined, and the other words, although not as extreme, also had numerous possible senses. This clearly makes the tagging a more substantial challenge than in other connectionist approachs (Towell and Voorhees, 1998) that use a very limited number of possible sense tags. Table 2 reports the number of senses and vocabulary sizes for the words tested.

3.3 Learning Method

The learning method for a given network is the standard error backpropagation method used in teaching neural networks. After feeding the network a training input, the resulting output is compared to the expected output, error is computed, and weights and biases are adjusted accordingly.

One useful indicator for ending learning is the convergence of error of the network to a steady state. Using this as a basis, my network would train until error reached an asymptotic level. In general, this means the networks would learn for 15 to 20 epochs, seemingly quite fast. Given the size of the training set and the speed in training perceptrons, this is not altogether surprising.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

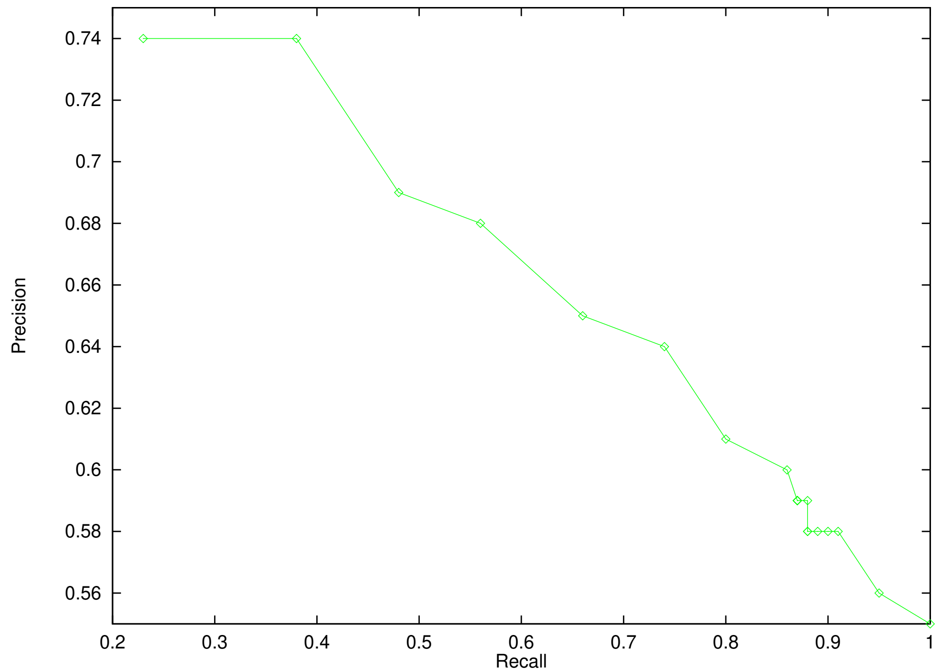


Figure 2: Precision-Recall Performance for *sanction*

word	*precision*	*F-Measure*
accident	70.4%	82.6%
band	64.4%	78.4%
brilliant	32.9%	49.5%
sanction	55.6%	71.5%
slight	69.7%	82.1%
average	58.6%	72.8%

Table 3: Performance with 100% Recall

4 Results

After training five different networks to disambiguate the target words, I was able to test the network's performance on the SENSEVAL test sets. The test sets were generally substantially smaller than the training sets. Initial testing results are reported in Table 3.

Again, it should be noted that the architecture of the neural net is set up to give 100% recall. For any given test sentence, a particular output node will be activated to a greater extent than any other output node. This 100% recall can be a problem, as most effective word sense disambiguators have a much lower recall. Nonetheless, performance is still quite good on all words except for brilliant. Also, the system's perfect recall leads to rather high F-Measures.

Given the potential problem that 100% recall is causing, a next step in testing was to try to lower the recall rate and raise precision. To do this, it was necessary to give the system the capability to tag a particular word as 'unknown.' I implemented this functionality by creating a threshold value to determine the certainty of the network's output. If the difference between the two highest activated output nodes was not greater than the threshold, then the system has an unacceptable degree of uncertainty about its output and chooses not to tag the word.

Adding this threshold technique for determining sense outputs, precision should be increased. To test the effectiveness of the threshold, I sampled a range of thresholds to plot the relation between recall and precision. We would expect to see an inverse relation. As the network stops tagging words that it is relatively unsure of, its recall falls but, since certainty of its taggings is higher, precision has likely increased. The test of this hypothesis is reported in Figure 2. Using the network trained to disambiguate the word *sanction*, increasing the certainty threshold causes a fall in recall and a rise in precision, the expected result.

Although this threshold permits the network to evaluate the certainty of various output taggings, the approach still has a few weaknesses. For one, the threshold must be assigned by an outside observer, and there appears to be no general rule for assigning the threshold. Instead, I sampled a variety of possible thresholds for the words and thereby selected thresholds that yielded seemingly reasonable results. It would be much more desirable to have the network generate its own thresholds and learn from them.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

word	*precision*	*recall*	*F-Measure*
accident	74.0%	90.6%	81.5%
band	64.5%	99.0%	78.1%
sanction	63.8%	74.1%	68.6%
slight	78.3%	69.7%	73.7%
average	70.2%	83.4%	75.5%

Table 4: Performance with Lowered Recall

This well-documented relation between precision and recall suggests that better results can be achieved by lowering the sensitivity of the network's output. Using arbitrary thresholds, the networks' precision improved substantially, as shown in Table 4. It should be noted that brilliant has not been thoroughly tested with a threshold, due mostly to time constraints involved with finding thresholds for any particular net.

Unfortunately, the rise in precision in this approach was met with a more than proportional fall in recall. This fact can be seen by observing the change in F-Measures between the two tests. The average is slightly higher due to the absence of brilliant's results, but every individual F-Measure is worse than the tests with 100% recall. This drop in total system performance is certainly unexpected, and actually supports keeping the initial system intact and not using any threshold for determining certainty. One potential reason for this anomaly is the aforementioned arbitrary nature of the thresholds. A network that had incorporated certainty measures throughout learning would perhaps perform in a more expected fashion.

5 Discussion

Using the results from the SENSEVAL study, the connectionist approach described here stands up quite well. The state of the art for the statistical approaches used in the exercise is around 75-80% for both precision and recall(Kilgarriff and Rosenzweig, 2000). Although my system performs slightly worse on the five words I attempted, results are nonetheless quite comparable. A more apt comparison would clearly come from looking at the differing F-Measures for all the systems. Unfortunately, SENSEVAL results do not report this statistic for the evaluated systems. A rough calculation can be made, using the reported results of the best performers. If the best systems were between 75-80% in both precision and recall, then the system's F-Measures must be bounded between 75-80% as well. Using that as a comparison, my system performs admirably, with all tested words except brilliant having comparable results in the 100% recall test.

Although the network performed comparably on four of the five tested words, the results presented here are not a complete comparison to the SENSEVAL exercise. The full exercise had 35 words with 41 associated disambiguation tasks. These tasks included much more challenging tasks such as words with differing parts of speech and words with limited or no training data. The use of data with an ample training set might have unfairly influenced my system's performance. Nonetheless, my results are promising enough in general to prove that the connectionist approach can potentially compete with excellent statistical classifiers. Further work is certainly warranted to more generally test this approach's viability.

With regard to the specifics of the network performance, one important fact is the tendency for the networks to only focus on the most frequent senses. Even when presented with several senses, the network would usually ignore senses with very low frequency. In general, the network would only select the two or three most common senses as its chosen tags. One possible explanation for this behavior is the lack of hidden nodes. Hidden nodes would allow the network to develop a more nuanced approach to the context relevant for categorizing senses, and, as such, would be more likely to uncover the occurrences of less frequent words.

6 Future Work

The lack of hidden nodes provides an interesting arena for future research. The slow speed of network training prohibited an in-depth look at this current time, but I feel that future work could look into several interesting areas. As has been previously noted, fine taggings are likely to be better handled with hidden layers. Additionaly, hidden layers should be able to extract more intricate levels of meaning such as distinct phrases. Towell *et al* discuss this possibility, describing how diagnostic phrases such as 'stand in line' cannot be fully represented in a simple perceptron net based on topicality(Towell and Voorhees, 1998). A hidden layer would allow this sort of phrase to be characterized directly in one hidden node, albeit with that node probably handling several possible phrases from different contexts.

Another problem with the approach presented here is its reliance on having a unique network for every target word. A more robust possibility would be to create an enormous neural network that would incorporate the entire vocabulary from all the training sets as input nodes and additional input nodes specifying what word is currently ambiguous. The outputs for this network would be all the senses of all the words. A network architecture of this type is clearly enormous and is probably prohibitively costly to train or test, but nonetheless could potentially provide a much more general solution to the problem of word sense disambiguation.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

7 Conclusion

This paper has presented a connectionist method of implementing word sense disambiguation. As this method is currently underexplored within the domain of natural language processing, this paper represents an important step in showing the feasibility of using neural networks for computational linguistic tasks. Further, the tests presented lend themselves to easy comparison to other systems' attempts at solving the same problem, as it utilizes the same testing and training corpora that were used in the SENSEVAL exercise.

My network has clearly demonstrated its ability to reasonably disambiguate a word given a sentence of context. Although the full range of SENSEVAL words was not fully tested, the results perform comparably with the systems that participated in the exercise, with the F-Measure of precision and recall averaging around 75%. Clearly, a fuller testing of all the words should provide a more complete analysis of the viability of a connectionist model.

Steps forward clearly include a deeper look into the potential advantages of using hidden nodes to allow increased generalization and more subtle analysis of context. Also, the automatic generation of certainty thresholds during training should permit the network to efficiently trade off between precision and recall. Nonetheless, this paper has successfully demonstrated that neural networks provide a reasonable framework for disambiguating word senses.

References

Kilgarriff Adam. 1998. SENSEVAL: An Exercise in Evaluating Word Sense Disambiguation Programs. Information Technology Research Institute Technical Report Series University of Brighton, U.K.

Kilgarriff Adam and Joseph Rosenzweig. English SENSEVAL: Report and Results. In Proceedings of the 2nd International Conference on Language Resources and Evaluation. 2000.

Mihalcea Rada and Dan I. Moldovan. 1999. An Automatic Method for Generating Sense Tagged Corpora. *AAAI IAAI*. 461-466.

Mooney Raymond J. 1996. Comparative Experiments on Disambiguating Word Senses: An Illustration of the Role of Bias in Machine Learning. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Ed. Eric Brill and Kenneth Church. Association for Computational Linguistics. 82-91.

Ng Hwee Tou 1997. Getting Serious About Word Sense Disambiguation. *Tagging Text with Lexical Semantics: Why, What, and How? ANLP-97 Workshop*. Washington D.C.

Quinlan J. R. 1994. Comparing Connectionist and Symbolic Learning Methods. *Computational Learning Theory and Natural Learning Systems: Volume 1: Constraints and Prospects*. MIT Press. 445-456.

Towell Geoffrey and Ellen M. Voorhees 1998. Disambiguating Highly Ambiguous Words. *Computational Linguistics*. V. 24, N. 1 125-145.

Yarowksy David. 1993. One Sense Per Collocation. In *Proceedings, ARPA Human Language Technology Workshop*. Princeton, NJ. 266-71.

Yarowsky David. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. *Proceedings of the 33rd Annual Meeting of the Assocation for Computational Linguistics*. 189-96.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 78–83
Computer Science Department, Swarthmore College

Automatic Rule Generation and Generalization for an Information Extraction System Using MAXIM

J. McConnell
Swarthmore College
500 College Ave.
Swarthmore, PA 19081
mmcconn1@swarthmore.edu

Anteneh Tesfaye
Swarthmore College
500 College Age.
Swarthmore, PA 19081
atesfay1@swarthmore.edu

Abstract

Many recent information extraction (IE) systems have ignored the tedious and time-consuming nature of the preparation involved in using them. The abundance of graduate students has eased the pain of providing annotated corpora, pre-filled answer templates, and manual examination of automatically-generated rules and final answers. In this paper, we present a new system comprised of previously published solutions to different aspects of IE in an effort to automate as much of the task as possible while achieving competitive results.

1 Introduction

1.1 Background Information

The recent availability of large quantities of text in electronic format on the World Wide Web, has greatly increased the importance of intelligently extracting desired information from such text. The task of information extraction is most easily described as that of filling a database with information found in structured, semi-structured, or free text. Structured text can be thought of as text in a pre-defined format, like in a printed spreadsheet. Semi-structured text follows general guidelines, but is not as predictable as structured text. It is often ungrammatical with a lot of fragmented sentences. An example of such text might be telegraphic, military communications, or birthday invitations. Free text is just normal prose, as found in a news article or a work of fiction, for example.

DARPA recognized the significance of this growing field in the late 1980's when they funded the first Message Understanding Conference (MUC-1). The MUC has been held semi-annually since, and has highlighted developments in IE research. Since the early knowledge-engineered (KE) systems developed for MUC-1, the field has seen a trend towards automation to circumvent the bottleneck associated with KE.

This trend is fueled by the difficulties inherent in all KE tasks. Extensive, yet required, human involvement makes them costly to develop, test, and apply to different problem domains both in time and money. These systems often require annotated corpora, pre-filled answer templates, human designed rules, or, if the systems automate the rule-making process, the manual examination of such rules to weed out the poor ones. These requirements are simply unacceptable for many potential applications, and we believe they are unnecessary.

