TECHNICAL UNIVERSITY OF CRETE

Department of Electronics & Computer Engineering

AUTOMATIC DERIVATION OF ONTOLOGIES FROM TEXT

a thesis by

Iosif Elias

Supervisory Committee

Potamianos Alexandros (Supervisor)
Digalakis Vassilios
Petrakis Euripides

2004

to my father

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express sincere appreciation to my advisor Mr. Alexandros Potamianos for his invaluable guidance.

I am also very grateful to the members of the supervisory committee.

I would like also to thank my friends and especially my family for their pure love.

Special thanks to Eleni.

Introduction

Many applications dealing with textual information require classification of words into semantic classes (concepts). A natural language understanding module, which is embedded in advanced computer dialogue agents, requires knowledge of semantic classes. For example, during a human-machine dialogue interaction, the human's audio stream is passed on to a speech recognizer. The audio stream is transcribed to a string of text by the automatic speech recognizer. Then this string is passed on to a computer dialogue agent, which is responsible for the extraction of the information contained in the utterance. The subject of our work is the extraction of semantic concepts from text using an automatic procedure.

Manually constructing semantic classes is a tedious and time-consuming task. Moreover, it requires expert knowledge. An inexperienced developer may omit important components for each semantic class. The lexical information is not specific to any domain. Rather, the entries attempt to capture what applies to the language at large, or represent specialized senses in a disjunctive manner. Note that semantic lexical knowledge is most sensitive to domain changes. Unlike syntactic constraints, semantic features tend to change as the word is used in different ways for different domains. All these reasons raise the need for an automatic procedure.

Our effort, for meaning identification, relied on the hypothesis that words that appear in similar lexical contents are semantically similar. The lexical environment was the only text's feature we explored. In some sense, our approach tried to discover the meaning, which is hidden in a flexible use of natural language, without the use of strict syntactic rules.

The first step in designing an understanding module for a domain is to obtain a corpus of transcribed utterances. We have mainly worked on a single domain, ATIS, which is an air reservation system. The used unannotated corpus is composed of transcribed human requests taken from a human-machine spoken dialogue interaction over the telephone network. The used ATIS corpus is a relatively small homogeneous corpus. The term "homogeneous" means that in the transcribed requests there is no expressional complexity and the used vocabulary is limited to the purpose of the provided service.

The proposed iterative procedure consists of three main steps to auto-inducing classes. First, a "lexical phraser" groups words in a single lexical unit. Second, a "semantic generalizer" maps words (and concepts) to concepts. Lastly, a "corpus parser" re-parses the corpus using the classes generated by the semantic generalizer. The lexical phraser uses the notion of

mutual information. The semantic generalizer uses four different metrics for semantic distance.

The phenomenon of natural language

The concept of culture covers all those skills and ways of life that are transmitted by interpersonal communication and tradition. It includes many different aspects of the life of people, their knowledge and language, their religion, beliefs and laws, their customs, rituals and arts, their tools, foods, and other means getting a living.
The changes in human life in the last 10 000 years have certainly been tremendous, both in quantity and quality. People who had achieved the power of communication by symbolic means can reasonably ascribe this explosion to the exploitation of the new favorable climatic conditions. Words reflect the ability to discuss the properties of the objects and events that are not present, and so to extend the range of actions to meet the future.
The capacity to embody information in records outside the body provides a proof about the rate's acceleration of evolutionary change. With language and writing we have an extra-corporeal information store. Written records are, like all other tools, an artificial substitute for a function that was previously performed in the body. As has happened with other tools the invention of the capacity to make extra-somatic memory records in written codes has led eventually to an understanding of the code in the body.
The inventions of language and writing were the essential tools, if we may call them, with which all the others were produced, and led to Bronze, Iron and Machine ages. Yet the acquisition of a store of information has not been by any means uninterrupted. Periods of advance, such as those of the Chinese, Sumerians, Minoans and Greeks have been followed by stagnation in the increase of knowledge, or its loss, as in Barbarian Europe after the fall of the Romans [21].

The study of human language is particularly a study of human knowledge, which may be considered as a 'cognitive system'. Cognitive systems are based on the interaction of experience and the organism's ability of constructing and dealing with it, combined with the determinants of maturation and cognitive growth. Experience and knowledge are separated by a gap which expresses the essence of the Plato's intellectually exciting problem: "How we can know so much given that we have such limited evidence?" [22].

Researchers in the field of neuropsychology note that the natural language is an uncompleted attempt of meaning signaling and remains uncompleted forever! A lot of times the matter that is being said is totally different from the matter that is not being said. In other words, natural language often speaks using the silence, which is transformed into allusion during the speech procedure [23].
The phrase 'know so much' appeared in the previous question lies in the fact that the human in every aspect of his activity uses a knowledge background well formed through history, ignoring how this knowledge was gained. On

the other hand his lifetime is too short to be able to construct the whole evidence's schema. Plato's problem seems on the surface to be contradictory, but suggests the following challenging approach. The answer is to identify the principles, often hidden in a noisy environment, and extract some sense of meaningful patterns.

An empiricist corpus-based approach is found in the work of American structuralists. Generally it was a noticeable attempt of exploring the methods that can model the language's structure automatically, without computer implementation. The empiricist approach to natural language processing denotes that the study of language's structure is accomplishable by defining a language model and inducing the several parameters by using statistics to a large collection of texts.

The aim of the linguistic science is to discover and describe the inner structure and the functionality of the linguistic phenomenon, as is expressed in conversations, writing and other media. In order to achieve this aim the linguistic community has proposed several sets of rules, which sketch the framework of linguistic expressions. This approach became rigorous as linguists tried to explore more sophisticated grammars. The linguistic community ignored an important fact. People tend to adjust the grammatical rules to their communicative needs. Simply, people choose the easiest way in order to communicate.

It is more reasonable to study further the creativity of language use, rather than focusing only in strict grammatical rules. Adopting this approach we can ask," What are the most common patterns that occur in language use? ". Statistics can provide the useful tools for this aim. British linguist coined: "You should know a word by the company it keeps". Probability theory is the scientific field that can implement this statement. Statistical language processing suggests the use of corpora with regard to the textual context, in order to situate language in a real world context.

Zipf's law gives an example of a surprising and interesting natural phenomenon. Harvard linguistics professor George Kingsley Zipf observed that the frequency of the *k-th* most common word in a text is roughly proportional to *1/k*. He justified his observations in a book titled "Human behavior and the principle of least effort" published in 1949. Principle of least effort states that people will act so as to minimize their probable average rate of (present or future) work. This reminds us the fact that people try to find the easiest way to satisfy their communicative needs. While Zipf's rationale has largely been discredited, the principle still holds, and others have afforded it a sounder mathematical basis [2].

1 Why we need statistics?

This chapter introduces some basic material on probability and statistics required for the understanding of our work. These definitions play central role in corpora processing since through their interpretation and implementation, epitomize the meaningful data. This ability can be compared with a dive in a deep and foggy lexical environment that results in discovery of the treasure of meaning. Corpora can be large collections of texts written by any editor or transcribed human-human, human –computer dialogues.

Obviously the desired meaning is hidden inside of each word. Also each word can be viewed as a distinct event which whenever it occurs, it transmits, in a sense, some general information. The notion of "word" in linguistics is denoted by the term "lexeme". Thus a lexeme is the minimal unit of language, which has a semantic interpretation and embodies a distinct cultural concept. Furthermore the extracted knowledge can be broader if we study a larger lexical field that consists of more than one lexeme. A single lexeme can be a "lexical unit", but in writing it is very common, more than one lexemes to behave as a lexical unit. So, is there any difference between lexeme and lexical unit or not? Cruse distinguishes lexemes from lexical units. The former are the items listed in the lexicon, or ideal dictionary of a language. A lexeme corresponds to a particular word or word form, and can be associated with indefinitely many senses. The latter are form-meaning complexes with stable and discrete semantic properties, and the meaning component is called a sense, corresponding to the intuitive notion of sense. So bank is a lexeme, while bank-financial institution and bank-edge of a river are lexical units. We will see that for our work this approach does not propose clear criteria for establishing Cruse's distinction. We underline only the fact that it is myopic policy to isolate word from its lexical environment. It is reasonable to claim that a word preserves a kind of relationship with its neighboring words, in some way. Still the concept of "neighbor" remains abstract. We can guess correctly that it is worthless to treat a whole sentence as a single lexical unit that expresses a very specific meaning. Therefore, few consecutive words may form a lexical unit with a particular meaning. In such case we are sure that each word contains an amount of information about the other words of the lexical unit. This means that the probability of one word's appearance is not independent. In the opposite case the consecutive words would not form a lexical unit with a comprehensive meaning. These considerations trigger off the thought to find an alternative, single lexeme in order to express the same meaning that the whole lexical unit does. But still we need a metric to estimate the precision of this idea. Additionally we cannot ignore the fact that a certain matter can be said or written using more than one lexical unit. Practically this is interpreted to the fact that people use many different lexical units to express approximately the same concept. So it is important to explore

the semantic relationship between the lexical units of a corpus regardless of their place in the text.

1.1 Counting Words

Probability theory deals with predicting how likely it is that an event will take place. There are two interesting views of probabilities:
The objectivist view states that probabilities are real aspects of the world that can be measured by relative frequencies of outcomes of experiments. In contrast, according to the subjectivist view, probabilities are descriptions of an observer's degree of belief or uncertainty rather than having any external significance. These contrasting views are also referred to as Frequentist vs. Byesian. Both views are relevant for linguistics; yet, the laws of probability theory remain the same under both interpretations.
Probabilities are based on counting things. But, in our case what these things are? Statistical language processing requires computation of word probabilities. These probabilities are computed by counting words or lexical units in a training corpus. The classical definition of probability, as given by Pascal is: "The probability of an event x is computed as the relative frequency with which x occurs in a sequence of n identical experiments ". So, the probability of a word w is the relative frequency with which x occurs in the corpus.

$$p(w) = \frac{occurrences\ of\ word\ w}{number\ of\ words} \qquad (1.1.a)$$

The role of punctuation marks

Suppose that we have to count the words of the following sentence from Shakespeare's Hamlet and compute the probability of the word "God".

"Oh God, I could be bounded in a nutshell and count myself a king of infinite space."

If we count punctuation marks as words, the sentence has totally 19 words and the *p("God") = 1/19*. If we do not count punctuation marks as words, the sentence has totally 17 words and the *P("God") = 1/17*.
Whether we count punctuation marks as words depends on the task. Tasks such as grammar checking or author-identification must treat punctuation marks as words because in these cases the location of the punctuation is important. Usually corpora of spoken language do not have punctuation marks. In our work we used corpora of spoken language without punctuation marks.

1.2 N-grams

In the introduction of this chapter, we mentioned the significance of considering consecutive words through a probabilistic model. This model proposes the assignment of probabilities to strings of words. Based on this method we can easily compute the probability of an entire sentence or predict the next word in a sequence.
The simplest probabilistic version of this model allows every word have the same probability of following every other word. A more robust model let every word follow every other word, with the appearance of the following word to be depended on its normal frequency of occurrence. We still consider the individual relative frequency of each word. The following example based on real data can verify the precision of this simplistic approach.
Brown corpus has 1 000 000 words. The word "the" occurs 69 971 times in the corpus and the word "rabbit" occurs 11 times. Thus the probabilities are 0.07 and 0.00001 for the words "the" and "rabbit", respectively.
Suppose that we have just read this part of a sentence:
"Just then, the white ".
Furthermore suppose that we are curious about what the next word will be. If we use the simple model, we will conclude that the word "the" is the most possible word to follow "white". But this seems totally false because there is no meaning in the sentence "Just then, the white the". Doubtless the sentence "Just then, the white rabbit" sounds more reasonable.
This example shows that the computation of the probability of word sequences must use the conditional probability of a word given the previous word. Particularly that means that the probability of a word given the previous one is higher than its probability otherwise.

1.2.1 Conditional probability and independence

The notion of conditional probability can be considered as a kind of updated probability of an event given some knowledge. The probability of an event before gaining additional knowledge is referred to as the prior probability of the event. The new probability of the event estimated using the additional knowledge is called posterior probability of the event. The event of interest is formed by the occurrences of a word in the corpus. Using symbol Ω we denote the sample space, which is discrete, having finite number of elements. The sample space, which corresponds to a corpus, includes all the occurrences of each distinct word of the corpus. That is, the sample space Ω includes all the events of the corpus.
For instance, assume a small corpus: "A tiny corpus tiny".
Ω = {occurrence of the word "A",
occurrence of the word "tiny",
occurrence of the word "corpus",
occurrence of the word "tiny" for second time}

The event of the occurrence of "tiny" in the corpus is denoted as tiny and is:

tiny = {occurrence of the word "tiny",
occurrence of the word "tiny" for second time}

We define the probability of the occurrence of the word "tiny" according to (1.1.a) as:

$$p(tiny) = \frac{|\,tiny\,|}{|\,\Omega|},$$

where $|\,tiny\,|$ is the number of elements in the set tiny and $|\,\Omega\,|$ is the number of elements in the probability space Ω. Thus, $p(\Omega) = 1$.
So the probability of the event tiny, $p(tiny)$, is:

$$P(tiny) = \frac{|\,tiny\,|}{|\,\Omega|} = 2/4$$

For the general case:

$$P(event) = \frac{|\,event\,|}{|\,\Omega|} \qquad (1.1.b)$$

Each event we consider is a subset of Ω.

The conditional probability of a word w_2 assuming that word w_1 has occurred ($p(w_1) > 0$), denoted $p(w_2 \,|\, w_1)$, equals

$$p(w_2 \,|\, w_1) = \frac{p(w_2 \cap w_1)}{p(w_1)} \qquad (1.2)$$

Multiplying through, this becomes

$$p(w_2 \,|\, w_1)p(w_1) = p(w_2 \cap w_1) \qquad (1.3)$$

Rearranging (1.2) gives

$$p(w_1 \,|\, w_2) = \frac{p(w_1 \cap w_2)}{p(w_2)} \qquad (1.4)$$

We can do the conditionalization either way because set intersection is symmetric, $w_2 \cap w_1 = w_1 \cap w_2$.

Solving (1.4) for $p(w_2 \cap w_1) = p(w_1 \cap w_2)$ and plugging in to (1.2) gives

$$p(w_2 \mid w_1) = \frac{p(w_2)\, p(w_1 \mid w_2)}{p(w_1)} \qquad (1.5)$$

Two words w_1, w_2 are independent each other if $p(w_2 \cap w_1) = p(w_2)p(w_1)$.
If $p(w_1)$ not equals zero, is equivalent to write that $p(w_2) = p(w_2 \mid w_1)$ because is known that w_1 does not affect the probability of w_2.

1.2.2 Unsmoothed N-grams

Conditional probability and independence can be the basis for computing the probability of a string of words.
A string of words can be represented as $w_1,w_2,\ldots,w_{n-1},w_n$ *or* w_1^n.
Assuming the occurrence of each word in the corpus as an independent occurrence, we can write the probability of a string of words as follows:

$$p(w_1,w_2,\ldots,w_{n-1},w_n) \quad or \quad p(w_1^n) \qquad (1.6)$$

Using the chain rule of probability we represent $p(w_1^n)$ as:

$$p(w_1^n) = p(w_1)\, p(w_1 \mid w_2)\, p(w_3 \mid w_1^2) \ldots p(w_n \mid w_1^{n-1})$$
$$= \prod_{k=1}^{n} p(w_k \mid w_1^{k-1}) \qquad (1.7)$$

Since there is not any easy way for computing the probability of a word given all the previous words, an alternative solution for this task is to find a satisfactory approximation. The bigram model proposed for solving this difficulty, assumes that the probability of a word depends only on the previous word. In other words, $p(w_n \mid w_1^{n-1})$ is approximated by the conditional probability of the word that preceded $p(w_n \mid w_{n-1})$. This approximation is referred as a Markov assumption. Markov models are probabilistic models, which predict a future event without knowing a lot of things about the past. In the case of bigram (first order Markov model) models they need to know only the preceding word.
It is obvious that the trigram (second order Markov model) model looks two words into the past. Generalizing bigrams and trigrams, N-grams are resulted by which the probability of a word given all the previous words can be approximated by the probability given only the previous N words.

$$p(w_n \mid w_1^{n-1}) \approx p(w_n \mid w_{n-N+1}^{n-1}) \qquad (1.8)$$

For a bigram grammar, $p(w_1^n)$ can be found by substituting (1.8) into (1.7):

$$p(w_1^n) \approx \prod_{k=1}^{n} p(w_k \mid w_{k-1}) \qquad (1.9)$$

A practical problem: numerical underflow

Since probabilities are less than 1, the product of multiplication of many probabilities is too small. To avoid computational problems it is better to take the logarithm of each probability.
In practice many programs for language modeling perform all calculations in the log space, like the CMU Statistical Toolkit [16].

There is one more detail to be mentioned. Trying to compute the probability of a sentence, i.e. $w_1\, w_2\, w_3\, w_4\, w_5$, using the bigram model according to (1.9), is clear that we have to multiply the bigam probabilities. But there is no past for the first word, w_1, of the sentence. In order to face this trouble we can adopt a special pseudo-word, <*s*>, meaning "start of sentence":

$P(w_1\, w_2\, w_3\, w_4\, w_5) = P(w_1 \mid \langle s \rangle)\, P(w_2 \mid w_1)\, P(w_3 \mid w_2)\, P(w_4 \mid w_3)\, P(w_5 \mid w_4)$

In the same case when we use trigram probabilities two pseudo-words, <*s1*> and <*s2*>, must be used.

Recalling (1.1.b), the way that N-grams can be trained is simply by counting and normalizing the event of interest. In other words, we use some training corpus, and find the count of a particular N-gram and divide the count by the sum of all the N-grams that have the same first word:

$$p(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}\, w_n)}{\sum_w C(w_{n-N+1}^{n-1}\, w)} \qquad (1.10)$$

Since the sum of all bigram counts that have, as first word the word w_{n-1}, is equal to the unigram count for the word w_{n-1}, equation (1.10) can be simplified as follows:

$$p(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}\, w_n)}{C(w_{n-N+1}^{n-1})} \qquad (1.11)$$

Function *C(.)*, presented in the two last equations, means "take the count of ".

The ratio, that results from equations (1.10) and (1.11), is called relative frequency.

The whole picture about N-grams probabilities

Any particular training corpus has limited sentences, so some N-grams are possible not to be presented in the corpus. The consequence is that the N-gram model assigns zero probability to these N-grams. Also, using only relative frequencies to estimate N-grams probabilities might produce poor estimates when the counts are too small. This major problem raises the need to find a way of reevaluating zero probability and low probability N-grams and assigning them non-zero values.
This way is called smoothing.

Dealing with corpora: a sensitive issue

An N-gram model is trained on a corpus in order to produce the N-grams probabilities. An important issue is that the training corpus must be carefully designed in such way that preserving a representative sample of the task of interest. If the training corpus is specific task oriented may not generalize properly to new sentences. On the other hand, if the training corpus is too general the probabilities may not capture sufficiently the nature of the domain. Additionally, if we intend to construct our own training corpus using smaller, completed corpora, the resulting corpus needs to be balanced, which means to be proportional to some predefined criterion of importance.
Furthermore suppose we are given a representative and balanced corpus in order to train and test a language model. The proper way to treat the given data is to divide the data into a training set and a test set. The training set is the field where the statistical parameters would be trained and the test set is about to be used for computing the probabilities.