1.2 Relevant Work

Since MUC-1, researchers have been searching for an effective, autonomous IE system. Showcased at MUC-4, Riloff's AutoSlog (1993) program automatically generated a dictionary of concepts which were later used to extract information from text similar in category to those with which it was trained. This system proved capable of achieving 98% of the performance of a hand-crafted dictionary developed by graduate students. The students' dictionary took 1500 person-hours to build, while the AutoSlog dictionary only required 5 person-hours in order to hand-filter automatically-generated rules.

Despite the obvious benefits of AutoSlog, it was still not practical for real-world use. As input, AutoSlog required either a set of answer keys or a semantically-annotated corpus. This was used to provide AutoSlog with examples of information to be extracted. Consequently, AutoSlog does not port well to different domains since it takes many person-hours to either fill in a set of answer keys or annotate a corpus.

To address these concerns, Riloff developed AutoSlog-TS (1996). This improvement on AutoSlog automatically generated extraction rules given completely unannotated text. As input, it requires two texts, one relevant to the problem domain, and one completely irrelevant. It works by generating an extraction pattern for every noun phrase

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

in a given text. It then compares the extraction patterns found in the relevant text to those found in the irrelevant text. Those patterns that show up frequently in the former but not at all in the latter are presumed to be pertinent to the problem domain. This system is hindered however by the need for manual examination of the resulting rules, normally about 2,000, in order to discard poor choices.

Another system developed to automate the rule generation process is Rapier (Califf and Mooney, 1997). Rapier takes as input sets of training text paired with a corresponding answer key. It then uses this information, combined with the output of a POS tagger and semantic class tagger, each given the training text, to create specific extraction rules that extract the correct answers. Up to this point, Rapier is quite similar in execution to AutoSlog; however, the system then goes on to generalize these specific rules in order to make the resulting dictionary more robust. That robustness is the strength and the point of the Rapier system.

2 The MAXIM System

2.1 Basis for MAXIM

The Maximally Automated eXtractor of InforMation (MAXIM) system was developed out of the need for an IE method that required next to no human intervention or preparation but still received satisfactory results. AutoSlog-TS's necessity for manual examination of rules as well as its lack of robustness given the specificity of its resultant rules leaves something to be desired. Rapier's need for answer keys means many person-hours are required before training on any particular problem domain. The respective strengths and weaknesses of these systems complement each other well. As a result, we propose a joint system built with the better halves of both. This system consists of an implementation of the rule generation phase of AutoSlog-TS and a rule generalization process inspired by Rapier's rule representation and learning algorithm.

MAXIM takes as input pre-classified text from both inside and outside the problem domain.[1] It will feed this text to the rule-generation phase of AutoSlog-TS which has been modified to represent rules in a format similar to that of Rapier's (see Subsection 2.2). To minimize the time spent on the manual examination of the many resultant rules, we used a clustering algorithm in order to group similar rules. We were then required to examine only a fraction of the rules outputted by AutoSlog-TS.

[1] By "classified text", we mean a text that is marked relevant or irrelevant. We assume that finding qualified texts should not be difficult or time-consuming given that relevant text is required for training any system and irrelevant text is readily available online.

Pre-filler:	Filler:	Post-filler:
1) tag: {nn,nnp}	1) word: undisclosed	1) sem: price
2) list: length 2	tag: jj	

Figure 1: Sample Rule Learned by Rapier

2.2 Rule Representation

2.2.1 Rapier

Rapier's rules consist of three parts, a pre-filler pattern, filler pattern, and post-filler pattern. Each pattern consists of any number of constraints. Constraints can be defined to match an exact word, any word with a given POS tag, or any word that matches a given semantic class as defined by WordNet (Miller et al., 1993). Within constraints, expressions can be disjunctive if written as $\{constraint_1,\ constraint_2,\ \ldots\}$ where the constraints musts all be either exact words, POS tags, or semantic class tags. For example, to specify that a pattern should match with either an adjective or an adverb, the constraint would be {JJ, ADV}.

The appeal of Rapier's rule representation is its ability to express a general idea as opposed to relying on specific word choice. For example, a rule generated by Rapier for extracting the transaction amount from a newswire regarding a corporate acquisition is shown in Figure 1. The value being extracted is in the filler slot and its pre-filler pattern is a list of at most two words whose POS tag is either noun or proper noun. The post-filler pattern requires a WordNet semantic category "price".

2.2.2 MAXIM

Rapier's slot constraints form the underlying idea of the rule representation that we have adopted for MAXIM. Due to the inconsistencies between the methods used by AutoSlog-TS and Rapier to generate extraction patterns, we had to use the pre- and post-filler slots as containers for extracted values, which contrasts with Rapier's use of the filler slot for this purpose. This simple but crucial alteration in slot design meant that we could not use Rapier's rule generalization algorithm without modification. Also, the fact that this algorithm was highly dependent on the answer key provided to Rapier reinforced our decision to abandon this specific generalization algorithm entirely.

Our implementation of AutoSlog-TS returns the extraction patterns in the form of <noun phrase><verb phrase><noun phrase> which aligns nicely with the pre-filler, filler, and post-filler slot arrangement of the rule generalization phase. We set up three constraints for the pre-filler and post-filler slots and one constraint for the filler slot. The pre and post filler constraints consist of the maximum number of words in the noun

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

phrase, *max-len*, a 39-dimensional vector that serves as a histogram bin [2] for the POS tags of the words in the noun phrase, *POS-classification-vector*, and a required POS tag, *required-POS*, that is determined from the noun phrase's *POS-classification-vector*. The 39-dimensional vector was composed of 36 Penn Treebank part of speech tags and three tags that were added later to cover punctuation marks. The constraint associated with the filler slot is just a set of words in the verb phrase, *filler-set*. Like Rapier, MAXIM avoids this problem by generalizing rules using the slot constraints.

2.3 Problem Domain

Our problem domain is articles reporting the results of soccer games. It was chosen based on a mutual interest and the wide availability of such stories. In order to diversify our training corpus as much as possible while staying in the domain, we have collected stories from different countries and authors, resulting in texts with different writing styles. All articles are in English. Our sources include, the FIFA coverage of the World Cup 2002, the English Premiere League's archives of the past seven years, as well as the Major League Soccer's archives over the past six years. All sources were downloaded from the World Wide Web and have been stripped of all HTML tags.

Having considered various choices for the irrelevant text to be input into AutoSlog-TS, we decided that it would be beneficial to try different sources in order to compare and contrast how the choice of irrelevant text affects results. We have picked the Wall Street Journal corpus for its journalistic style and its specific and consistent subject matter. Conversely, we have chosen the Brown corpus for its broad range of writing style and diverse subject matter.

2.4 Building the Relevant Corpus

Once all the HTML tags had been stripped from our compilation of soccer stories, we performed two tasks in order to convert our relevant data to a corpus that was compatible with both the AutoSlog-TS and Rapier systems. The first task was to remove text that was not in a sentence format such as headers and game statistics. The second task was to put the soccer stories in one-sentence-per-line format. As a result, we implemented a highly customized sentence boundary disambiguator which included some common proper nouns from the problem domain. The fnTBL (Ngai and Florian, 2001) POS tagger and text chunker was then run on the formatted text, which included about 800,000 tokens.

[2]It is not technically a histogram, because the POS counts for the phrase are weighted. More common tags, like NN, for example, are weighted less than tags like CD, which carry more information.

PATTERN	**EXAMPLE**
<**subj**> passive-verb	<team> was defeated
<**subj**> active-verb	<player> scored
<**subj**> verb infin.	<team> attempted totie
<**subj**> aux noun	<player> was team
passive-verb <**dobj**>	kicked <player>
active-verb <**dobj**>	beat <team>
infin. <**dobj**>	to card <player>
verb infin. <**dobj**>	tried to foul <player>
gerund <**dobj**>	coaching <team>
noun aux <**dobj**>	coachName is <coach>
noun prep <**np**>	goal against <team>
active-verb prep <**np**>	beat by <goals>
passive-verb prep <**np**>	was injured at <time>

Figure 2: AutoSlog Heuristics

2.5 Implementing AutoSlog-TS

Though Riloff generously offered her AutoSlog-TS implementation for our research, we obtained the code too late to make use of it. Also, since modifications were necessary to the rule representation, time to become familiar with the code would be called for. For these reasons, we decided to implement what we needed of AutoSlog-TS ourselves. Due to the extensive need for using regular expressions and the limited time allotted for development, we decided to implement AutoSlog-TS with Perl. AutoSlog-TS generates extraction patterns for a given text in two stages.

2.5.1 Stage 1

In the first stage, AutoSlog-TS identifies the noun phrases using a sentence analyzer.[3] For each noun phrase, it uses 15 heuristic rules to generate extraction patterns that will be used in the second stage. These heuristic are shown in Figure 2.

When the first stage is run on the corpus, a huge dictionary of extraction patterns is created. This list of extraction patterns is capable of extracting every noun phrase in the corpus. AutoSlog-TS allows multiple rules to be activated if there is more than one match. This results in the generation of multiple extraction patterns for a single noun phrase. For example, running our implementation of AutoSlog-TS on a test set of the WSJ, the sentence "... have to secure additional information and reports ..." produced two patterns: **have to secure** <**dobj**> and **to secure** <**dobj**> in response to the two of the rules in

[3]We used the pre-trained englishTextChunker that comes as part of fnTBL.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

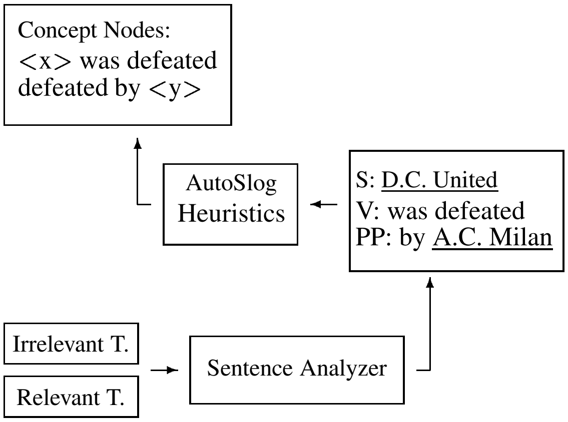


Figure 3: AutoSlog-TS Stage 1 flowchart

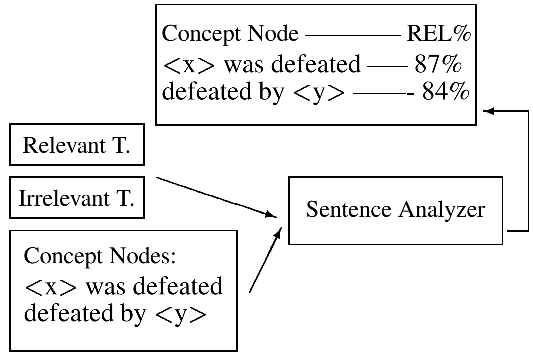


Figure 4: AutoSlog-TS Stage 2 flowchart

Figure 2, verb infin. <**dobj**> and infin. <**dobj**>, respectively. The relevance statistics performed in the second stage will decide which one of the two extraction patterns is more representative of the domain.

The process of Stage 1 is summarized in Figure 3.

2.5.2 Stage 2

In this stage, we examine both relevant and irrelevant texts to compute the relevance statistics for each pattern. For each pattern in our dictionary, we go through each sentence in both our relevant and irrelevant corpora and keep track of the number of cases that were activated by this pattern. We use this to estimate the conditional probably that a text is relevant given that it activates a given extraction pattern according to the formula:

$$P_r(rel\text{-}text \mid text\ contains\ pattern_i) = \frac{rel\text{-}freq_i}{total\text{-}freq_i} \quad (1)$$

where $rel\text{-}freq_i$ is the number of times $pattern_i$ appeared in the relevant corpus while $total\text{-}freq_i$ is $rel\text{-}freq_i + the\ number\ of\ times\ pattern_i$ appeared in the irrelevant text. The idea behind computing the conditional probability for each pattern is that domain-specific expressions will show up more in relevant texts than irrelevant ones. Since AutoSlog-TS generates thousands of extraction patterns, we need to rank each pattern in order of relevance to the domain and discard the less important ones. This will allow for a person to review the most relevant patterns. The ranking function used by AutoSlog-TS is a simple one:

$$rank_i = relevance\ rate_i * log_2(total\text{-}freq) \quad (2)$$

where $relevance\ rate_i$ is as calculated by $eq(1)$ and $total\text{-}freq$ is also the same as in $eq(1)$. Rank is set to 0 if $relevance\ rate_i$ is ≤ 0.5. $Eq(2)$ gives a higher rank for patterns with a high frequency. This was done to save patterns that are common in both relevant and irrelevant texts from poor ranking. Both equations 1 and 2 were originally used in AutoSlog-TS. Riloff admits that they are simple and may be improved upon but they were suitable for her purposes. Even though we feel similarly, strictly improving AutoSlog-TS was not the focus of our work, so we decided to use the equations as presented.

The process of Stage 2 is summarized in Figure 4.

2.6 Rule Clustering & Generalization

When AutoSlog-TS outputs its rule extraction patterns and sorts them according to the ranking function, there is a danger that some important extraction patterns might be discarded. For example, in our problem domain of choice, the phrase "<Rookie Ali Curtis > netted <his second goal>" might have been ranked a lot higher than "<Forward Chris Brown> notched <a goal>". If our rule representation relied solely on the words it found in the training text and their ranks, these would be treated as separate rules throughout the program and the second phrase may be discarded as irrelevant if the top x rules are blindly selected.