1.3 Smoothing

It was mentioned that a problem of underestimating the probability of N-grams is possible to occur when we use N-gram model, caused to the finite nature of the training corpus. Our goal is to assign a non-zero probability to zero probability N-grams.

1.3.1 Add-One Smoothing

This algorithm suggests to take the bigram counts and before normalizing them to probabilities, to add one to all the counts. This algorithm is very simple and in practice does not perform well. However is an introduction to the concept of smoothing that is implemented much better by other algorithms.

Considering the unsmoothed maximum likelihood estimate of the unigram probability:

$$p(w_x) = \frac{c(w_x)}{\sum_i c(w_i)} = \frac{c(w_x)}{N}$$

, where N is the total number of word tokens in the training corpus.

The basic idea of smoothing relies on the c's adjustment. The adjusted count for add-one smoothing is defined by adding one to the count c and then multiplying by the factor $N/(N+V)$, which is a normalization factor. Then the adjusted count is:

$$c_i^* = \frac{(c_i + 1)\, N}{(N+V)} \qquad (1.12)$$

, where V is the vocabulary size of the training corpus.

Equation (1.12) can be turned into probabilities p_i^* by dividing by the total number of word tokens.

$$p_i^* = \frac{(c_i + 1)}{(N+V)}$$

Applying this useful baseline to the equation (1.11), the add-one-smoothed probability for a bigram is defined as:

$$p^*(w_n \mid w_{n-1}) = \frac{(c(w_{n-1}\, w_n) + 1)}{(c(w_{n-1}) + V)} \qquad (1.13)$$

An alternative view of smoothing: actually a smoothing algorithm discounts some non-zero counts. This is a way to find the probability mass, which will be assigned to the zero counts. An alternative way to refer to lowered counts c^* is to define a discount ratio d_c:

$$d_c = \frac{c^*}{c} \qquad (1.14)$$

The choice of value "one" which is added to the each count c is arbitrary. This affects the probability mass that is moved near the zero value. A solution to this problem is the choice of smaller values regarding the situation.

Generally add-one smoothing produces poor estimates. It has been showed by Gale and Church that variances of the counts produced by this smoothing algorithm are worse than those produced by unsmoothed Maximum Likelihood Estimation method.

1.3.2 Witten - Bell Discounting

This method of discounting is much better than Add-One smoothing and is commonly used in language modeling toolkits such as CMU Statistical Toolkit.

The algorithm is referred as Method C, a method initially introduced by Alistair Moffat. In [18], Witten and Bell surveyed and compared several approaches to the zero-frequency problem that have been used in text compression systems. Witten and Bell described the zero-frequency problem in the case of adaptive word coding assuming a coding scheme in which the encoder reads the next word of text, searches for it in a list and transmits an index extracted from the list in place of the word. If the next word is not appeared in the list, a special code, called escape code, must be transmitted followed by the unknown word. This new word is added to the encoder and decoder's lists in case it appears again. According to this method each word is assigned an associated frequency. The computing of probability of the escape character, by estimating the likelihood of a novel word occurring, can solve the zero-frequency problem.

Similarly, a novel N-gram could then be assigned the probability of seeing it for the first time. The basic idea behind this conception is to "use the count of things we have seen once to help estimate the count of things we have never seen "

We can compute the probability of seeing a novel N-gram by counting the number of times we saw N-grams for the first time in the training corpus. The count of the first-time seen N-grams is simply the number of N-gram types we have already seen.
Hence we can estimate the total probability mass of all the zero N-grams by dividing the number of N –gram types we have seen with the sum of number of tokens and the number of N –gram types we have seen:

$$\sum_{i:ci=0} p_i^* = \frac{T}{N+T} \qquad (1.15)$$

, where T is the N-gram types we have already seen and N is the number of tokens.

Probability given by (1.15) is the total probability of unseen N-grams. This "amount of probability" needs to be divided in order to assign a part of it to each zero N-gram. A simple compromise is to divide equally. Letter *Z* denotes the total number of N-grams with count zero. So the equal share of the probability mass is:

$$p_i^* = \frac{T}{Z(N+T)} \qquad (1.16)$$

The probability of all the seen N-grams is given by the equation:

$$p_i^* = \frac{c_i}{N+T}, \quad \text{if } c_i>0 \qquad (1.17)$$

Extending the Witten-Bell discounting to bigrams, the type-counts are conditioned on some history. The probability of seeing for first time a bigram $w_{n-1}\ w_n$ is equivalent to the probability of seeing a new bigram starting with the word $w_{n-1.}$
According to the equation (1.15) the probability of a bigram $w_x\ w_i$ we have not seen is:

$$\sum_{i:c(w_x w_i)=0} p^*(w_i \mid w_x) = \frac{T(w_x)}{N(w_x)+T(w_x)} \qquad (1.18)$$

,where $T(w_x)$ is the number of bigram types on the previous word w_x we have already seen and $N(w_x)$ is the number of bigram tokens on the previous word $w_{x.}$

Distributing the probability mass of the equation (1.18) among all the unseen bigrams, we get:

$$p^*(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})\ (N+T(w_{i-1}))} \quad \text{if } c(w_{i-1}w_i)=0 \qquad (1.19)$$

,where $Z(w_{i-1})$ is the total number of bigrams with w_{i-1} as the first word, that have count zero.

For the non-zero bigrams, we parameterize *T* on the history:

$$p^*(w_i \mid w_x) = \frac{c(w_x w_i)}{c(w_x)+T(w_x)} \quad \text{if } c(w_{i-1}w_i)>0 \qquad (1.20)$$

1.3.3 Good-Turing Discounting

A slightly more complex algorithm than the Witten-Bell algorithm is the Good-Turing discounting. The central idea underlying Good-Turing smoothing is the re-estimation of the probabilistic mass, which is assigned to N-grams with zero or low counts, by looking at the number of N-grams with higher counts. This algorithm is suitable for a corpus with large vocabulary that provides large number of observations.
The first step is to examine N_c, the frequencies of different frequencies (also referred as count-counts). In other words, we calculate the number of N-grams that occur c times. The number of N-grams that occur c times is called the frequency of frequency c.

A "non-conditional" discounting

In the previous method we conditioned the smoothed bigram probabilities on the previous word. It is also possible to treat the bigrams as a unit by discounting, not the conditional probability $p(w_i | w_x)$, but the joint probability $p(w_x w_i)$. This different kind of discounting is often used for the Good-Turing discounting.

Under of this idea to smoothing the joint probability of bigrams, N_c is the number of bigrams b of count c.
A smoothed count c^* is give by the following equation:

$$c^* = \frac{(c+1)\, N_{c+1}}{N_c} \qquad (1.21)$$

Applying this equation in the case of bigrams that never occurred we have:

$$c^*_0 = \frac{(0+1)\, N_1}{N_0}$$

Equation (1.21) highlights the key concept of the Witten-Bell discounting: use the count of things we have seen once (N_1) to estimate the count of things we have never seen.

A reasonable question

One may ask: "Which is the number of unseen bigrams?"
Since the total number of bigrams is V^2, the number of bigrams we have not seen, N_0, is V^2 minus all the bigrams we have seen.

It has been proved that the Good-Turing smoothing makes the assumption that the distribution of each bigram is binomial.

In real applications the discounted estimate c^* is not used for all counts c. For some threshold k, the correct version of equation (1.21) is:

$$c^* = \frac{(c+1)\,(N_{c+1} / N_c) - (\,c(k+1)\,(N_{k+1}) / N_1\,)}{1 - ((k+1)\,(N_{k+1}) / N_1)} \qquad (1.22)$$

1.4 Backoff

So far the algorithms we have presented have all made use of the frequency of an N-gram and have tried to compute the best estimate of its probability. In general N-grams that never appeared or appeared only few times, were given the same estimate. A reasonable extension of the previous methods (smoothing) is to try to build better estimates by looking at the frequency of the (N-1)-grams found in the N-gram.

If (N-1)-grams, found in the N-gram, are appeared rarely, then a low estimate is given to the N-gram. Otherwise, N-grams with (N-1)-grams of moderate frequency are given a higher probability estimate. This issue grounded in a more general discussion deals with combining multiple probability estimates making use of different models. That is, if there are no examples of a particular trigram, let's say $w_{n-2}w_{n-1}w_n$, the computation of $p(w_n \mid w_{n-1}w_{n-2})$ can be achieved through the use of the bigram probability $p(w_n \mid w_{n-1})$. In the same manner, if we have no examples of $w_{n-1}w_n$ in order to compute $p(w_n \mid w_{n-1})$, we can use the unigram probability $p(w_n)$.

In the backoff model, as described above, an N-gram model is built based on a (N-1)-gram model. We only look to a lower-order N-gram if we have no examples of a higher-order N-gram.
So the backoff model for the trigram $w_{i-2}w_{i-1}w_i$ is:

case 1: $c(w_{i-2}w_{i-1}w_i) > 0$

$$p\,(wi \mid wi_{-2}w_{i-1}) = p\,(wi \mid wi_{-2}w_{i-1})$$

case 2: $c(w_{i-2}w_{i-1}w_i) = 0$ *and* $c(w_{i-1}w_i) > 0$ (1.23)

$$p\,(wi \mid wi_{-2}w_{i-1}) = a_1\,p\,(wi \mid wi_{-1})$$

case 3: *otherwise*

$$p\,(wi \mid wi_{-2}w_{i-1}) = a_2\,p\,(w_i)$$

The a values are weighting factors, which ensure that the result of the equation (1.23) is a true probability.