However, MAXIM keeps all the rules from AutoSlog-TS, computes the *POS-classification-vectors* for both noun phrases (i.e. the pre- and post- fillers) and the *filler-set* for each rule and runs a two-level clustering program. This program first clusters the rules with the same *filler-set* together. It then calculates the average *POS-classification-vectors* of these simple clusters ($simple\text{-}cluster_i$ to $simple\text{-}cluster_n$) and computes the cosine similarity between all vectors using two n x n matrices (one for the pre-filler slot and the other for the post-filler). Next it chooses the pair of simple clusters that are most related by finding the pair whose pre-filler and post-filler cosine similarities' sum is highest as long as the pre-filler similarity's value is above a set threshold and the post-filler similarity's value is above a separate threshold.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

Pre-filler:	Filler:	Post-filler:
1) *max-len*	1) *filler-set*	1) *max-len*
2) *POS-classification-vector*		2) *POS-classification-vector*
3) *required-POS*		3) *required-POS*

Figure 5: MAXIM Rule Generalization

Then it continues this process, not considering the previously paired simple clusters, finding the next most related pair of simple clusters until either all clusters are paired or there are no two clusters with both pre- and post-filler similarities higher than their respective thresholds. The second stage of the clustering program can be repeated, cutting down the number of clusters by half each time.

Once all the rules from AutoSlog-TS are clustered in their respective group, a human will go through each cluster and link the pre- and post-fillers to the appropriate slot(s) in the template to be filled. Irrelevant clusters are eliminated at this point and the good clusters are assigned template slots and fed in to our rule generalization program. This is the only phase that requires human involvement. We timed ourselves doing this task together on 628 simple clusters and it took us just under one hour. Compared with the 5 hours it took the AutoSlog-TS team to manually assign slots to approximately 2,000 rules and you find that we are already saving time, and this was before the second stage of the clustering, where we cut those 628 simple clusters into less than 314 clusters.

The generalization program is very similar to our clustering algorithm since it relies heavily on POS information. As discussed in section 2.2, the three constraints for the pre and post filler slots are *max-len*, *POS-classification-vector*, and *required-POS*, and the only constraint for the filler slot is the *filler-set*. The *POS-classification-vector* for each cluster is computed by just taking the average of the individual *POS-classification-vectors*. From this vector, we obtain the *required-POS* by finding the maximum element. When the rules are applied to a test set, a sentence has to satisfy a total of four constraints before the content of its pre- and post-filler slots are sent to the template slot as important information. The structure of our rule representation is summarized in Figure 5.

2.7 Discussion of Results

We set out to fill a template with five slots: the name of the teams that played, the winner of the game, the score, the scorer(s), and the names of people who were ejected from the game. MAXIM allows for multiple values per slot using a comma as the delimiter. Ideally, the slots are filled with only the relevant information. However, the slots are generally filled with entire noun phrases, which contain the desired information (as the example output in Figure 6 shows). Note that this is still better than some research-level systems, which return the entire sentence that contains the desired information.

Figure 6: A Filled Template

soccer−game template

teams: The Colorado Rapids, the Los Angeles Galaxy 1 - 0
score: the Los Angeles Galaxy 1 - 0
winner: The Colorado Rapids
scorer: Agogo
ejected: Galaxy defender Ezra Hendrickson

	WSJ		Brown	
	Recall	Precision	Recall	Precision
1^{st} 25% of corpus	19.85%	68.97%	19.54%	65.31%
2^{nd} 25% of corpus	10.53%	61.33%	11.91%	64.2%

Table 1: Effect of Writing Style and Irrelevant Text Choice on Performance

Although our training corpus comprises 1,300 soccer articles (800,000 tokens), it was not possible to train our implementation of AutoSlog-TS within the given time frame. As a result, we trained on only $1/4$ of our corpus size. This made it possible to analyze the effect of different writing styles within our problem domain on the performance of our system, as we could pick different parts of our corpus that correspond to the different soccer leagues. We tested MAXIM on 200 articles and calculated recall and precision by comparing the templates filled out by MAXIM to a human-filled answer key. As can be seen in Table 1, both recall and precision were better when we trained on the first 25% section (section A) of the training corpus than the second 25% section (section B). This is believed to be due to the fact that this second section contained mostly articles about Premier League games while the test corpus contained only articles from the 2000 MLS season. The writing styles from the British writers and the American writers varied greatly. For example, where one British writer writes, "<player> dinked a delightful cross", the average American writer writes, "<player> blasted a cross".

In addition, the result of different choices of irrelevant text was analyzed by training our system on both the Wall Street Journal (WSJ) and the Brown corpora[4]. We were hoping to show that training on the WSJ corpus would lead to better results than training on the Brown due to the commonality of the journalistic writing style between

[4]Of course, the size of these corpora is proportional to that of the relevant text.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

		WSJ		Brown	
		Recall	Prec.	Recall	Prec.
1^{st} 25%	*team*	24.74%	72.39%	23.47%	74.80%
	score	11.86%	95.83%	11.86%	92.00%
	winner	15.34%	83.33%	7.98%	86.67%
	scorer	21.26%	60.00%	23.56%	55.91%
	ejected	10.26%	100.0%	12.82%	55.56%
2^{nd} 25%	*team*	9.44%	52.11%	12.50%	61.25%
	score	4.64%	81.82%	5.15%	83.33%
	winner	7.36%	75.00%	7.98%	68.42%
	scorer	15.13%	62.70%	15.90%	63.36%
	ejected	2.56%	100.0%	2.56%	100.0%

Table 2: Breakdown of Results According to Slots

	team	*scorer*	*winner*	*score*	*ejected*
Rules	73	39	12	7	4
Recall	24.74%	21.26%	15.34%	11.86%	10.26%
Prec.	72.39%	60.00%	83.33%	95.83%	100.0%

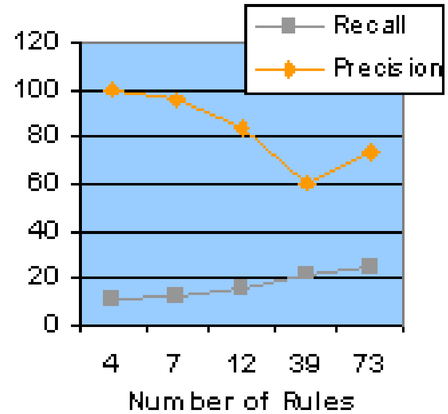
Table 3: Number of rules assigned to each slot in the Section A/WSJ rule-set compared to both the recall and precision for the slot.

this corpus and our relevant training corpus. We thought that a group of common journalistic phrases may appear a lot in the relevant training corpus but hardly at all in the Brown and thus be deemed as relevant. The results are inconclusive. We have not been able to link any effects to the choice of irrelevant text.

The breakdown of our results according to the five slots in our template is shown in Table 2. Our recall was generally best for the *team* and *scorer* slots. This is true in our results from section A as seen in the first half of Table 2. These recall values are followed, in descending order, by *winner*, *score*, *ejected*. These numbers correspond exactly with the number of rules that were assigned to these slots as clearly shown in Table 3. It is not surprising that the more rules assigned to a slot, the higher the recall for that slot, as there are a greater number of ways to fill the slot.

The precision values are ordered in exactly the opposite way (with the exception of the *scorer* slot) as is also seen in Figure 7. This is inversely related because the greater the number of rules, the greater the possibility of a mistake. Also, precision is dependent on how specific a rule is to the desired information and how commonly it is used in the problem domain. For instance, the most common rule to fill the *scorer* slot often appears something like, "Diallo scored in the 17th minute". However, it is also very common to read, "D.C. United scored in the 17th minute". Despite the presence of this second form, we must assign this rule to the *scorer* slot since this is the most likely way that we will find this information. Here

Figure 7: Recall/Precision as Functions of the Number of Rules



		WSJ		Brown	
		Recall	Prec.	Recall	Prec.
Stage 2 done once	1^{st} 25%	19.85%	68.97%	19.54%	65.31%
	2^{nd} 25%	10.53%	61.33%	11.91%	64.20%
Stage 2 done twice	1^{st} 25%	10.84%	79.44%	4.78%	74.12%
	2^{nd} 25%	2.65%	64.81%	4.62%	74.39%

Table 4: Clustering vs. Recall and Precision

we are sacrificing precision in order to increase recall.[5] We found that these kinds of decisions were required frequently. Unfortunately, we were not able to experiment enough to become adept at choosing the best choice. We believe that this, poor choice of rules to keep/delete and slots to assign them to, played a role in our less-than-ideal results.

The effect of the number of times the clustering algorithm was run on the performance of our system was also examined. The second stage of our clustering program reduces the number of clusters by half every time it is run, cutting the human involvement time by the same amount. However, this is done at the cost of recall, as shown in Table 4. The most likely reason for this is that more good rules are discarded because they were grouped with three other irrelevant rules. This decreases the number of rules assigned to each slot, which, as was seen in Figure 7, directly influences recall. The over-all increase in precision when clustering again can be explained by the same logic, though it is somewhat counterintuitive. One may very well expect that precision would decrease with increased clustering. This was not our experience, though we still expect that this behavior may be seen if one were to cluster more than the two times that we did.

[5]Examples like these are suspected to be the reason for the low precision of the *scorer* slot because they are very common, thus the anomaly in Figure 7.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

What was reassuring were the results of the clustering. After clustering twice, with nothing but part-of-speech information and *filler-set* information, we were left with less than $1/4$ of the ”rules” that we started with. [6] Of these ”rules”, or rather, clusters of rules, some very different *filler-sets* were quite correctly grouped together. Some of these include {*blanked, defeated, rocked, rattled*}, {*knotted, narrowed, had scored, pressed*}, {*netted, notched, missed, scored in*}, and {*was issued, was shown, assumed, was ejected for*}. Obviously, ”missed” and ”assumed” do not belong in their respective clusters, and ”pressed” is certainly arguable at best, but the rest of these results are fairly remarkable for the simplicity of the system in its current state.

Besides the potential improvements discussed in Section 2.8 that could be added to improve MAXIM, there were some issues that could be held accountable for the low recall and unexceptional precision. First, in the problem domain we were dealing with, figurative/descriptive language was used heavily in order to bring the excitement of the game to the article. This results in the same idea being expressed in a multitude of ways and with a variety of action words. We could not adequately train for this since it just took too long. Resorting to training on $1/4$ of our corpus hurt us here. This is not as big a problem for the domains many other IE systems have chosen to test on.

Secondly, we found it quite common for authors of an article to refer to the previous week's games, or other games from earlier in the season. So if MAXIM finds ”<player> scored 3 goals” it has no way of determining whether or not this happened during the current game or a previous one and must assume the former. We automatically get this wrong if the phrase in question is not describing the current game.

Another problem we are sure that we ran into to some degree is human error in both generating the answer keys and comparing our results to the answer keys. For one thing, any of the problems that MAXIM runs into when extracting information from an article, a human runs into as well. Also, there is always question about what to do about things like own goals in soccer. Who to mark as the scorer is likely up to debate and this results in inconsistencies.

The last major problem that we had was errors from pre-processing. Some abbreviations escaped detection by our sentence boundary disambiguator. Words like ”give-and-go” and ”game-winner” were split into multiple parts and the hyphens were chunked as being outside of the surrounding noun-phrase by fnTBL. This broke up some noun-phrases that would have otherwise held extractable information. Finally, and mistakes made by the POS tagger directly effected our performance.

2.8 Conclusions

As it is clearly seen in the Results section, MAXIM suffered from low recall. There are several factors that might be responsible for this. Working towards the elimination or minimization of these factors, we believe, will improve both recall and precision.

The ability to train on the whole corpus may improve recall as more rules will be detected. This will also avoid the writing style dependency discussed in the Results section. If MAXIM trains on MLS, Premier League, and World Cup stories (the whole corpus), it will have a more generic and style-insensitive rule dictionary.

In addition, we might have been too restrictive and inflexible with our AutoSlog-TS's heuristic rules. The addition of rules to Figure 2 that are targeted to our problem domain and the elimination of those which are unlikely to be useful will result in a more refined AutoSlog-TS output. We also required our rules to be in *¡subj¿ verb ¡dobj¿* format in order to make the two phases of MAXIM compatible (see Section 2.2). However, if we allow *¡subj¿ verb* and *verb ¡dobj¿* to exist by themselves (which means that we have to allow for empty pre- or post- fillers), we could improve our recall.

Furthermore, we would like to minimize the exact-word-matching dependency of the filler slot. The implementation of this phase was considered using WordNet but preliminary trials indicated this to be a futile effort. For example, WordNet will not consider {*blanked, defeated, rattled, rocked*} to be in the same semantic class. Even though these words were grouped together by our clustering algorithm to form the *filler-set* for the *winner* and *team* slots, MAXIM will depend on finding one of these words in the test set during the application of the generalized rules. The sentence *¡Team A¿ crushed ¡Team B¿*, for example, will not help us extract *team* and *winner* information. The use of a thesaurus or ontology would have been ideal if, but they just do not currently have a rich enough synonym base to be of any practical, real-world use in any specific problem domain. At least not one where figurative language is used as often as in the domain we trained and tested on. It is worth noting that when Rapier incorporated WordNet senses in its rules, its performance was not significantly better than without this information.

Incorporating position information in our *POS-classification-vectors* might improve our clustering. Currently, we only keep track of the frequency of each POS tag within a noun phrase. If this can be extended to include some kind of contextual information, e.g. the position of POS tags or POS tag n-grams, we may be able to more accurately group different clusters expressing the

[6] We had less than $1/4$ of the rules because the clustering algorithm discards any rules that are not similar enough to another rule.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

same idea together. This would both decrease the human involvement required, by decreasing the amount of attention necessary when examining clusters, and increase precision, by minimizing the number of irrelevant rules in any cluster.