For the general case the form of backoff is:

$$\hat{p}(w_n \mid w_{n-N+1}^{n-1}) = \tilde{p}(w_n \mid w_{n-N+1}^{n-1}) + \theta(p(w_n \mid w_{n-N+1}^{n-1})).a\,\hat{p}(w_n \mid w_{n-N+2}^{n-1}) \qquad (1.24)$$

The θ notation indicates a binary function that selects a lower-order model only if the higher-order model produces a zero probability.

$$If\ x = 0\ then\ \theta(x) = 1,\ else\ \theta(x) = 0 \qquad (1.25)$$

Each $p(.)$ is a Maximum Likelihood Estimation.

1.5 A clever combination: Backoff and Discounting

Previously, we have used discounting methods to find how much probability mass to assign to unseen events, assuming that they were equally probable. Combining discounting with backoff we can distribute this probability more cleverly. It is important to understand the role of a values in equation (1.24). The a values are weighting factors, which ensure that the result of the equation (1.24) is a true probability.

Consider the following example, which shows how backoff can lead to probability greater than 1:
Using relative frequencies, $\sum_{i,j} p(w_n \mid w_i w_j) = 1$, which means that the probability of a word w_n over all N-gram contexts equals to 1. If we use backoff in this case, adopting a lower order model, the probability of w_n will be greater than 1. So, discounting must be applied to backoff model.

Thus, the correct form of equation (1.24) is:

$$\hat{p}(w_n \mid w_{n-N+1}^{n-1}) = \tilde{p}(w_n \mid w_{n-N+1}^{n-1}) + \theta(p(w_n \mid w_{n-N+1}^{n-1})).a(w_{n-N+1}^{n-1})\,\hat{p}(w_n \mid w_{n-N+2}^{n-1}) \qquad (1.26)$$

$\tilde{p}(.)$ stands for the discounted MLE probabilities:

$$\tilde{p}(w_n \mid w_{n-N+1}^{n-1}) = \frac{c^*(w_{n-N+1}^{n})}{c(w_1^{n-N+1})} \qquad (1.27)$$

Function *a* represents the amount of probability mass, which must be distributed from an N-gram to an (N-1)-gram:

$$a(w_n \mid w_{n-N+1}^{n-1}) = \frac{1 - \sum_{\beta} \tilde{p}(w_n \mid w_{n-N+1}^{n-1})}{1 - \sum_{\beta} \tilde{p}(w_n \mid w_{n-N+2}^{n-1})} \qquad (1.28)$$

, where β denotes $w_n : c(w_{n-N+1}^{n-1}) > 0$.

For example, a trigram form of the backoff model is:

case 1: $c(w_{i-2}w_{i-1}w_i) > 0$

$$\hat{p}(w_i \mid w_{i-2}\, w_{i-1}) = \tilde{p}(w_i \mid w_{i-2}\, w_{i-1})$$

case 2: $c(w_{i-2}w_{i-1}w_i) = 0$ *and* $c(w_{i-1}w_i) > 0$ (1.29)

$$\hat{p}(w_i \mid w_{i-2}\, w_{i-1}) = a(w_{n-2}^{n-1})\, \tilde{p}(w_i \mid w_{i-1})$$

case 3: *otherwise*

$$\hat{p}(w_i \mid w_{i-2}\, w_{i-1}) = a(w_{n-1})\, \tilde{p}(w_i)$$

A last comment

Probability estimates can change suddenly in adding more data when the backoff algorithm selects a different order of N-gram model on which to base the estimate, but in general backoff works well in practice.

1.6 Summary

In this chapter we introduced some basic material on probability and statistics. Also we explained the term "corpus", the notion of "lexical unit" and the reason for which probability theory pays a central role in corpora processing. Next we saw the use of N-grams in natural language modeling. Through the study of smoothing we learned how to assign a non-zero probability to zero probability N-grams. The backoff model it was a case where an N-gram model was built based on (N-1)-gram model, when there were no examples of a particular N-gram. Lastly, the combination of smoothing and backoff showed how to distribute cleverly the probability mass to unseen events. Generally this chapter is very important for the understanding of the next chapters where some concepts from the information theory are presented. Moreover, the used CMU toolkit put into

practice many issues that were referred in chapter 1, such as the Witten-Bell discounting.

2 Elements of information theory

In the previous chapter we saw that it is possible to use several N-grams in order to build a language model. Each N-gram embodies a particular amount of information, which can be used for several tasks. The concept of information is invaluable for natural language processing since is the fundamental element for the evaluation of language model. Also it can discover important properties of a given text that can lead us to useful observations. Recall from chapter 1 that in some cases we need to find these co-occurring lexical units that behave as a single unit and treat them in the next procedures as a single unit. This can be achieved by studying the relation between the co-occurring words from the information view of point.

2.1 Entropy

Natural language is a kind of information source. A script of natural language can be viewed as a stochastic process, which consists of a sequence of words. The distribution of the next word is highly dependent o the previous words. There is a great deal of variability and uncertainty in natural language. Entropy is a measure of information. Alternatively entropy is can be considered as a measure of "uncertainty" of a random variable [17].

For example, suppose that we have a set of possible events in a given text that is the set of the occurrences of some words. Their probabilities of occurrence are: $p_1,p_2,\ldots,p_n$. Concerning which event will occur, entropy $H(p_1,p_2,\ldots,p_n)$, is the measure of how much "choice" is involved in the selection of the event.
This measure it is reasonable to have the following properties:

1. H is continuous in the p_i.

2. If all the probabilities of occurrence are equal, $p_i = 1/n$, then H is a monotonic increasing function of n. If there are more possible events with equal p_i, there is more "choice" or "uncertainty".

3. If a choice is divided into choices, the original H is the weighted sum of the individual values of H. This property is illustrated in the following figure:

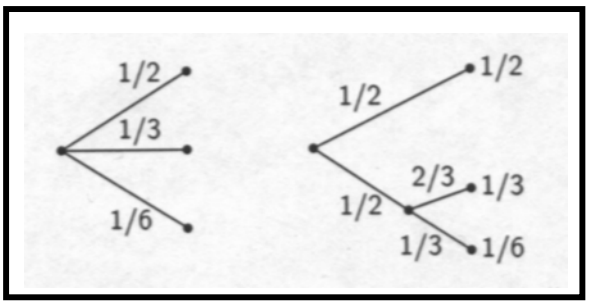


Figure 2.1: Breaking down a choice into two choices

In this case $H(1/2,1/3,1/6) = H(1/2, 1/2) + 1/2H(2/3,1/3)$

The form of H, which satisfies these three properties, is:

$$H = - k \sum p_i \log p_i \qquad (2.1)$$

, where k is a positive constant.

The entropy in the case of two possible events with probabilities p_1 and $q_1=1-p_1$ is:

$$H = -(p_1 \log p_1 + q_1 \log q_1)$$

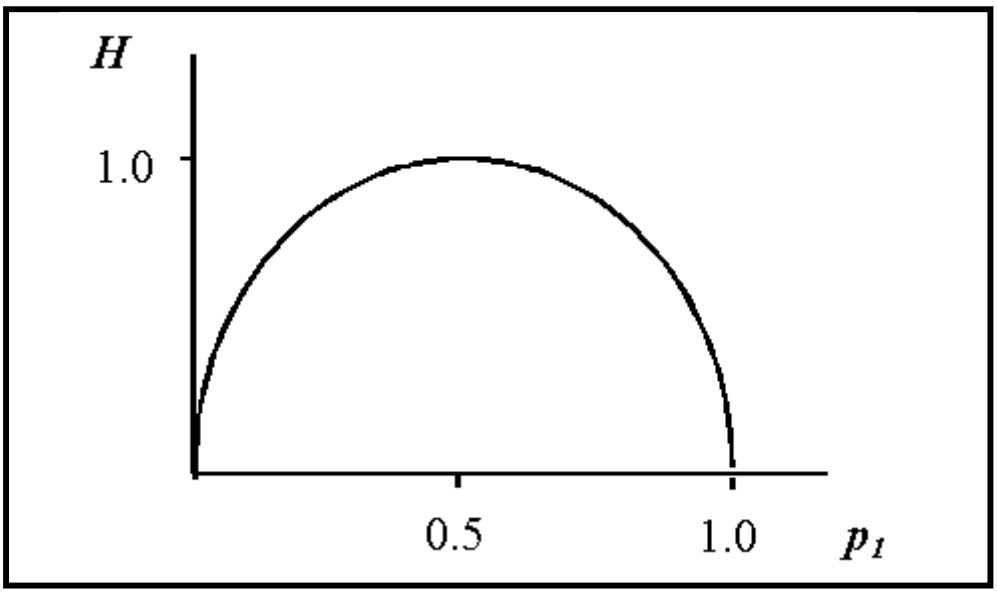


Figure 2.2: Entropy in the case of two possible events

Since in our work we manipulate lexical units, we establish a discrete random variable W that ranges over the set of words, which is the vocabulary V, and that has a probability function $p(w)$.

The entropy $H(W)$ of a discrete variable W is defined as:

$$H(W) = - \sum_{w \in V} p(w) \log p(w) \qquad (2.2)$$

It is easily understood that $p(w)$ is the probability of occurrence of lexical unit w in the corpus, calculated by the methods presented in the previous chapter. So the entropy of the lexical unit is:

$$H(w) = - p(w) \log p(w)$$

If the log is to the base 2 and the resulting units are called binary digits, as suggested by J.W. Tukey. If the base 10 is used, the resulting units are expressed in decimal digits. Making a simple calculation we find that a decimal digit is about 3.3 bits:

$$\log_2 x = \log_{10} x / \log_{10} 2$$
$$= 3.32 \log_{10} x$$

Generally if the base of the logarithm is a, the entropy is denoted as $H_a(X)$.