Faults aside, we believe we have presented a novel approach to information extraction that requires less human preparation and supervision than any system available at this time. With further research into how to most effectively translate AutoSlog's rules to MAXIM's/Rapier's format recall can be greatly increased. Similarly, more work looking into a comprehensive list of extraction patterns for AutoSlog-TS would be of great benefit. Improved relevancy-rate and ranking functions for AutoSlog-TS would help to get good rules to the top of the list.[7] As mentioned above, an adequate thesaurus/ontology would be of the greatest benefit when it comes to generalization. Also, an improvement on the representation of the *POS-classification-vector* would be helpful. With improvements made in these areas, MAXIM may prove to be as effective as other systems available while requiring at most half as much human involvement. This is a big step towards the development of IE systems practical for real-world, multi-domain use.

References

Mary E. Califf and Raymond J. Mooney. 1997. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the ACL Workshop on Natural Language Learning*, pages 9-15. AAAI-Press, Menlo Park, CA.

G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. 1993. Introduction to WordNet: An on-line lexical database. Available by ftp to clarity.princeton.edu.

G. Ngai and R. Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of NAACL'01*, pages 40-47.

E. Riloff. 1993. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811-816. AAAI Press/MIT Press.

E. Riloff. 1996. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044-1049. The AAAI Press/MIT Press.

[7] Although we were able to abandon looking at the ranking of rules altogether. The clustering was so effective and time-saving that we sent all of the rules AutoSlog-TS outputted into the clustering stage and just deleted bad clusters, taking care of multiple rules at once.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 84–91
Computer Science Department, Swarthmore College

Latent Semantic Analysis for Computer Vocabulary

Andrew Stout & Todd Gillette
Dept. of Computer Science
Swarthmore College
{stout,gillette}@cs.swarthmore.edu

Abstract

In the work described in this paper we undertook a fundamental, proof-of-concept exploration of an unsupervised technique for extracting and representing word meanings, called Latent Semantic Analysis. In this paper we detail an implementation of LSA for the purpose of vocabulary acquisition and report preliminary results from testing it on English vocabulary tests akin to the Test of English as a Foreign Language (TOEFL). We encountered several difficulties, which are also described.

1 Motivation

Most natural language processing research, and particularly most statistical natural language processing research, focuses on formal aspects of natural language: parsing and part-of-speech tagging are concerned with the rules of syntax, morphology is concerned with capturing and employing the formal rules of morphology in languages. While the machine translation task is certainly concerned with semantics, current approaches work mostly by considering formal structures and formal methods for word-alignment. In general, rather little attention is given to developing computational systems for understanding the *meaning* of natural language. We feel that in order to achieve the near-human level of natural language competence which must be the ultimate goal of Natural Language Processing, NLP research must confront the problem of meaning.

At its core, the problem is a familiar one, at least in concept, to researchers in Artificial Intelligence: it is the problem of symbol grounding. How can a computer understand what, say, an apple is, that is what the word "apple" means, if it is unable to experience fruit in any of the ways humans do, and by which humans ground their understanding of the symbol in natural language meaning apple? A computer has no experience of taste, or hunger, or touch, or even sight in most NLP settings. Clearly, a conventional computer's understanding of "apple" is going to be fairly limited, but it need not be limited to merely "noun, for plural add -s, Apfel in German", etc. What computers do have access to is a large volume of text, from which subtle statistical features and constraints can be gleaned.

Significant and fruitful (no pun intended) work has been carried out in the construction of semantic ontologies such as WordNet (Fellbaum, 1998), which capture important information about the semantics of words. However, such efforts are dependent on a great deal of human labor, subject to annotator bias, language-specific, and above all still lack a lot of information about the relationships between various words, categories, concepts, etc. What we seek is an unsupervised algorithm which constructs a representation allowing the computer to ground its understanding of the word "apple" in its 'experience' of 'reading' about apples in a vast amount of unannotated text.

2 Latent Semantic Analysis

Latent Semantic Analysis (LSA) (Landauer and Dumais, 1997) is a statistical method for quantifying meaning in natural language, predicated on the notion that the entirety of the contexts in which a word does or does not appear provide a set of constraints capturing the meaning of that word. The basic approach is to represent words as high-dimensional vectors, where similarity in meaning can be calculated as a function of the difference between two vectors.

Our goal in the work described in this paper was to engage in a fundamental exploration of LSA, for the purpose of solidifying our own understanding of it, and to attempt to replicate the encouraging results of others' early work on LSA.

The remainder of this paper is organized as follows.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 92–96
Computer Science Department, Swarthmore College

The next section describes related previous work and attempted or potential applications of LSA. Section 4 contains a detailed description of the LSA algorithm, and an explanation of our implementation. Section 5 explains our testing of our system, and section 6 analyzes our results. Finally, we conclude with a more general discussion in section 7.

3 Related Work

3.1 Method

The Latent Semantic Analysis method was pioneered by Landauer and Dumais (Landauer and Dumais, 1997), and is closely related to Latent Semantic Indexing (Deerwester et al., 1990). The computational basis for LSA is described in (Berry et al., 1993). The method is described in section 4 below.

3.2 Applications

Many potential and already realized applications of LSA exist. Perhaps the most fundamental is simply to apply LSA in order to obtain a computational representation of meanings of words. More advanced work on understanding has also been undertaken, such as understanding metaphors (Kintsch and Bowles, 2002) (Kintsch, 2000). LSA has been trained on aligned corpora in different languages to be applied to machine translation (Landauer et al., 1998b), although LSA by itself has no production capability. By averaging the vectors of a paper and comparing to multiple other source documents one can determine which sources were most useful to the writer of the paper. By modeling the meaning of a large document, sentences which most closely match the meaning of the whole document can be identified for the purposes of summarization (Kintsch et al., 2000). By comparing the vector representations of one sentence to the next, LSA can provide a measure of cohesion within a text (Landauer et al., 1998b). LSA systems can also be trained to estimate the evaluation of an essay, and to choose an appropriate next text to maximize student learning (Landauer et al., 1998b). LSA has also been integrated into computer tutoring systems (Wiemer-Hastings, 2002).

The most exciting applications of LSA, however, are those yet to come: LSA offers the potential to develop NLP systems that understand the meaning of language and can process or do useful things–from better automatic translation to passing the Turing test–based on that understanding.

4 Implementation

4.1 Corpus

Latent Semantic Analysis learns word meanings through processing a large volume of unannotated training text; this corpus corresponds to what the system 'knows' after training. We started LSA runs on a subset of the Encyclopedia Britannica (EB) containing 4,148 unique words, factoring out all one letter words, two letter words, and words that appeared in only one document. There were 588 documents corresponding to encyclopedia entries, each of which was approximately a paragraph long. After running through the system with the Encyclopedia Britannica corpus, we chose the 1989 Associate Press corpus for our full-size corpus, using the natural division of an article as a document. The corpus contains over 98,000 documents and over 470,000 unique words.

4.2 Preprocessing

Once the training corpus has been divided into documents, our system builds a large matrix X of word occurrences. Each entry $x_{i,j}$ corresponds to the number of times word i appears in document j. Each entry is then transformed to "a measure of the first order association of a word and its context" by the equation

$$\frac{\log\left(x_{i,j}+1\right)}{-\sum_j\left(\left(\frac{x_{i,j}}{\sum_j x_{i,j}}\right)\cdot\log\left(\frac{x_{i,j}}{\sum_j x_{i,j}}\right)\right)}$$

(Landauer et al., 1998b).

4.3 Singular Value Decomposition

The mathematical underpinning of LSA is a linear algebraic technique called Singular Value Decomposition (SVD), which is a form of eigenvector-eigenvalue analysis which states that any rectangular matrix X can be decomposed into three matrices W, S, and C which, when multiplied back together, perfectly recreate X:

$$X = WSC^T$$

where

- X is any w by c rectangular matrix
- W is a w by m matrix with linearly independent columns (also called *principle components* or *singular vectors*)
- C is a m by c matrix with linearly independent columns
- S is a m by m diagonal matrix containing the *singular values* of X

WSC^T is guaranteed to perfectly recreate X provided m is as large as the smaller of w and c. Integral to LSA is the fact that if one of the singular values in S is omitted, along with the corresponding singular vectors of W and C, the resulting reconstruction $W'S'C'^T = X'$ is the best least-squares approximation of X given the remaining dimensions. By deleting all but the n largest singular

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 92–96
Computer Science Department, Swarthmore College

values from S, SVD can be used to compress the dimensionality of W. When X is a words by documents matrix, the compressed W' is called the *semantic space* of the training corpus from which it was generated.

Before compression, the matrices used for Latent Semantic Analysis are very large: $100,000 \times 98,000 = 8.9 \times 10^9$ elements. However, as any document will only contain a few hundred distinct words, the matrices are very sparse. As we discovered the hard way, cookbook algorithms for Singular Value Decomposition are completely impractical for matrices of this size. Fortunately, libraries for large sparse matrix Singular Value Decomposition do exist (Berry et al., 1993).[1]

It has been found empirically (Landauer et al., 1998a) that an n (dimension) of between 300 and 400 is generally the optimal level of compression to synthesize knowledge for LSA. The exact optimal number of dimensions depends on the particular corpus.

A diagram of our system's architecture is shown in Figure 1.

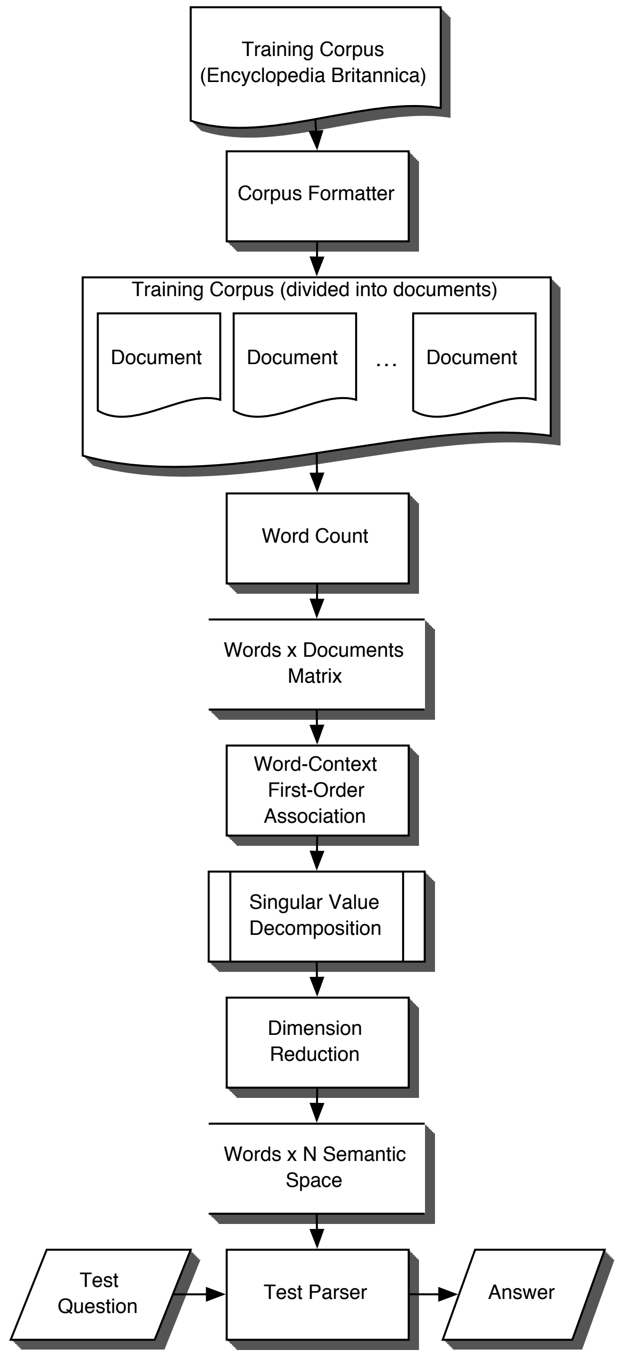


Figure 1: System Architecture

5 Testing

Since we were engaging in a fundamental, proof-of-concept exploration of LSA's potential as a means of capturing semantic information, we chose to test our system on simple vocabulary tests inspired by the Test of English as a Foreign Language (TOEFL) and manually generated using WordNet synonyms from the lexicon of the AP corpus. An example question is shown in Figure 2.

A question is evaluated in the following manner. The LSA vectors of each word in the target phrase ("*levied*" in the example in Figure 2) are averaged to give an average meaning for the phrase. The possible options are then averaged in the same way. Each possible answer is then compared to the target using cosine similarity: the cosine (or, equivalently, normalized dot product) between the target vector and each option vector is computed, and the largest cosine indicates the option with meaning most similar to the target.

6 Results

We tested the Encyclopedia Britannica corpus semantic space on questions developed for the AP corpus. As we expected, the results were poor due to the small size of the EB corpus and the mis-match between training and testing. Many of the words were not even in the EB corpus, and the few questions that had both question and answers produced poor results. We have so far been unable

[1]Unfortunately, the best of them was originally written in FORTRAN, uses counterintuitive data file formats, has completely incomprehensible source code, and is poorly documented (length and mathematical rigor of the accompanying "manual" notwithstanding).

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 92–96
Computer Science Department, Swarthmore College

For the following question, choose the answer phrase whose meaning most closely matches the meaning of the target phrase.
Target: *levied*
Possible answers:
a. *imposed*
b. *believed*
c. *requested*
d. *correlated*

Figure 2: A sample test question.

to complete the preprocessing on the AP corpus so we do not have results for that data set.

6.1 Complications

Our initial tests on a small (5000 word) subset of the Brown corpus made it clear that the computational complexity of the SVD algorithm would be a serious obstacle. Standard cookbook SVD recipes (Press et al., 1997) (which is what we were using) were simply not viable for matrices of the size required for LSA, and both the memory requirements and running time proved to be prohibitive.

The sparse matrix SVD package (Berry et al., 1993) provided an solution for the SVD problem, however a new problem arose when considering a corpus large enough to produce good results. The time necessary to count, process, and compress over 100,000 unique words and nearly 100,000 documents was too great to fit into the time available. We suspect that the first steps will take something on the order of a day or two, but it is still unknown how long the SVD will take. Given more time it might be feasible to carry out the test to completion, but we must at this time be satisfied with tests on a smaller corpus.

7 Conclusion

Latent Semantic Analysis has many useful applications and has been implemented successfully, however there are many difficulties in producing an effective implementation. Once a corpus has been processed and a semantic space has been generated, using it is very efficient and, based upon the work of others, effective.

There is still the question of size. How big must a corpus be to have a certain accuracy? Answering such a question would be more time consuming than testing other aspects and uses of LSA. We do know that the size of the EB corpus was too small, though it was processed very quickly (less than an hour). A corpus of a size in between our EB and AP corpora could be appropriate for testing LSA given the resources of our work space.

8 Future Work

With extra time, within reason, it will be possible to test the AP corpus on the questions. This will allow for a true test of the success of our implementation of LSA. In addition, a subset of the AP corpus might provide positive results in a shorter time frame. In a practical sense, much of our short-term future work would need to be devoted to efficient processing of the extremely large amount of data required for effective Latent Semantic Analysis.

8.1 Domain Specific LSA

Another solution would be to use a corpus of reduced size, though one larger than the EB corpus. Domain specific corpora might provide a means to test the system effectively in a more timely manner. If the lexicon itself were effectively reduced by reducing the scope of the domain, a smaller corpus might turn out to be useful. Some domains would be relatively small, such as news articles about a specific topic, such as baseball. There is a relatively small, consistent lexicon centered around baseball involving bases, hits, home runs, etc. It may be that a relatively small selection of articles from newspapers and/or magazines would produce a working semantic space for the domain of baseball.

(Landauer et al., 1998a) used a psychology textbook in order to make a semantic space of psychology. An introductory textbook would in no way cover all of the terms and ideas contained within the study of psychology, but for the student first being introduced to the field it would suffice. The small semantic space could be used for grading homeworks, evaluating summaries, or comparing test scores with that of a student. It would be interesting to find out what size of a corpus would be necessary for various domains. It would also be interesting to see if a different number of dimensions are found to be optimal for different domains, as well as how dependent that number is on the specific corpus.

8.2 Analysis of Dimensions

Another area of interest that could be explored in the future is that of what the dimensions of the semantic space represent. By taking words that are thought to be strongly related and seeing if there is a particular dimension of which each has a large component, we might be able to find out how semantic space is representing our language. By extension this might offer some insight into our organization of thoughts with respect to language.

References

Michael Berry, Theresa Do, Gavin O'Brien, Vijay Krishna, and Sowmini Varadhan. 1993. SVDPACKC user's guide. University of Tennessee Computer Science Department Technical Report.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 92–96
Computer Science Department, Swarthmore College

S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407.

Christiane D. Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Walter Kintsch and Anita Bowles. 2002. Metaphor comprehension: What makes a metaphor difficult to understand? *Metaphor and Symbol*, 17:249–262.

Kintsch, Steinhart, Stahl, and LSA research group. 2000. Developing summarization skills through the use of lsa-based feedback. *Interactive learning environments*, 8(2):7–109.

Walter Kintsch. 2000. Metaphor comprehension: A computational theory. *Psychonomic Bulletin & Review*, 7(2):257–299.

Thomas K. Landauer and Susan T. Dumais. 1997. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211–240.

Thomas Landauer, Peter Foltz, and Darrel Laham. 1998a. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284.

Thomas K. Landauer, Darrell Laham, and Peter Foltz. 1998b. Learning human-like knowledge by singular value decomposition: A progress report. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1997. *Numerical Recipies in C*. Cambridge University Press.

Peter Wiemer-Hastings. 2002. Adding syntactic information to LSA.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 92–96
Computer Science Department, Swarthmore College

An In Depth Look at Two Approaches to Sentiment Classification

Shannon McGrael
Swarthmore College
CS97 — Senior Conference
smcgrae1@swarthmore.edu

Stephen Michael Smith
Swarthmore College
CS 97 — Senior Conference
ssmith1@swarthmore.edu

Abstract

There have been a variety of approaches to the problem of categorizing text based on sentiment. Many of these approaches require a large amount of user input and effort. In this paper we compare two very different manners of categorizing text, both which requires minimal user input. Both techniques parse through reviews in various domains (video games, movies, etc) and attempt to correctly categorize the review as positive or negative. Through this process we will be able to better understand the problem, as well as the strengths and weaknesses of the solutions. From this we hope to create a basis for developing a more robust system.

1. Introduction

As the world enters into the information age, technologies are emerging that allow faster and more complete access to vast bodies of knowledge and text. The growth in quantity and diversity means that it has become increasingly more difficult to locate text that is particularly relevant to a certain topic. In many cases the shear size of the available data can be a hindrance causing one to find less data rather than more. This obstacle has created demand for a classification system capable of sorting through text, and going beyond a regular search engine by taking a user s personal preferences and the sentiment of the text into account as well. There are undoubtedly millions of applications for this type of technology. What has held it back is the low accuracy of previous results and the need for user input that is difficult to amass. What is needed is a robust system that can accurately perform sentient classification with little input from the user.

2. Applications

Peter Turney (Turney, 2002) and Ken Lang (Lang, 1995) speak of different applications and algorithms for addressing the problem of automated review ranking and text categorization. This technology is useful in search engines, news services, information boards, sales and many other applications.

There are various applications for review ranking in search engines. Turney suggests that a user could type in a query such as Akumal travel review and the search engine could then return the result that There are 5,000 hits, of which 80%

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

are thumbs up and 20% are thumbs down (Turney, 2002). There are millions of objects that are reviewed on the web; the problem is compiling it all together. Users could also ask to see only reviews of a certain ranking. This way they could sort through the reviews to get only the data they needed. This could be useful in immeasurable ways. An artist or any user could easily sift through reviews to understand the criticism of a certain work of art. Web designers could quickly find out what features were liked/disliked by their users. Or it could be as simple as consumers quickly being able to benefit from the opinions of others in an effort to get the most of their money.

Ken Lang (Lang, 1995) uses this technology in a netnews-filtering system called NewsWeeder. This system learns a user's preferences through the reviews they give to certain news articles, and through the reviews that other users like them give to news articles. With this information NewsWeeder is able to present users with more of the news articles that are important and relevant to them and less of those that are not.

These systems could also be used in order to determine when a particular piece of text was unusually positive or negative. Many times it is very important that text be impartial and fair, such as judge s comments, text books, and news articles. Systems such as the ones described below could be used to flag text that is unusually emotional, when that type of response is not appropriate. Likewise these systems may even be able to be adapted to learn to detect ideological differences in articles. For instance a system may be able to detect if a newspaper was particularly liberal or conservative or feminist or anti-feminist.

3. Approaches

3.1 Semantic Orientation

Peter Turney utilized the concept of Semantic Orientation (SO) in his approach to this task (Turney, 2002). A phrase has a positive semantic orientation if it is most often seen in conjunction with known positive words such as excellent . A phrase has a negative semantic orientation if it is most often used in conjunction with known negative words such as poor . His approach to this problem centers on the idea that adjectives and adverbs are often present in evaluative (as opposed to factual) sentences (Hatzivassiloglou & Wiebe, 2000; Wiebe, 2000; Wiebe et al., 2001). This means that they often convey the feelings of the writer more so than any other part of speech. The algorithm attempts to estimate semantic orientation of an entire document (a single review) using information about the adjective phrases within that document.

The first step in this approach is to part-of-speech tag the document and extract all bigram phrases that contain adjectives or adverbs. We used the fnTBL1.0 englishPOS tagger for the POS tagging. The bigrams are extracted because the adjectives or adverbs by themselves don t necessarily give all of the information. This is true because adjectives and adverbs are, by definition, modifiers, so it is easy to see that their semantic orientation is influenced by the words that they are modifying. An example that Turney gives is:

> the adjective unpredictable may have a negative orientation in an automotive review, in a phrase such as unpredictable steering , but it could have a positive orientation in a movie review, in a

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

phrase such as unpredictable plot (Turney, 2002)."

So, the modified word acts as context words to accent the sentiment of the adjective. See Turney 2002 for the rules used to extract usable two-word phrases from reviews.

Turney then estimates the semantic orientation of each adjective phrase by making queries to the AltaVista Search Engine (www.altavista.com), ours looked like this:

(<Word1>+AND+<Word2>)+NEAR+poor.
(<Word1>+AND+<Word2>)+NEAR+excellent.

We then recorded the number of hits for each phrase (we ll call them hitsneg and hitspos, respectively). We also recorded the total number of hits if we just search for poor and excellent individually (we ll call them hitspoor and hitsex, respectively). Using this information, we can estimate the SO for a given adjective phrase by:

$SO(phrase) = \log_2((hitspos * hitspoor) / (hitsneg * hitsex))$.

The nature of this calculation is such that if hitsex > hitspoor, then $SO_{phrase} > 0$. Alternatively, if hitspoor > hitsex, then $SO_{phrase} < 0$.

To compute the SO for a given review, we simply take the average of the SO s for all of the adjectives that we extracted from this document. In order to avoid counting phrases that have very little correlation to either poor or excellent , we remove any phrase which gets less than 4 hits for both of the AltaVista queries from this calculation. Also, to avoid a possible division by zero, we added .01 to the number of hits for every individual phrase. Any document with an SO greater than zero is considered to be recommended, and any document with a negative SO is considered to be not-recommended.

3.2 Bag of Words

Lang s method, commonly referred to as the bag of words method, requires separate training and testing data. The idea behind this method is that similar texts will contain similar words. Lang's method does not take word order or syntax into account. The method assumes that most of the information needed to differentiate between texts is in the words themselves. For the purposes of this research we have implemented a system similar to that described by Ken Lang in his paper: NewsWeeder: Learning to Filter Netnews.

In order to implement the bag of words method, the text must first be parsed separated into tokens, each token being a word or punctuation mark. These tokens are then converted into vectors the length of the vocabulary. One main vector is created for the whole corpus that is a list of every unique token in the corpus. Then each separate document or review has its own corresponding vector that contains numerical values indicating the number of times a particular word appeared in that document. A zero for at the location of a certain word would indicate that it did not appear in the document, whereas a 20 would indicate that it frequently appeared in the text.

In order to complete the learning process a representative vector must be created for each separate category. First the document vectors must be changed into normalized vectors, then by using least-squares regression, a numeric value is assigned to each of them. All the vectors in the training corpus pre-determined to be of the same classification are then averaged to get a representative vector for each category.

At this point the training process is completed. To test the system a review must be parsed and turned into a word

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

vector and then given a least-squares regression value in the same manner as the vectors in the training corpus. This value is then compared to each of the representative category vectors. The system guesses that the new document is of the same category of the representative vector closest to it.

4. Corpus Creation

Our corpus consists of 100 reviews taken from the Epinions web site (www.epinions.com). The original idea of this project was to solve the problem of sentiment classification on video game reviews. However, any game that was popular enough to have more than a handful of reviews were generally given a recommended rating. So, we took 50 reviews of the game Tekken Tag Tournament (10 not recommended , 40 recommended), and we took 50 reviews of the movie Gladiator (25 not recommended and 25 recommended) for our corpus data. Since Lang s algorithm requires training data, we split the two halves of the corpus into 40 reviews for training and 10 reviews for testing. The results given in this paper are for those 20 reviews we used for testing, with additional results for Turney s system having been run on the entire corpus.

5. Results

For the purposes of comparison, we ran both algorithms on the smaller corpus of 20 reviews that we separated from our corpus for testing. The bag of words method uses the other 80 reviews for training purposes. Because Turney s algorithm does not require training data we were able to run it on the entire corpus of 100 reviews.

5.1.1 Turney s Results (Comparable to Lang s Approach)

When run on the test corpus, Turney s approach receives a 40% on the video game reviews, and a 60% on the movie reviews. Tables 2 and 3 show the results we achieved for the test corpus.

Table 2. Results for 10 Tekken Tag Tournament reviews

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	2	2
SYSTEM	NOT RECOMMENDED	6	0

Apparently, the reviewers used some pretty negative word-choice to describe their pleasure with the game. This is understandable since it is a fighting game and we may be getting negative feedback from words that the authors mean to use as positive aspects (i.e. prolific gore , massive hits , etc.).

Table 3. Results for 10 Gladiator reviews

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	2	3
SYSTEM	NOT RECOMMENDED	3	2

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

Again, the reviewers seem to have used phrases which in everyday random text would be viewed as negative, but in the context of the entire document, it should be seen as positive. One example bigram from our corpus is disastrous effects . This bigram appears in a section of the review in which the author is describing something that happens in the movie, not how he/she feels about the movie itself. However, our algorithm would incorporate this bigram into a lower Semantic Orientation score than the review most likely deserves.