Entropy has a number of interesting properties:

1. $H >= 0$.
Entropy is zero if and only if all p_i are equal, except one having the value unity. In such case the uncertainty is vanished.
2. H is a maximum when all p_i are equal. For n possible events, $H_{max} = \log n$. This situation is the most uncertain.

3. **Joint Entropy**: extending the definition of entropy to a pair of lexical units, $w1, w2,$ the joint entropy $H(w_1, w_2)$ is:

$$H(w_1, w_2) = - p(w_1,w_2) \log p(w_1,w_2) \qquad (2.3)$$

, where $p(w_1,w_2)$ is the joint probability of w_1 and w_2.
It is easily proved that

$$H(w_1, w_2) <= H(w_1) + H(w_2) \qquad (2.4)$$

The uncertainty of a joint event is less than or equal to the sum of the individual uncertainties. The equality holds only if the two lexical units are independent ($p(w_1,w_2) = p(w_1)\, p(w_2)$).

4. If $p(w_1) < p(w_2)$ and decrease $p(w_2)$, increasing $p(w_1)$ an equal amount so that $p(w_1)$ and $p(w_2)$ are more nearly equal, then H increases.

5. **Conditional Entropy:** the conditional entropy $H(w2 \mid w1)$ is defined as:

$$H(w_2 \mid w_1) = - p(w_1,w_2) \log p(w_2 \mid w_1) \qquad (2.5)$$

The conditional entropy measures how uncertain we are of the possible occurrence of lexical unit w_2 when we know w_1.

6. Combining equations (2.3) and (2.5) we have:

$$H(w_1) + H(w_2) \geq H(w_1, w_2) = H(w_1) + H(w_2 \mid w_1) \qquad (2.6)$$

Thus

$$H(w_2) \geq H(w_2 \mid w_1)$$

The knowledge of w_1 never increases the uncertainty of w_2.

A variant of entropy: Perplexity

Perplexity is a metric for the evaluation of a language model. If a language model is trained on a training set, perplexity measures how well the model matches the test set. Perplexity is defined as:

$$PP = 2^H \qquad (2.7)$$

with H being the entropy a sequence of words. We focus to a sequence of words, and not to a single variable, because is more reasonable to consider the natural language to be composed of sequences of words.
For a sequence of words $W = \{ w_0,w_1,w_2,\ldots,w_n \}$ we can have a variable, which ranges over these words. The entropy of this variable is computed as:

$$H(w_0,w_1,w_2,\ldots,w_n) = - \sum_{W_1^n \in V} p(W_1^n) \log p(W_1^n) \qquad (2.8)$$

Also, the entropy rate is defined as the entropy of this sequence divided by the number of words.

Perplexity can be viewed as the weighted average number of choices a random variable has to make. In other words, with respect to Figure 2, perplexity is the probabilistic approximation of the average branching factor.
Generally, the perplexity is an indicator of the task difficulty and the linguistic constraint implied by a model trained on a specific text domain.

2.2 Mutual Information

Recall the information that conditional entropy provides: the amount of uncertainty (or certainty) about the possible occurrence of an event, if we are given the previous event. In our work the event is thought as the occurrence of a lexical unit.

Consider the following familiar event:
Imagine a reader who reads in the newspaper an article about the international computer market and he meets the word "Hewlet". Automatically he expects "Packard" to be the following word.
The reader's expectation seems to be completely reasonable, since the "Hewlet Packard" company has a distinguished position in the computer market. So, it is not likely the word "Hewlet", to be followed by other word rather than word "Packard", in a computer related article. If we want to use more formal expression, we can say that the conditional entropy of this pair results to a very low uncertainty.

This word association is a linguistic phenomenon, which can be discovered by statistical processing. For this processing an association metric is needed.

If two lexical units (words), w_1 and w_2, have probabilities $p(w_1)$ and $p(w_2)$, then their mutual information, ,$MI(w_1, w_2)$, is defined to be:

$$MI_{.}(w_1,w_2) = log \frac{p(w1, w2)}{p(w1)\, p(w2)} \qquad (2.9)$$

Since words w_1 and w_2 are individual elements, the calculated association between them is referred as point-wise mutual information.

In practice point-wise mutual information compares the joint probability of w_1 and w_2 (the probability of seeing w_1 and w_2 together) with the probabilities of observing w_1 and w_2 independently. If w_1 and w_2 are strongly associated, then the joint probability $p(w_1, w_2)$ will be much greater than $p(w1)p(w_2)$, and $MI(w_1, w_2) > 0$. If there is no interesting dependence between w_1 and w_2, then $p(w_1, w_2) \approx p(w_1)p(w_2)$ and consequently $MI(w_1, w_2) \approx 0$. Lastly, if there is no relationship between w_1 and w_2, then $p(w_1,w_2)$ will be much less than $p(w_1)p(w_2)$, and $MI(w_1, w_2) < 0$.

Average point-wise mutual information

Point-wise mutual information is sensitive to marginal probabilities $p(w_1)$ and $p(w_2)$. It tends to give higher results as $p(w_1)$ and $p(w_2)$ decrease, independently of the distribution of their co-occurrence. To avoid this

"sensitivity" a weighted measure of mutual information can be used, which is called average point-wise mutual information:

$$MI_{av.}(w_1,w_2) = p(w_1, w_2)\ log\ \frac{p(w1, w2)}{p(w1)\ p(w2)} \qquad (2.10)$$

2.3 Relation between entropy and mutual information

The mutual information (or average point-wise mutual information) is the reduction in the uncertainty of one random variable due to the knowledge of the other. Obviously, mutual information is closely related with the concept of entropy.

Consider two random variables X and Y ranging over V, with a joint probability mass function $p(x,y)$ and marginal mass functions $p(x)$ and $p(y)$. We can rewrite the definition of mutual information as:

$$\begin{aligned} MI(X,Y) &= \sum_{x,y\in V} p(x,y)\ log \frac{p(x,y)}{p(x)p(y)} \\ &= \sum_{x,y\in V} p(x,y)\ log \frac{p(x\mid y)}{p(x)} \\ &= -\sum_{x,y\in V} p(x,y)\ log(x) \ + \sum_{x,y\in V} p(x,y)\ log(x\mid y) \\ &= -\sum_{x,y\in V} p(x,y)\ log(x) \ - \left(-\sum_{x,y\in V} p(x,y)\ log(x\mid y)\right) \\ &= H(X) - H(X\mid Y) \end{aligned}$$

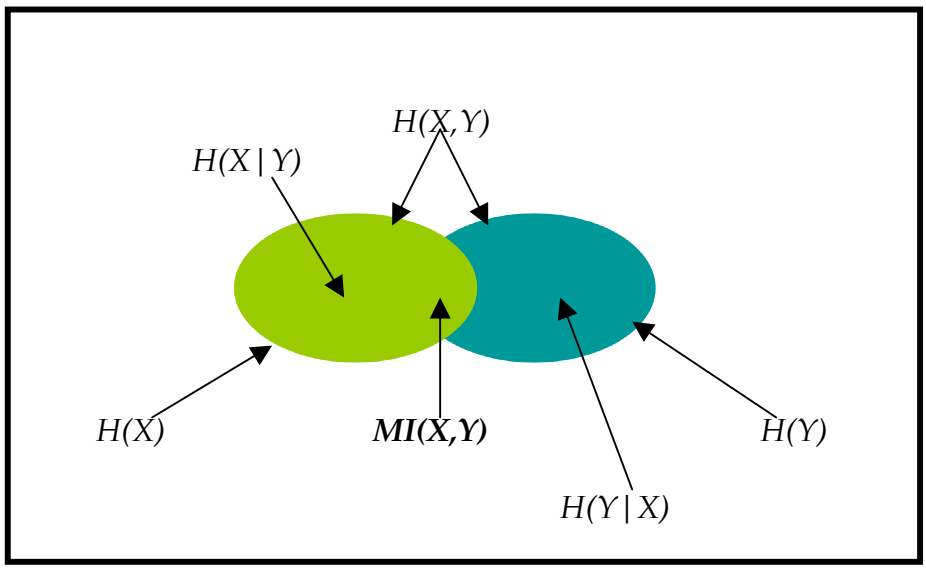


Figure 2.3: Relationship between entropy and mutual information

Therefore the mutual information *MI(X,Y)* is the reduction in the uncertainty of *X* due to the knowledge of *Y*. By symmetry, it also follows that:

$$MI(X,Y) = H(Y) - H(Y \mid X)$$

Since H(X,Y) = H(X) + H(Y | X), we have :

$$MI(X,Y) = H(X) - H(Y) - H(X,Y)$$

Finally, we can write:

$$MI(X,X) = H(X) - H(X \mid X) = H(X)$$

The mutual information of a random variable with itself is the entropy of the random variable. This is the reason that entropy sometimes is called "self-information".

All these results are expressed in figure 2.3.

2.4 Summary

In this chapter we viewed the natural language as a kind of information source. We considered the written natural language as a stochastic process that consists of a sequence of words. This consideration embodies variability and uncertainty. Next, we introduced the fundamental concept of entropy as a measure of information or as a measure of "uncertainty ". The entropy has a number of interesting properties. Through the study of these properties we defined the joint entropy and the conditional entropy, which can be applied to corpus processing providing valuable information about consecutive lexical units. Finally, we saw the concept of mutual information as a measure of word association. In corpus processing, mutual information can be a tool for identifying the words that tend to occur, almost, together. Mutual information has a central role to the implementation of our system, since is used by the component of lexical phraser in order to generate chunks, as we will see in chapter 4.

3 Contextual word similarity

The concept of word similarity was traditionally captured within thesauri. A thesaurus is a lexicographic resource that specifies semantic relationships between words, listing for each word related words such as synonyms, hyponyms and hypernyms. Thesauri have been used to assist writers in selecting appropriate words and terms and in enriching the vocabulary of a text. To this end, modern word processors provide a thesaurus as a built in tool.
The area of information retrieval has provided a new application for word similarity in the framework of query expansion. Good free-text retrieval queries are difficult to formulate since the same concept may be denoted in the text by different words and terms. Query expansion is a technique in which a query is expanded with terms that are related to the original terms that were given by the user, in order to improve the quality of the query. Various query expansion methods have been implemented, both by researchers and in commercial systems that rely on manually crafted thesauri or on statistical measures for word similarity.
Word similarity may also be useful for disambiguation and language modeling in the area of NLP and speech processing. Many disambiguation methods and language models rely on word co-occurrence statistics that are used to estimate the likelihood of alternative interpretations of a natural language utterance (in speech or text). Due to data sparseness, though, the likelihood of many word co-occurrences cannot be estimated reliably from a corpus, in which case statistics about similar words may be helpful.
Consider for example the following utterances, which may be confused by a speech recognizer.

a. The bear ran away.
b. The pear ran away.

A typical language model may prefer the first utterance if the word co-occurrence bear ran was encountered in a training corpus while the alternative co-occurrence pear ran was not. However, due to data sparseness it is quite likely that neither of the two alternative interpretations was encountered in the training corpus. In such cases information about word similarity may be helpful. Knowing that bear is similar to other animals may help us collect statistics to support the hypothesis that animal names can precede the verb ran. On the other hand, the names of other fruits, which are known to be similar to the word pear, are not likely to precede this verb in any training corpus. This type of reasoning was attempted in various disambiguation methods, where the source of word similarity was either statistical or a manually crafted thesaurus.

It should be noted that while all the applications mentioned above are based on some notion of "word similarity" the appropriate type of similarity relationship might vary. A thesaurus intended for writing assistance should identify words that resemble each other in their meaning, like aircraft and airplane, which may be substituted for each other. For query expansion, on the other hand, it is also useful to identify contextually related words, like aircraft and airline, which may both, appear in relevant target documents. Finally, co-occurrence-based disambiguation methods would benefit from identifying words that have similar co-occurrence patterns. These might be words that resemble each other in their meaning, but may also have opposite meanings, like increase and decrease [24].

In this chapter, we consider theoretical and computational properties of several functions measuring the similarity between distributions. We refer to these functions as distance functions and, in one case, as similarity functions.

Definition of "distance function"

A metric space is a set X together with a function d (called a "metric" or "distance function") which assigns a real number $d(x,y)$ to every pair $x,y \in X$ satisfying the properties:

1. $d(x,y) \geq 0$ *and* $d(x,y) = 0 \Leftrightarrow x=y$,
2. $d(x,y) = d(y,x)$,
3. $d(x,y) + d(y,x) \geq d(x,z)$

The last property is called the "triangle inequality" because it says that the sum of two sides of a triangle is at least as big as the third side.

3.1 Kullback-Leibler (KL) distance

We define the function $D(q||r)$ as

$$D(q||r) = \sum_{y \in Y} q(y) \log \frac{q(y)}{r(y)} \qquad (3.1)$$

Function (3.1) goes by many names in the literature, including information gain, relative entropy, cross entropy, and Kullback Leibler distance. Kullback himself refers to the function as information for discrimination, reserving the term "divergence" for the symmetric function ($D(q||r) + D(r||q)$. We will use the name "Kullback-Leibler (KL) distance" throughout this thesis. The KL distance is a standard information-theoretic measure of the dissimilarity between two probability mass functions. It is not a metric in the technical sense, for it is not symmetric ($D(q||r) \neq D(r||q)$) and does not obey the triangle inequality

Since the KL distnace is 0 when the two distributions are exactly the same and greater than 0 otherwise, it is a measure of dissimilarity. This yields an intuitive explanation of why we should not expect the KL divergence to obey the triangle inequality: dissimilarity is not transitive.

What motivates the use of the KL divergence, if it is not a true distance metric?

Let Y be a random variable taking values in Y. Suppose we are considering exactly two hypotheses about Y: Y is distributed according to $q(H_q)$, and Y is distributed according to $r(H_r)$. Based on Bayes' rule, the posterior probabilities of the two hypotheses are written as:

$$p(H_q \mid y) = \frac{p(Hq)q(y)}{p(Hq)q(y) + p(Hr)r(y)}$$

and

$$p(H_r \mid y) = \frac{p(Hr)r(y)}{p(Hq)q(y) + p(Hr)r(y)}$$

Taking logs of both equations and subtracting:

$$\log\frac{q(y)}{r(y)} = \log\frac{p(Hq \mid y)}{p(Hr \mid y)} - \log\frac{p(Hq)}{p(Hr)}$$

We can therefore consider $\log\frac{q(y)}{r(y)}$ to be the information y supplies for choosing H_q and H_q. $D(q \mid\mid r)$ is then the average information for choosing H_q over H_q. So, the KL distance measures the dissimilarity between two distributions, since the greater their divergence is, the easier it is, on average, to distinguish between them.

Using a bigram language model, the KL distance between two words (lexical units), w_1 and w_2, is measured as the distance between the two conditional distributions $p(v \mid w_1)$ and $p(v \mid w_2)$ over the vocabulary V ($v \in V$):

$$D(w_1,w_2) \equiv D(p_1 \mid\mid p_2) \qquad (3.2)$$

,where $p_1 \equiv p(v \mid w_1)$ and $p_2 \equiv p(v \mid w_2)$.

Combining equations (x.1) and (x.2) we can write:

$$D(w_1,w_2) = \sum_{v \in V} p(v \mid w_1) \log \frac{p(v \mid w_1)}{p(v \mid w_2)} \qquad (3.3)$$

$D(w_1,w_2)$ defined only if $p(v \mid w_2)$ is greater than 0 whenever $p(v \mid w_1)$ is. This condition does not hold in general when using the Maximum Likelihood Estimator (MLE), where the estimate for $p(v \mid w_2)$ is 0 when $freq(v \mid w_2)=0$. This forces using a smoothed estimator which assigns non-zero probabilities for all $p(v \mid w_2)$ even when $freq(v \mid w_2)=0$. However, having zero association values gives a computational advantage, which cannot be exploited when using the KL distance as a similarity measure. Moreover, the need to use a smoothing method both complicates the implementation of the word similarity method and may introduce an unnecessary level of noise into the data. The Information-Radius distance remedies this problem.

In order to calculate the similarity between two lexical units according to their lexical environment, we have to take into account left and right contexts. Let's consider a word w with its neighbors in a word sequence

$$\ldots \quad v_{1,L} \quad w \quad v_{1,R} \quad \ldots$$

with $v_{1,L}$ representing the word in the left context and $v_{1,R}$ representing the word in the right context. Two probability distributions are calculated, $p(v_{1,L} \mid w)$ and $p(v_{1,R} \mid w)$, for the left and right contexts respectively. The right-context bigrams are calculated using the usual word order, and the left-context bigrams are calculated with a reversed-order training corpus.

In order to estimate the similarity of two words, w_1 and w_2, we need the sum of the symmetric left and right context-depended distances [5]. So, the total distance between the probability distributions for w_1 and w_2 is :

$$D^{LR}(w_1,w_2) = D_{12}^{L} + D_{21}^{L} + D_{12}^{R} + D_{21}^{R} \qquad (3.4)$$

According to equation (3.3) :

$$D_{12}^{L} = \sum_{v_{1,L} \in V} p_1^{L}(v_{1,L} \mid w_1) \log \frac{p_1^{L}(v_{1,L} \mid w_1)}{p_2^{L}(v_{1,L} \mid w_2)} \qquad (3.5.a)$$

$$D_{21}^{L} = \sum_{v_{1,L} \in V} p_2^{L}(v_{1,L} \mid w_2) \log \frac{p_2^{L}(v_{1,L} \mid w_2)}{p_1^{L}(v_{1,L} \mid w_1)} \qquad (3.5.b)$$

$$D_{12}^{R} = \sum_{v_{1,R} \in V} p_1^{R}(v_{1,R} \mid w_1) \log \frac{p_1^{R}(v_{1,R} \mid w_1)}{p_2^{R}(v_{1,R} \mid w_2)} \qquad (3.5.c)$$

$$D_{21}^{R} = \sum_{v_{1,R} \in V} p_2^{R}(v_{1,R} \mid w_2) \log \frac{p_2^{R}(v_{1,R} \mid w_2)}{p_1^{R}(v_{1,R} \mid w_1)} \qquad (3.5.d)$$

The Kullback-Leibler distance is unbounded since it includes ratios whose denominators may approach zero. This has the consequence that a few terms can dominate the calculation of the KL distance. This is an important issue the case of language modeling for new domains for which here are limited training data and the statistics can be rather poor, with only one or two observations for some extant N-grams. This raises the need to use other bounded metrics, which are described next.

3.2 Information-Radius (Shannon-Jensen) distance

The Information-Radius (IR) distance is similar to the KL distance but is bounded because the denominator for the logarithmic ratio is the average of the two probabilities being considered, and is defined as:

$$IR(q||r) = \sum_{y \in Y} q(y) \log \frac{q(y)}{\frac{1}{2}(q(y) + r(y))} \qquad (3.6)$$

By definition $\frac{1}{2}(q(y)+r(y))$ is greater than 0. Therefore IR distance, unlike KL distance, does not impose any constraints on the input data. Estimates that approach zero can be used directly [5]. It can be shown that $IR(q||r)$ ranges between 0 and log2.