Table 3a shows a comparison of actual adjective bigrams tested in reviews that Turney s system correctly classified as recommended or not-recommended.

Table 3a. Comparison of adjectives in correctly classified positive and negative reviews.

SAMPLE BIGRAMS from a correctly classified POSITIVE REVIEW (TEKKEN TAG)	SAMPLE BIGRAMS from a correctly classified NEGATIVE REVIEW (GLADIATOR)
Nice night	Many clues
Extra cash	Upper hand
Good amount	Bad guy
Good characters	Pretentious corniness
Main arcade	Embarrassingly easy
Different stages	Inefficient republic
Pretty cool	Natural enemy
Other modes	Not interested
Basic fighters	Moral development
Different buttons	Not entertained
Fairly new	Loving violence
Special combos	Violent scenes
Good fighting	Elaborate distraction
Awesome graphics	Less distasteful
Widely successful	Misogynistic hatred
Joyous sound	Questionable moments

5.1.2 Turney s Extended Results

Since Turney s algorithm does not require training, we decided it might be interesting to see how it runs on the entire corpus of 100 reviews. The final results for this run are as follows:

1. We correctly classified 21 of the 50 Tekken Tag Tournament reviews, giving a precision score of 42%, which is slightly better than the score we received by only running it on a small number of reviews.
2. We correctly classified 26 of the 50 Gladiator reviews,
giving a precision score of 52%, which is only slightly better than the baseline score of 50% if one were to choose recommended every time.

Table 4a is a compilation of the results from the 50 Tekken Tag Tournament reviews, and Table 4b is a compilation of the results from the 50 Gladiator reviews.

5.2 Lang s Results

For the following tests the data from Tekken Tag Tournament and Gladiator were always tested separately. The corpus was divided into two categories of reviews, recommended and non-recommended. In order to train the system and create the representative vectors for each category in each domain, 40 of the 50 available reviews were read in and put into token vectors, as previously described. The vectors created from the test documents were then compared to these two representative vectors.

The first round of results were very poor. The results were near 50% for both domains. The data was difficult to manipulate because of it very high

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

dimensionality. Also commonly used words such as and , I , is , etc occur so frequently that they were outweighing other words that would have had much more descriptive power. Because of this the representative vectors for the two separate categories were so similar that it was very difficult to differentiate them.

To improve upon our results we implemented a pruning method that would limit the number of words that the system took into account. After all the training and testing data was read into the system. Each token that appeared more than X number of times in the whole corpus was eliminated from the word vectors. Then these pruned vectors were normalized and there cosine similarity was found. This has the effect of disregarding words that are used frequently throughout the corpus and that have no relevance in determining the category of a particular document. It also reduced the dimensionality of the vectors making them more precise and easier to manipulate.

The bag of word s approach is correct 40% on the movie reviews, and a 90% on the video game reviews. Tables 5 and 6 show the results we achieved for the test corpus in two different formats.

Table 4a. Results from the complete corpus of 50 Tekken Tag Tournament Reviews

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	18	7
SYSTEM	NOT RECOMMENDED	22	3

Table 4b. Results from the complete corpus of 50 Gladiator Reviews

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	9	8
SYSTEM	NOT RECOMMENDED	16	17

Results for 10 Gladiator reviews
Table 5b.

Actual Rating	Our Rating
N	Y
N	N
N	Y
N	N
N	N
Y	N
Y	N
Y	Y
Y	N
Y	N

Table 5a.

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	8	1
SYSTEM	NOT RECOMMENDED	0	1

6. Difficulties

As we attempted to create these systems we came upon many obstacles. Some of these obstacles were unique to our chosen method and others were related to the problem and the way people express themselves.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

Results for 10 Tekken Tag Tournament Reviews
Table 6a.

		ACTUAL	
		RECOMMENDED	NOT RECOMMENDED
OUR	RECOMMENDED	2	3
SYSTEM	NOT RECOMMENDED	3	2

Table 6b.

Actual Rating	Our Rating
N	Y
N	N
Y	Y
Y	Y
Y	Y
Y	Y
Y	Y
Y	Y
Y	Y
Y	Y

The test results can be found in greater detail in appendix A.

One of the largest problems that we both should have expected but that became very apparent as we sorted through our text and results was that people use very different ways of expressing themselves. This problem took a few different forms. What one person may consider to be a negative attribute others would consider a positive. One line of text could be indicative of contrasting emotions depending upon the person, their manner of speech and their likes and dislikes. For example: I had never seen a movie so focused on war, destruction and lifelike, gory violence as Gladiator. While one reviewer may hate the movie for its extensive action scenes and gory details, others may find that to be one of its most exciting aspects.
The other case where the diversity of people is a big obstacle is in the inconsistency of their rankings. Each review was written by a different person with a different concept of what recommended and non-recommended means, where to draw that line, and for what reasons. For example, about half the people that ranked Gladiator as three stars recommended it and the other half did not. But, even more difficult to deal with were the cases where a reviewer would rank the movie as one or two stars, however would still recommend the movie.

6. Pros and Cons

6. 1 Turney Pro s and Con s

In general, this is a very simple algorithm. The system is standalone, and requires no training time or training data. This must be why he called his system "unsupervised," even though by hard-coding the words "poor" and "excellent" into the estimation algorithm, he automatically gave his system prior knowledge that was helpful in solving the problem.

This system is easily portable to other domains while remaining generalizable within the domain of reviews. Turney showed that he could get much better results with reviews from other sub-domains such as automobile reviews and bank reviews (Turney, 2002). Also, one could replace the words "poor" and "excellent" with "conservative" and "liberal" to create a system which could read a newspaper article and report which of those two categories that article was most heavily associated.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

One positive element of this system that was not immediately obvious is how it handles negation. As the bigrams are extracted, given the rules developed by Turney (Turney, 2002), negation words such as "not" are extracted and used in the computation of that document's SO. For example, the bigram "not interested" receives a much more negative SO than the bigram "more interested" which appears in a different review.

Turney's algorithm is easy to implement (low coding overhead), however, this fact also makes it easy to find specific cases that it does not cover. In this way, the generalizability of the system negatively affects its accuracy. The rest of this section will focus on specific problems that we encountered.

While this algorithm takes relatively little time to implement and no time to train, it takes an incredibly long time to issue all of the queries to the Search Engine. In fact, Turney's experiment on 420 reviews took over 30 hours to run (Turney, 2002)! This has an adverse effect on the scalability of the system.

In a review, it is common for the author to compare the subject that they are reviewing with something else that is well-known in that domain. Adjective phrases that are associated with anything other than the reviewed subject should not have the same impact on the semantic orientation of that review. However, with the current approach, we give equal weight to all adjective bigrams, regardless of the subject of that particular sentence.

It seems intuitively obvious that any system attempting to use the entire World Wide Web as a source of information on a language would have many unforeseeable shortcomings which could be very difficult evaluate. For example, any words remotely related to pornography in a review may have huge weight in the semantic orientation of a review, while very specific technical observations may even be ignored by their relative non-presence on the Web.

Lastly, one reviewer may use language that, in the overall scheme of things, is seen as negative in everyday language. However, if the reviewer is using that language as if it is describing positive events, this algorithm will wrongly classify those phrases as actually being negative. Some examples from our corpus include bigrams such as "incredibly gruesome", "mighty blows", and "very painful." Each of these examples remains ambiguous since it is possible that one person would find these things positive in certain contexts.

6. 2 Bag of Words Pro s and Con s

In the bag of words method there were many problems that were unique to the method because it does not take syntax or word order into account, thus making sense disambiguation impossible. For example if a review were to use the word bad, the system is unable to determine if this word means awful, or refer to an evil person, or poor conditions, or even the slang use of the word that has positive connotations. The system lumps all of these uses of the word bad together.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

Another problem can be seen in the following example taken from the corpus: Why is this analogy in the movie? Because it sounds really cool This is an example of one of the many speeches used in Gladiator to make it characters sound neat. This is a negative review that was wrongly classified as positive. The system picks up on the positive words such as really , cool , and neat. It is unable to see that this reviewer is being sarcastic and does not at all like the analogy used or the other long speeches in Gladiator. The system is unable to pick up on this. Another way a very similar problem occurs is with the word not or other negatives. Because words are not taken in context the system could easily misinterpret phrases such as not good or not bad.

However there are also many Pro s to Lang s approach. It is much faster that Turney s approach and therefore much more likely to be useful to applications such as search engines and navigating the web. It is also very easy to implement and to modify. As long as the training data is correctly set up and the number of categories is set correctly the system can deal with any domain.

7. Improvements

7.1 Ways to improve on Turney s algorithm

1. Do some pre-processing on the corpus to tag Subject-Object pairs. This would allow adjective phrases in certain sentences to be weighted more than others. In this case, adjectives in sentences in which the actual product is the subject would have more weight than adjective phrases in sentences about the author s girlfriend.
2. As Turney himself pointed out, the SO estimator is extremely simple. A much deeper mathematical and statistical analysis of the problem could give us a better estimator, and therefore a better system.
3. Over the next few years, technology is becoming faster, cheaper, and more reliable. It seems possible that in a very short amount of time, we will have the processing power to actually process entire documents for meaning, sentiment, etc., instead of doing a token-by-token discrete analysis.
4. This algorithm doesn t necessarily get to the heart of the problem. It seems as though the semantic orientation of a phrase can be modified for the better by one or more informative subroutines (example, see #1).

7.2 Ways to Improve Results from Lang s Method:

1. Pruning the data

The first step in pruning the data provided much better results. However further experimentation in better methods could further improve the results.

a. TF-IDF Weighting

TF-IDF weighting refers to term frequency/inverse-document frequency weighting. This is an accepted and tested technique. It is based on the idea that the more times a token t appears in a document d, or the *term frequency*, the more likely it is that t is relevant to the topic of d. However, the more times that t occurs throughout all documents, or *document frequency*, the more poorly t discriminates between documents. The TF-IDF weight of a token is computed by multiplying the *term frequency* by the inverse of the *document frequency*.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

b. Minimal Description Length (MDL)
According to Lang the MDL principal provides an information-theoretic framework for balancing the tradeoff between model complexity and training error (Lang). This would involve how to determine the weights of different tokens as well as how to decide which tokens should be left out. MDL seeks to find the optimal balance between simpler models and models that produce smaller error when explaining the observed data (Lang).

2. Better Grouping of data

a. Root Words
Words with the same root are classified as unique tokens. By combining these tokens it might decrease dimensionality and give a more accurate view of sentiment.

b. Punctuation
Uninterrupted punctuation is seen as one word. For example !!!!! is in one token instead of being 5 instances of !. Also !!! is a separate token.

8. Conclusion

During our analysis of each of these approaches, we realized that the two systems differ in the type of review that they will correctly classify. Turney's approach works better on reviews with full sentences written as a concise narrative. These reviews should include little extraneous material. Also, this system will lose any benefit of long strings of adjectives, and all punctuation, since these strings will be parsed into bigrams, and will not be viewed as a single adjective phrase. Lang's approach, conversely, works better on reviews written by the colloquial, informal writer. Sentence structure can be ignored and strings of adjectives do not have a negative impact on the ability of this system to classify a review correctly. However this system loses information from negation words such as "not" and "but".

After analyzing their alternative approaches, it seems as though Ken Lang (Lang, 1995) and Peter Turney (Turney, 2002) developed systems that could be easily combined in parallel to create a more robust system. The output of Turney's system is a ranking, from –1.0 to 1.0, with positive rankings indicating a positive review, and negative rankings indicating a negative review. Lang's approach gives a similarity ranking, in degrees, to pre-trained "positive review" and "negative review" vectors. If both systems give the same rank (positive or negative) to the same review, we can be surer about the correctness of our classification. If they disagree, we can normalize Lang's output by dividing by 360, and compare the magnitude of the rankings (by taking the absolute value) given by each approach, choosing the ranking that is closer to 1.0. This would basically choose the ranking from the system that was "most sure" of its ranking.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

Appendix A

Gladiator Results:

Test 1-5 should have been negative and 6-10 should have been positive

PosAvg = 16.0186606915661
NegAvg = 11.8380473345302

TestReview 1 = Positive
Cosine = 23.2709696439365 Positive Average = 16.0186606915661

TestReview 2 = Negative
Cosine = 13.7287385024832 Negative Average = 11.8380473345302

TestReview 3 = Positive
Cosine = 21.0261787760811 Positive Average = 16.0186606915661

TestReview 4 = Negative
Cosine = 11.5234315064659 Negative Average = 11.8380473345302

TestReview 5 = Negative
Cosine = 12.7149150507662 Negative Average = 11.8380473345302

TestReview 6 = Negative
Cosine = 11.4870277804537 Negative Average = 11.8380473345302

TestReview 7 = Negative
Cosine = 13.3473100260505 Negative Average = 11.8380473345302

TestReview 8 = Positive
Cosine = 14.2941176470588 Positive Average = 16.0186606915661

TestReview 9 = Negative
Cosine = 13.1102578104888 Negative Average = 11.8380473345302

TestReview 10 = Negative
Cosine = 11.2413709596575 Negative Average = 11.8380473345302

Tekken Tag Tournament Results:

Test 1 and 2 should have been negative and the 3-10 should have been positive

NegAvg = 11.8229121304717
PosAvg = 13.1909226170445

TestReview 1 = Positive
Cosine = 14.2503313100448 Positive Average = 13.1909226170445

TestReview 2 = Negative
Cosine = 12.5051490549628 Negative Average = 11.8229121304717

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

TestReview 3 = Positive
Cosine = 17.0557835789563 Positive Average = 13.1909226170445

TestReview 4 = Positive
Cosine = 15.4480815863922 Positive Average = 13.1909226170445

TestReview 5 = Positive
Cosine = 15.2916255149102 Positive Average = 13.1909226170445

TestReview 6 = Positive
Cosine = 17.4736133305576 Positive Average = 13.1909226170445

TestReview 7 = Positive
Cosine = 18.4138705812283 Positive Average = 13.1909226170445

TestReview 8 = Positive
Cosine = 16.9111365668947 Positive Average = 13.1909226170445

TestReview 9 = Positive
Cosine = 16.9747545634721 Positive Average = 13.1909226170445

TestReview 10 = Positive
Cosine = 13.3273304529849 Positive Average = 13.1909226170445

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 97–108
Computer Science Department, Swarthmore College

Language segmentation for Optical Character Recognition using Self Organizing Maps

Kuzman Ganchev

Abstract

Modern optical character recognition (OCR) systems perform optimally on single-font monolingual texts, and have lower performance on bilingual and multilingual texts. For many OCR tasks it is necessary to accurately recognize characters from bilingual texts such as dictionaries or grammar books. We present a novel approach to segmenting bilingual text, easily extensible to more than two languages. Our approach uses self organizing maps to distinguish between characters of different languages allowing OCR to be performed on each part separately.