Calculating the similarity of two words, w_1 and w_2, taking into account the symmetric left and right context-depended distances, we write:

$$IR^{LR}(w_1,w_2) = IR_{12}^{L} + IR_{21}^{L} + IR_{12}^{R} + IR_{21}^{R} \qquad (3.7)$$

As in previous metric, we define the four terms as follows:

$$IR_{12}^{L} = \sum_{v_{1,L} \in V} p_1^{L}(v_{1,L}|w_1) \log \frac{p_1^{L}(v_{1,L}|w_1)}{\frac{1}{2}(p_1^{L}(v_{1,L}|w_1)+p_2^{L}(v_{1,L}|w_2))} \qquad (3.8.a)$$

$$IR_{21}^{L} = \sum_{v_{1,L} \in V} p_2^{L}(v_{1,L}|w_2) \log \frac{p_2^{L}(v_{1,L}|w_2)}{\frac{1}{2}(p_2^{L}(v_{1,L}|w_2)+p_1^{L}(v_{1,L}|w_1))} \qquad (3.8.b)$$

$$IR_{12}^{R} = \sum_{v_{1,R} \in V} p_1^{R}(v_{1,R}|w_1) \log \frac{p_1^{R}(v_{1,R}|w_1)}{\frac{1}{2}(p_1^{R}(v_{1,R}|w_1)+p_2^{R}(v_{1,R}|w_2))} \qquad (3.8.c)$$

$$IR_{21}^{R} = \sum_{v_{1,R} \in V} p_2^{R}(v_{1,R}|w_2) \log \frac{p_2^{R}(v_{1,R}|w_2)}{\frac{1}{2}(p_2^{R}(v_{1,R}|w_2)+p_1^{R}(v_{1,R}|w_1))} \qquad (3.8.d)$$

3.3 Manhattan-norm (L_1) distance

If we think of probability mass functions as vectors, so that distribution p is associated with the vector $(p(y_1), p(y_2),\ldots.., p(y_N))$ in $< R^N$, then we can measure the distance between distributions. This metric can be viewed as a geometric distance and also, is a true metric metrics, as the name "norm" suggests.

The Manhattan distance is defined as:

$$MN(q,r) = \sum_{y \in Y} |q(y) - r(y)| \qquad (3.9)$$

The Manhattan-norm distance is the absolute value of the difference between two distributions and is closely related with the Euclidean distance, as is shown schematically by the following figure:

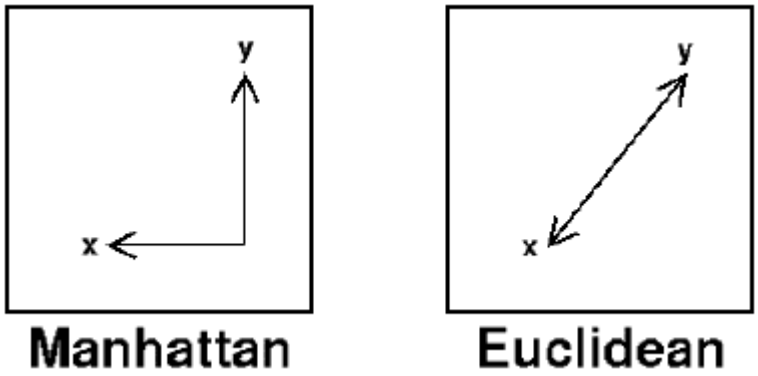


Figure 3.1: Manhattan-norm and Euclidean distance.

Clearly, $MN(q,r) = 0$ if and only if $q(y) = r(y)$ for all y. Moreover, it has shown that $MN(q,r)$ has an upper bound of two. Interestingly, $MN(q,r)$ bears the following relation, to $D(q||r)$:

$$MN(q,r) = (D(q||r). 2 \ln b)^{½}$$

,where b is the base of the logarithm used in the distance $D(.)$. Consequently, convergence in KL distance implies convergence in the Manhattan-norm distance.

The similarity of two words, w_1 and w_2, is calculated as the sum of the symmetric left and right context-depended distances:

$$MN^{LR}(w_1,w_2) = MN_{(12)}{}^{L} + MN_{(12)}{}^{R} \qquad (3.10)$$

The Manhattan-norm distance is symmetric since $MN_{12} \equiv MN_{21}$ [5]. The two terms of the equation (3.10) are defined as:

$$MN_{(12)}^{L} = \sum_{v_{1,L} \in V} | \, p_1^{L}(v_{1,L} \,|\, w_1) - p_2^{L}(v_{1,L} \,|\, w_2) \, | \qquad (3.11.a)$$

$$MN_{(12)}^{R} = \sum_{v_{1,R} \in V} | \, p_1^{R}(v_{1,R} \,|\, w_1) - p_2^{R}(v_{1,R} \,|\, w_2) \, | \qquad (3.11.b)$$

3.4 Vector Product similarity (cosine measure)

This geometric metric is a similarity measure, rather than a difference measure. It is related to the angle between two vectors; the "closer" two vectors are, the smaller the angle between them.

$$VP(q,r) = \frac{\sum_{v \in V} q(y) r(y)}{\sqrt{\sum_{v \in V} q(y)^2 \sum_{v \in V} r(y)^2}} \qquad (3.12)$$

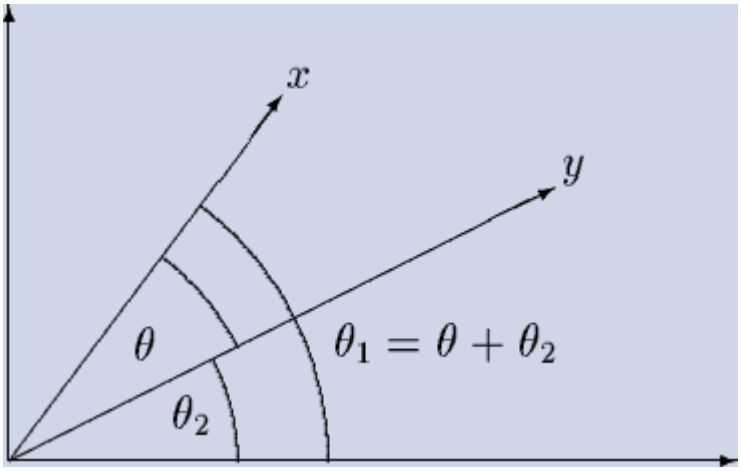


Figure 3.2: Angle between vectors. In our work each vector is a sequence of bigram probabilities.

Notice that the cosine measure is an inverse distance function, in that it achieves an upper bound of 1 when $q(y) = r(y)$ for all y, and is zero when the supports of q and r are disjoint. For all the other distances described above, it is just the opposite: they are zero if and only if $q(y) = r(y)$ for all y, and are greater than zero otherwise.

As before, the similarity between w_1 and w_2 is calculated as the sum of the symmetric left and right context-depended distances:

$$VP^{LR}(w_1, w_2) = VP_{12}^{L} + VP_{12}^{R} \qquad (3.13)$$

The Vector Product similarity is symmetric, as Manhattan distance, since $VP_{12} \equiv VP_{21}$ [5]. The two terms of the equation (3.13) are defined in the following equations (3.14.a and 3.14.b):

$$VP_{12}{}^{L} = \frac{\sum_{v_{1,L} \in V} p_1{}^{L}(v_{1,L} | w_1)\, p_2{}^{L}(v_{1,L} | w_2)}{\sqrt{\sum_{v_{1,L} \in V} p_1{}^{L}(v_{1,L} | w_1)^2 \sum_{v_{1,L} \in V} p_2{}^{L}(v_{1,L} | w_2)^2}}$$

$$VP_{12}{}^{R} = \frac{\sum_{v_{1,R} \in V} p_1{}^{R}(v_{1,R} | w_1) p_2{}^{R}(v_{1,R} | w_2)}{\sqrt{\sum_{v_{1,R} \in V} p_1{}^{R}(v_{1,R} | w_1)^2 \sum_{v_{1,R} \in V} p_2{}^{R}(v_{1,R} | w_2)^2}}$$

In the following table we summarize some properties of the described metrics for comparative purposes:

Measure	Bounded	Range of values	Measure of similarity/dissimilarity	Symmetric
KL	No	0. . .∞	Dissimilarity	No
IR	Yes	0. . .4log2	Dissimilarity	No
MN	Yes	0. . .4	Dissimilarity	Yes
VP	Yes	0. . .2	Similarity	Yes

Table 3.1: Four metrics for context supported word similarity.

3.5 Our approach

The main purpose of our work is, firstly, to use these metrics in order to infer from a given text the similarity between two lexical units and using an iterative procedure, to cluster similar units that are able to form a semantic class [5]. Our approach examines the lexical environment in which the lexical units appear, using statistics provided by a trained language model. The detailed procedure is described in the next chapter.

3.6 An extension of similarity measures: document–oriented approach

Search engines, such as Google, allow the users to search based on keyword or other criteria. However, sometimes one may have a document that is exactly what is needed, but additional sources are needed. It would very convenient to find additional similar websites without having to determine the keywords and searching on a search engine. Document matching could also be used by doctors to find other diseases with similar symptoms or by biologists to match different types of bacteria.

As is standard in Information Retrieval, each document is represented by a vector, which specifies how many times each word occurs in the document (the word frequencies). These counts are weighted to reflect the importance of each word. The weighting is the inverse of the log of the number of documents each word occurs in (the inverse document frequency). This vector of weighted counts is called a "bag of words" representation. Words from a specific list of "stop words" are not included in the representation. Also, words, which occur in few documents, are removed from the document representation, because they are too infrequent.

A popular measure of similarity for text clustering is the cosine of the angle between two vectors (Vector Product distance). An important property is that the cosine measure does not depend on the length. This allows documents with the same composition, but different totals to be treated identically which makes this the most popular measure for text documents.

3.7 Inducing semantic classes: the need for an automatic approach

Manual methods for word grouping suffer from disadvantages when compared with the automatic corpus-based approach:

1. All entries of a semantic group must be encoded by hand, which requires manual effort.

2. Changes to lexical entries may necessitate the careful examination and potential revision of other related entries in order to maintain the consistency of the whole list of semantic groups.