1 Introduction

Modern optical character recognition (OCR) systems perform optimally on single-font monolingual texts, especially when they have been trained on a font very similar to the one they need to recognize. Performance on bilingual texts, however is not nearly as good, especially when the two languages that occur in the text have similar characters. The reason for this is that dictionaries and language models are usually trained on a per-language basis. Sometimes the OCR program will assume that there are multiple fonts for the same language and make poor guesses about which characters are in which language. Despite these difficulties, there are real lines of motivation for performing bilingual OCR. For example is the rapid development of translation systems requires large amounts of training data. There are many languages around the world for which collections of texts as well as dictionaries and grammar books are readily available in printed form, but not in electronic form. For the task of rapidly developing a translation system, optical character recognition may be the only viable solution for obtaining training text.

This paper focuses on a sub-topic of bilingual OCR – namely deciding which characters are in which language. Once this has been done, monolingual OCR can be performed on each of the sections of text and the original document can be subsequently reconstructed. In order to perform this division, we use self organizing maps (SOMs) to distinguish between characters of one language and characters of the other. SOMs are a mechanism for sorting high dimensional vectors into a two dimensional grid, in such a way that similar vectors are sorted into grid locations near each other. A more detailed description of SOMs and how we use them is provided in Section 3.

Our results show that using the approach outlined in this paper, we can correctly determine the language of 98.9% of the characters in Uzbek-English dictionary text. Our experiments as well as these results are described in Section 4.

2 Related Work

2.1 Self Organizing Maps

Kohonen (1990) provides a concise introduction to self organizing maps in general, considering their applications since their conception. He describes how a SOM might work, and notes that it may be necessary to preprocess the information somehow: "it would often be absurd to use primary signal elements, such as . . . pixels of an image, for the components of [a SOM] directly"(Kohonen, 1990). Kohonen presents his SOMs in the context of "Vector Quantization" – the assigning of quantum values to vectors; similar to what we want to do, if the image is the vector, then the quantum representation would be the character that produced it or in the case of language segmentation, the language that that character is in. In this context he also talks about a "codebook" of vectors that are essentially the definition of the quantum to vector relationships. He presents three algorithms for "Learning Vector Quantization" called LVQ1, LVQ2 and LVQ3.

He and Ganchev (2003) use SOMs and neural networks for simple object recognition on a mobile robot, with limited success. The images taken from a camera

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

mounted on a Khepera robot moving in an enclosed space are sorted into a large SOM, after which a neural network is used to associate labels to images.

2.2 Optical Character Recognition

Berman and Fateman (Berman and Fateman, 1994) describe an OCR system for mathematical texts. The main application of their work is to recognize printed tables of integrals, or mathematical proofs. The algorithm they describe requires that images for every variation of every font be learned separately by the program. They use the Hausdorff asymmetric distance function which is defined as the largest Euclidean distance from any "on" pixel in the source image to the nearest "on" pixel in the destination image. One interesting feature of this metric is that in one direction it treats characters with missing "on" pixels as a perfect match, while in the other direction it treats characters with extra "on" pixels as a perfect match. This allows it to deal with broken glyphs and over-connected characters. Unfortunately the authors do not give any numerical results, and the system has to be retrained from scratch for even a slightly different font or font size. Since a maximum of distances is taken for the measure; this approach might also be very sensitive to dust. For example, a spec of dust in the middle of an "O" would make it as far from the model "O" as if it was completely filled in.

3 Abstract Implementation

In order to investigate methods for using SOMs to divide an image into languages, we developed different versions of a system for classifying characters into two sets based on the language which generated them. The first subsection gives an overview of the architecture of this system. Subsequent subsections describe each of the system components in more detail. Subsection 3.2 describes the Gnu Optical Character Recognition program, Subsection 3.3 gives an overview of self organizing maps, and Subsection 3.4 describes the system components that we implemented.

3.1 Architectural Overview

Figure 1 shows a graphical representation of the system architecture. In the figure, the dotted lines represent flow of information, while the solid lines represent the flow of control. Given a source image, we use the Gnu Optical Character Recognition program (GOCR) to find the boxes that contain characters of text. We then wrote a program to extract data vectors corresponding to characters of text from the image using the box descriptions provided by GOCR. Once this training data has been extracted, it is used to train a self organizing map (SOM). We used the SOM_PAK (Kohonen et al., 1995) toolkit for all the SOM tasks. Finally, we use the trained SOM to distinguish between characters of different character sets.

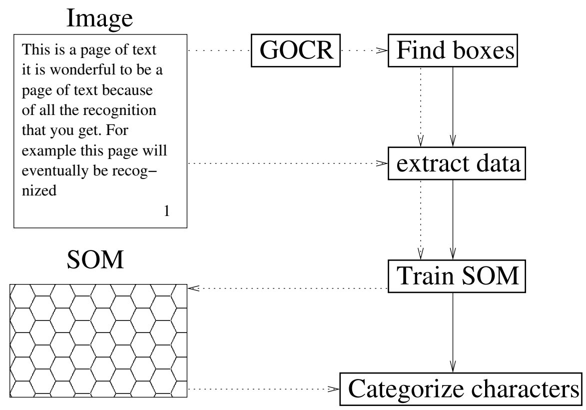


Figure 1: Architecture: The dotted lines represent flow of information, while the solid lines represent the flow of control. GOCR is used to find the boxes that contain characters, the data vectors corresponding to those boxes is extracted and used to train the SOM. Finally the trained SOM is used to categorize the characters into two languages.

3.2 GOCR

GOCR, developed by Joerg Schulenburg and Bruno Barberi Gnecco, is a free optical character recognition system under continuing development. GOCR reads in an image file, and scans it for boxes containing a single character of text. Each box ideally contains a single character and is the smallest rectangular area with sides parallel to the x and y axes. In practice boxes may contain more than one character (See figure 3) but in our experience they rarely contain less than a character. After a list of boxes has been generated, GOCR removes dust and pictures that do not need to be recognized, attempts to detect a rotation angle and lines of text within the scanned image, and attempts to correct glued boxes and broken characters. Finally GOCR runs an OCR engine on the sorted boxes. As an example of its performance GOCR program gives the following output for the image shown in Figure 2; GOCR also added two extraneous new lines between each pair of lines of text, probably because of the high resolution of the source image. The "_" characters represent characters that GOCR was not able to recognize. Figure 3 shows the boxes and lines of text that GOCR has found within the image.

```
itcan go. Butthe"30's
pas_. The eleven mus
ch_sen as the great o
periodbe_een 1940
```


Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

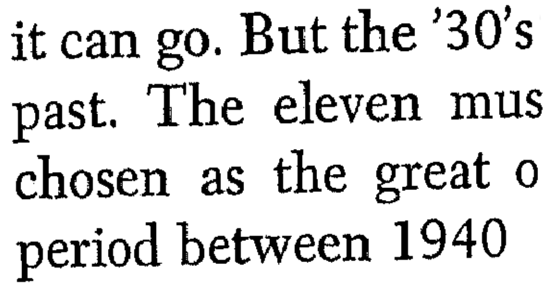


Figure 2: A sample of scanned text used for optical character recognition. Note that the image is very hight resolution, but imperfect and skewed.

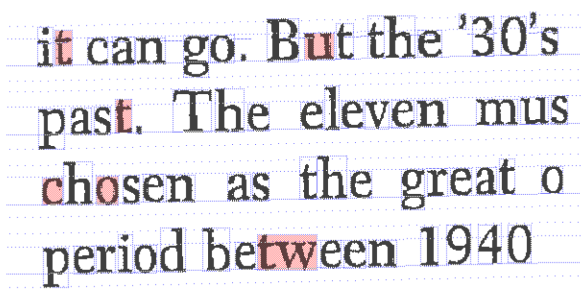


Figure 3: The same scanned piece of text, after processing by GOCR. The blue lines represent the lines of the text as well as the box boundaries. Characters for which GOCR does not have a high confidence are shaded in red. Note that the "tw" shaded in red on the bottom line is in fact a single box. GOCR is not able to recover from this error.

3.3 Self Organizing Maps

Conceptually, a self organizing map (SOM) is a structure for sorting vectors into a grid in such a way that vectors close together on the grid are similar, and vectors farther away are more dissimilar. After training, each grid location contains a vector that represents the ideal vector at that location called a "model vector". Typically, SOM grids are hexagonal, as shown in Figure 4. This allows each cell to be adjacent to six other cells, rather than four with a rectangular grid.

Before the SOM can be trained, it is initialized by setting all its model vectors to random values. Training is done as follows: for each data vector, find the model vector closest to it, and modify that vector and vectors nearby (in the grid) to make make them closer to the data vec-

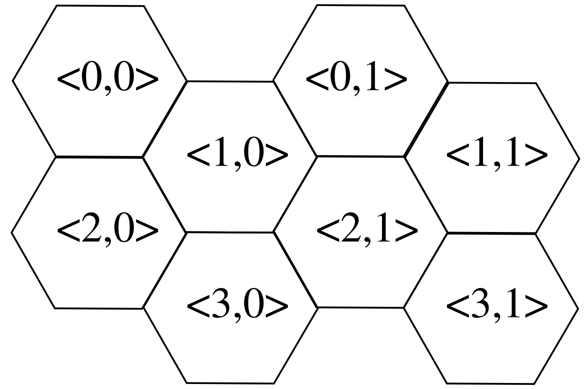


Figure 4: A hexagonal SOM topology. This is a 4×2 SOM – it has four rows and two columns.

tor. We used Euclidean distance to determine how close a data vector was to the model vector. We opted for this approach since it is the default for SOM_PAK; an other option would have been to use cosine similarity as the distance metric. One way to make one vector closer to another is to use a weighted average of the two. The relative weights of the vectors are determined by a learning rate parameter.

There are two stages of training. During the first short stage the learning rate is set high and the neighborhood of the vectors is large, so that the SOM roughly but quickly organizes the vectors into groups. During the longer second stage of training a lower learning rate and smaller neighborhood are set so that local arrangements can be made and the model vectors approach ideal characters.

After the training phase is complete the same vectors used for training can be sorted into locations on the grid. The location and difference from the nearest model for each vector is recorded for use as described in Subsection 3.4. We used SOM_PAK (Kohonen et al., 1995) as the SOM implementation for our experiments.

3.4 Custom Components

Our system relies on a number of small components other than SOM_PAK and GOCR. Section 3.4.1 describes how we convert the images described by the character boxes generated by GOCR, into data vectors used to train the SOM. Section 3.4.2 describes how we use the trained SOM to segment bilingual text.

3.4.1 Converting Images to Vectors

In using a SOM to sort images, we need to first convert the images into some vector representation, since SOMs are designed to work with vectors. We do this by using "1" to designate a white pixel and "0" to designate a black pixel, and then creating a vector that has one dimension per pixel. For example a 3×3 image with black top and bottom rows and a white middle row would be rep-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

resented as "(0,0,0,1,1,1,0,0,0)". This provides a mechanism to convert an image to a vector and vice-versa, but the sizes of the vectors depend on the size of the image, so a "." character, might be a 3×3 image (corresponding to a nine dimensional vector) while a "M"character might be much larger. This complicates matters because a SOM only works with vectors of the same size. We describe two techniques we used to deal with this in Section 4.

3.4.2 After Training

Once we have a SOM trained in one language, we map characters from an unknown language into that SOM. Based on the distance from the character vector to the nearest model vector in the SOM, we decide whether the character belongs to the same language on which the SOM has been trained or to some other language.

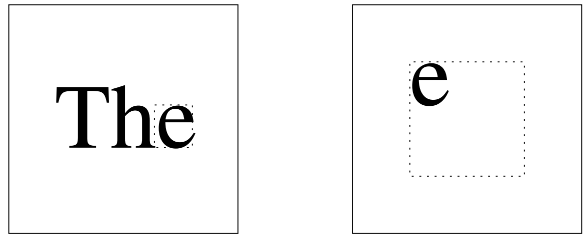
4 Experimental Results

This section describes the empirical evaluation we performed on our system. Subsection 4.1 describes our initial approach for converting character boxes to vectors – placing the box at the top left-hand corner of the image – and describes the problems that produces. Subsection 4.2 describes a refinement to this approach – scaling the characters – and provides comparative results to those obtained in Subsection 4.1. Subsection 4.3 describes how the algorithm performs on hand-selected training data for the task of segmenting dictionary text. Finally, Subsection 4.4 describes its performance when trained using unedited "dirty" training data.

4.1 Top Left Corner

The first approach we use to convert character boxes into vectors is to place the character box in the top left-hand corner of a white image that is the same size for all the characters. We create a white image, then copy the pixels of the character into the top left hand corner of this image. Figure 5 illustrates this. We can then convert the image into a vector as described in Section 3.4.1. Since the image into which we are copying the characters is the same for each character, all the vectors will be of the same size and we can use them to train the SOM and to categorize the characters.

This initial approach has a few limitations. Firstly, for even a slightly different font or different font-size, the SOM might have to be retrained from scratch. This is because a character in a different size or font would show a significant mismatch of the model vectors pixels, even when the fonts are identical. Secondly, when we started to test the system to find unlikely characters we found that this approach gave a considerable advantage to smaller characters over larger ones. Figure 6 shows some results that demonstrate this. The grey areas are parts of



Original Text Imagenary Image

Figure 5: Initial "top left corner" method of converting character boxes to vectors. On the left is a segment of text with a box (dotted line) drawn around a character boundary, on the right is the image that will be used to generate the vector for the character.

the image that were not part of a box. The color of each character is determined by how close the vector representation of the character is to the nearest model vector in the trained SOM, and hence how confident we are that they are in the language the SOM was trained on. Blue characters are close to the SOM model vector, while red characters are distant.

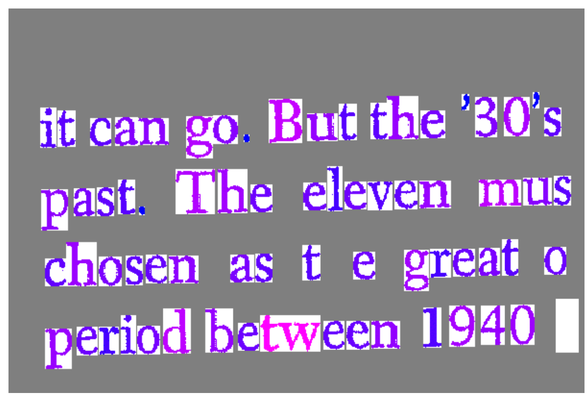


Figure 6: Distance from the nearest matching model vector for the characters in the text presented in Figure 2. Blue characters are close to the language the SOM was trained on, while red characters are classified outside that language. The ideal result would be to color all characters characters, except for the box that contains "tw", since this is not an English character (it is two).

We note that the small characters are closer to the models than the larger ones. This is because of the way we convert the images to vectors; any two small characters share a lot of "white space" and can differ only in the top left hand corner of the image we use to generate the vectors, since we pad the rest of the space with white pixels for both small characters. So for example, comparing a "." to a ";" the difference "," might occupy five pixels,

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

while two capital “M”s might have a larger difference. While both “.” and “;” occur in English, small characters in another character set are more likely to be closer to small Latin characters than two scans of identical characters.

4.2 Scaling Characters

To overcome the problems the “top left corner” technique incurs with different sizes of the same font and of giving an advantage to small characters, we tried scaling the characters when converting them to a vector representation. Again, we create intermediate images of the same size for all the characters, but this time we scale the character to this image. The intermediate image is then converted to a vector as above. Figure 7 illustrates the conversion.

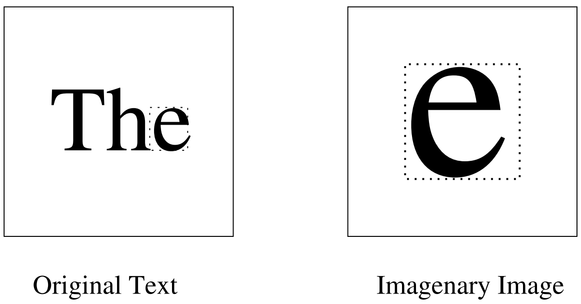


Figure 7: The “scaling” method of converting character boxes to vectors. On the left is a segment of text with a box (dotted line) drawn around a character boundary, on the right is the intermediate image with the character scaled and placed in the top left hand corner.

This method produced much better results. Figure 8 shows a sample of text color-coded in the same way as Figure 6. With the exception of the “m”s the boxes containing a single character in the main font are colored blue, as we would like. The italicized characters and the box containing the “th” bigram are correctly classified as different than the rest of the text. Section 5 describes possible improvements to this approach to deal with misclassified character like the “m”. The following experiment applies this approach to bilingual text.

4.3 Hand-Selected Regions

To test our system on the task of bilingual text segmentation, we attempted to segment text from the Hippocrene Uzbek-English English-Uzbek dictionary (Khakimov, 1994) into Cyrillic and Latin characters. We train the SOM using regions of a dictionary selected by hand to contain English text, and then use the trained SOM to estimate the language of unknown characters. Figure 9 shows a sample of Latin and Cyrillic as they occur in the dictionary, as well as the part selected as Latin text. The

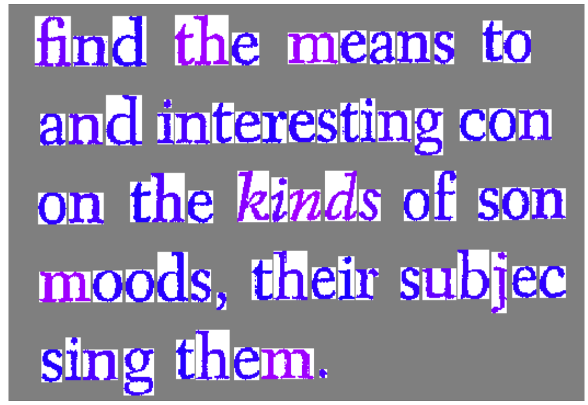
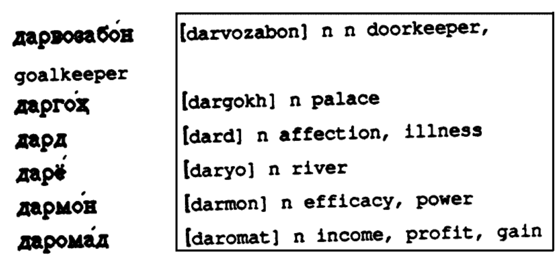


Figure 8: Distance from the nearest matching model vector using the scaling approach on English text. Blue characters are close to the nearest SOM model vector, red characters are far from all SOM model vectors. Note that the italicized characters are more pink than the rest of the text, as expected.

box in the figure is a part selected as English. Note that the word “goalkeeper” was not selected in this process to save time. The word will be classified as English by the trained system. As a rough estimate of how much human time this initial selection entailed, selecting English potions from 40 pages of dictionary text took roughly fifteen minutes.



дарвозабóн [darvozabon] n n doorkeeper,
goalkeeper
даргóҳ [dargokh] n palace
дард [dard] n affection, illness
дарё́ [daryo] n river
дармóн [darmon] n efficacy, power
даромáд [daromat] n income, profit, gain

Figure 9: Part of a page from the Uzbek-English dictionary used for the experiment described in Section 4.3. The part of the image enclosed by the box has been selected as Latin text, and so can be used to train the SOM.

It is important to note that the Cyrillic characters are in a bold font while the Latin characters are not, which may help performance. It is probably not unjustified to assume that this will often be the case, since changing the font also makes it easier for a human to distinguish between languages at a glance and a typesetter might do this to aid the human reader.

Figure 10 shows part of a dictionary page color-coded by the system. Letters closer to blue are more likely to be Latin, while those closer to red are more likely to be Cyrillic. We see that most of the English text is correctly

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

classified – the word “goalkeeper” is most interesting, since it was not used during training, and was correctly classified except for the “pe” bigram for which GOCR did not find the bounding box correctly. We also see that on a word by word basis, the system works pretty well – the Cyrillic words are more red than the Latin words, but there are a number of places where we make mistakes; several Latin characters are very pink, and several Cyrillic characters are very blue.

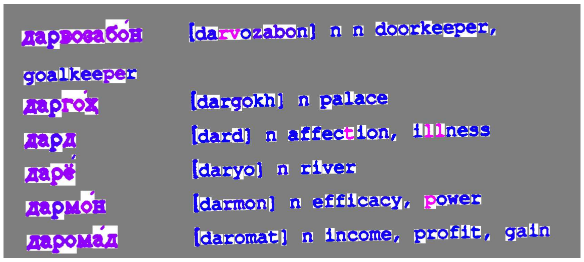


Figure 10: Distance from the nearest matching model vector for dictionary text. Blue characters are classified as Latin, red characters are classified as Cyrillic.

To obtain quantitative results for this experiment, we hand-selected all the English text in a dictionary page, and saved only this in an image. Similarly we placed all the Uzbek text from three dictionary pages into one image. This gave us 576 characters of English and 312 characters of Uzbek. We then used our system to classify all the characters in each image. The results are summarized in Table 1. The system made a correct guess of 98.9% of the time.

Language	Correct	Percentage
English	566 / 576	98.3%
Uzbek	312 / 312	100%
both	878 / 888	98.9%

Table 1: The performance of our system with hand-selected training data on the task of segmenting dictionary text.

4.4 Pages of a Dictionary

The results from the previous experiment are encouraging, but we wanted to investigate whether the human effort of segmenting some pages of text was really necessary. To investigate this we repeated the above experiment, but instead of training the system on hand-selected parts dictionary pages, we trained on entire pages of the dictionary. The idea behind this is to assume that since most of the characters on the page are of one language (English for the portion of the dictionary we used), it may be acceptable to pretend the rest of the characters are noise. Unfortunately, the results this provides are not nearly as good as those obtained in the previous experiments. Figure 11 illustrates the results for the same part of the page shown in Figure 10.

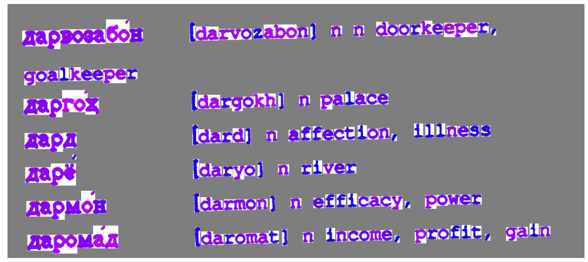


Figure 11: Distance from the nearest matching model vector for dictionary text, when using unedited pages of the dictionary. Blue characters are classified as Latin, red characters are classified as Cyrillic.

Quantitative results were obtained as above, after the system was trained on unsegmented pages of the dictionary. Overall the system makes the correct estimation 72.2% of the time. As a comparison, always choosing English would have given a performance of 64.7%, so this is not an impressive result at all. The details for each language are shown in Table 2.

Language	Correct	Percentage
English	339 / 576	58.9%
Uzbek	302 / 312	96.8%
both	641 / 888	72.2%

Table 2: The performance of our system trained on unedited pages of the dictionary.

5 Conclusions and Future directions

We presented a novel approach for using self organizing maps to segment bilingual text into its component languages, with applications to OCR. We tried this approach on pages from an Uzbek-English dictionary, and found that we were able to identify the correct language for 100% of the Uzbek characters and 98.3% of the English characters. With this success in mind, there are a number of limitations to the approach presented here, and we discuss these in Section 5.1. Section 5.2 concludes the paper with a discussion of possible future work.

5.1 Problems with this Approach

There are a number of limitations to the method presented in this paper. In order to get the results we obtained, we needed to manually select English text on which the SOM could be trained. While this did not take a lot of time, it is a cost that needs to be incurred for each new book on which OCR is to be performed. The method we describe

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College

is also very computationally intensive and requires a lot of space; training the system took several hours on the available hardware and the training data occupied about 115 MB of disk space (compared to less than 2.5 MB for the images). We also do not use character position information in any way, so we may guess that a character that is part of a long word is of a different language than the rest of the word. Making the assumption that there is one language per word would probably considerably improve our results.

5.2 Future Work

Investigating the limitations mentioned above would be the first line of future work. One part of this would be to investigate how much training data is really necessary. Perhaps it is enough for a human to segment one page of training data instead of forty. If a few pages of training data are sufficient this would also alleviate a lot of the space requirements we currently have. We also do not investigate the possibility of training the SOM for a smaller number of iterations. It would be interesting to find out how this affects performance.

Extending the method to use position information would also be a line of future work. The one language per word assumption would not be difficult to implement and may improve result – especially when we only have scarce training data. Further position information could also be learned on the fly. For example, in our dictionary all the characters in the left two-thirds of the page were English. Another easy extension of this work would be to adapt it to deal with more than two languages.

A more difficult future work (suggested by one of the anonymous reviewers) would be to combine our approach with a language model approach. For example, if we perform OCR on each word in the image assuming it is in one language, the resulting text would match the language model best if we guessed the correct language. We could repeat this for each possible language before making a final decision. Combining this approach with ours would be helpful, since it could be used to provide training data with no human involvement, while our approach would deal with words that do not match the language model, such as the Latinization of the Cyrillic words in our dictionary.

References

Benjamin P. Berman and Richard J. Fateman. 1994. Optical character recognition for typeset mathematics. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 348–353. ACM Press.

Kamran Khakimov. 1994. Hippocrene Books.

Kohonen, Hynninen, Kangras, and Laaksonen. 1995. The self organizing map program package.

Teuvo Kohonen. 1990. The self-organizing map. *Proceedings of the IEEE*, 78(9).

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 109–115
Computer Science Department, Swarthmore College