3. Most dictionaries emphasize the syntactic features of words, such as part of speech, number, and form of complement. Even when dictionary designers try to focus on the semantic component of lexical knowledge, the results have not yet been fully satisfactory.

4. The lexical information is not specific to any domain. Rather, the entries attempt to capture what applies to the language at large, or represent specialized senses in a disjunctive manner. Note that semantic lexical knowledge is most sensitive to domain changes. Unlike syntactic constraints, semantic features tend to change as the word is used in a different way in different domains. This inherent property of natural language was mentioned in the introduction of this work, indicating the trend of human to adjust the use of language according to a "least effort model", in order to communicate more easily, especially in spoken natural language.

5. Time-varying information, that is, the currency of words, compounds, and collocations, is not adjusted automatically.

6. The validity of any particular entry depends on the assumptions made by the particular lexicographers who compiled that entry. In contrast, an automatic system can be more thorough and impartial, since it bases its decisions on actual examples drawn from the corpus.

An automatic corpus-based system for lexical knowledge extraction offsets these disadvantages of static human-constructed knowledge bases by automatically adapting to the domain "sublanguage". Its disadvantage is that while it offers potentially higher recall, it is generally less precise than knowledge bases carefully constructed by human lexicographers. This disadvantage can be alleviated if human experts in a post-editing phase modify the output of the automatic system [3].

3.8 Summary

The main subject of this chapter is the contextual word similarity. In other words we tried to find how the similarity between words that occur in a corpus could be measured, according to their lexical environment (the lexical context of the corpus). During our study we decided to take into account the left and the right contexts. The left context can be taken by reversing the corpus. Next, we studied four different metrics for the computation of the similarity between words: Kullback-Leibler distance, Information-Radius distance, Manhattan-norm distance and Vector Product similarity. The first three metrics are measures of dissimilarity, while the last metric is a measure of similarity. The study of these metrics was based in bigram language model. These metrics will be used by the component of semantic generalizer during an iterative procedure, in order to induce groups of words that occur in the same lexical environment, as we will see in the next chapter. Such groups can form semantic classes since conceptually similar words tend to occur in the same lexical environment. Finally, we saw the advantages of an automatic procedure of inducing semantic classes.

4 Experimental procedure

In this chapter we describe the several steps for the automatic induction of semantic classes. Our work was strongly based in the procedure that is proposed in [5]. The whole implementation was done, mainly, in Perl. The clustering algorithm presented in 4.4.2 was implemented in Python. The required N-gram statistics were calculated by the CMU toolkit [16]. A brief description of the CMU toolkit, as well as some comments about Perl, can be found in the Appendix.

4.1 Calculating the required statistics

The first step in designing an understanding module, contained in a dialogue agent, is to obtain a corpus of transcribed utterances. We have worked on a single domain, ATIS, which is an air reservation system. The used unannotated corpus is composed of transcribed human requests (utterances) taken from a human-machine spoken dialogue interaction over the telephone network.

The used ATIS corpus is a small homogeneous corpus. The term "homogeneous" means that in the transcribed requests there is no expressional complexity and the used vocabulary is limited to the purpose of the provided service. In contrast, the Wall Street Journal (WSJ) is a collection of financial news articles, consisting of very long sentences written by professional editors on various topics.

The training sentences are the source of knowledge, which is used to train the statistical parameters of the language model. We used the 1996 version of the CMU toolkit to calculate the N-gram statistics. We decided to use this toolkit for many reasons. Firstly, the desired statistics are calculated in a minimal processing time, since in CMU's 1996 version, code's optimization techniques have been implemented. Secondly, the toolkit provides a lot of options in several parameters presented in Appendix. Moreover, by keeping the CMU toolkit as a reference toolkit, pieces of code that were written for other applications, also based on CMU, can be reused in slightly different future projects.

The statistics calculated by the CMU toolkit are used by the following system.

4.2 Short description of the system

The proposed procedure for automatically inducing semantic classes works iteratively. Iterating continues until a reasonable number of classes is reached.

The system consists of three main components:

(a) a lexical phraser
(b) a semantic generalizer
(c) a corpus parser

Each component directs its output to the input of the component, which follows, as shown schematically in Figure 4.1.

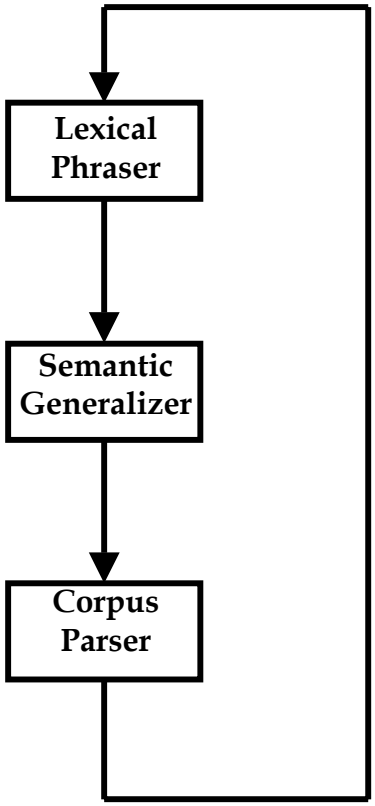


Figure 4.1: Iterative procedure for the auto-induction of semantic classes.

Firstly, the lexical phraser groups words in single lexical units. Next, the semantic generalizer generates rules that map words (and concepts) to concepts corresponding to semantic classes. In the final step, a corpus parser reparses the corpus, using the rules that were generated by the semantic generalizer. Then, the described procedure is repeated.

4.3 Lexical Phraser

The actual output of lexical phraser is a list of the most commonly co-occurring lexical units. The lexical phraser search over all consecutive words and groups them into phrases, using an association measure. The mission of this component is to find the lexical entities that co-occur often. In other words, it has to identify the associated individual lexical units within the lexical environment that can behave as an individual unit. For example, in the travel domain words such as "KANSAS" and "CITY" co-occur often. It is obvious that they are associated through a close lexical relation and they are able to behave as an individual unit in the lexical environment. Thus, it is critical to treat these words as a single one. Working with machine-readable texts, this can be done by eliminating the white space between them, or by substituting it with a predefined special symbol. (This idea can be applied in the cases where the white space is used in order to separate words within sentences). So, frequently co-occurring lexical units are chunked into a single phrase (chunk) as follows:

KANSAS CITY => *KANSASCITY*
or
KANSAS CITY => *KANSAS_CITY*
or
KANSAS CITY => *KANSAS*<>*CITY*

The used special symbol indicating the chunking action does not play any particular role, but obviously, in the above example, the last two transformations are more comprehensive than the first one. In our work we used the symbol "<>" to indicate the grouping of two lexical units into a single one. Up to this point we have established a notation indicating the form a chunk. The next step is to substitute the chunks, provided by the component of lexical phraser, to the original corpus. Assume that the lexical phraser associates the word pairs WOULD,LIKE and SAN,JOSE. Also consider the following sentences from the original corpus, where these four words are occurred:

. . .

I WOULD LIKE AN EARLY MORNING FLIGHT TODAY FROM LOS ANGELES TO CHARLOTTE

. . .

LIST AMERICAN AIRLINES FLIGHTS FROM MILWAUKEE TO SAN JOSE

. . .

Figure 4.2: A part of the original corpus.

According to the results of the lexical phraser, the chunks WOULD<>LIKE and SAN<>JOSE must substitute the patterns «WOULD LIKE» and «SAN JOSE», as is shown in the following figure:

. . .

I WOULD<>LIKE AN EARLY MORNING FLIGHT TODAY FROM LOS ANGELES TO CHARLOTTE

. . .

LIST AMERICAN AIRLINES FLIGHTS FROM MILWAUKEE TO SAN<>JOSE

. . .

Figure 4.3: A part of the transformed corpus according to the chunks.

Hierarchical Phrasing: However, there are a lot of cases where a reasonable chunk consists of more than two words. In such cases we must ensure that the lexical phraser is able to associate the proper words and form the correct chunk. This task is quite difficult, since the lexical phraser has to capture the exact count of words; neither less nor more. This need introduces the idea to permit the lexical phraser to operate on its own output inducing a hierarchical phrasing.

A representative example is the chunk "J<>F<>K", an airport's name. In the AtisTrain corpus the three letters are separated by two white spaces.

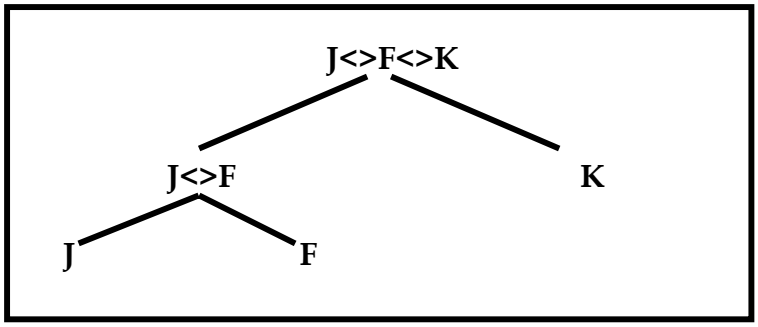


Figure 4.4: An example of hierarchical phrasing. The chunk J<>F<>K is formed after the second iteration of lexical phraser.

The previous example shows how the chunk J<>F<>K is formed. After the first iteration of lexical phraser, letters "J" and "F" are grouped into a single lexical unit. Furthermore, a second iteration is needed to group "J<>F" and "K" to the correct chunk. Sometimes, chunks consisting of three words are likely to be formed completely, during the first operation. However, our study showed that more reasonable chunks are obtained if the lexical phraser operates more than one time.

The transformed corpus is then given as input to the CMU toolkit in order to calculate the appropriate statistics. Figure 4.5 presents schematically the iterative operation of lexical phraser.

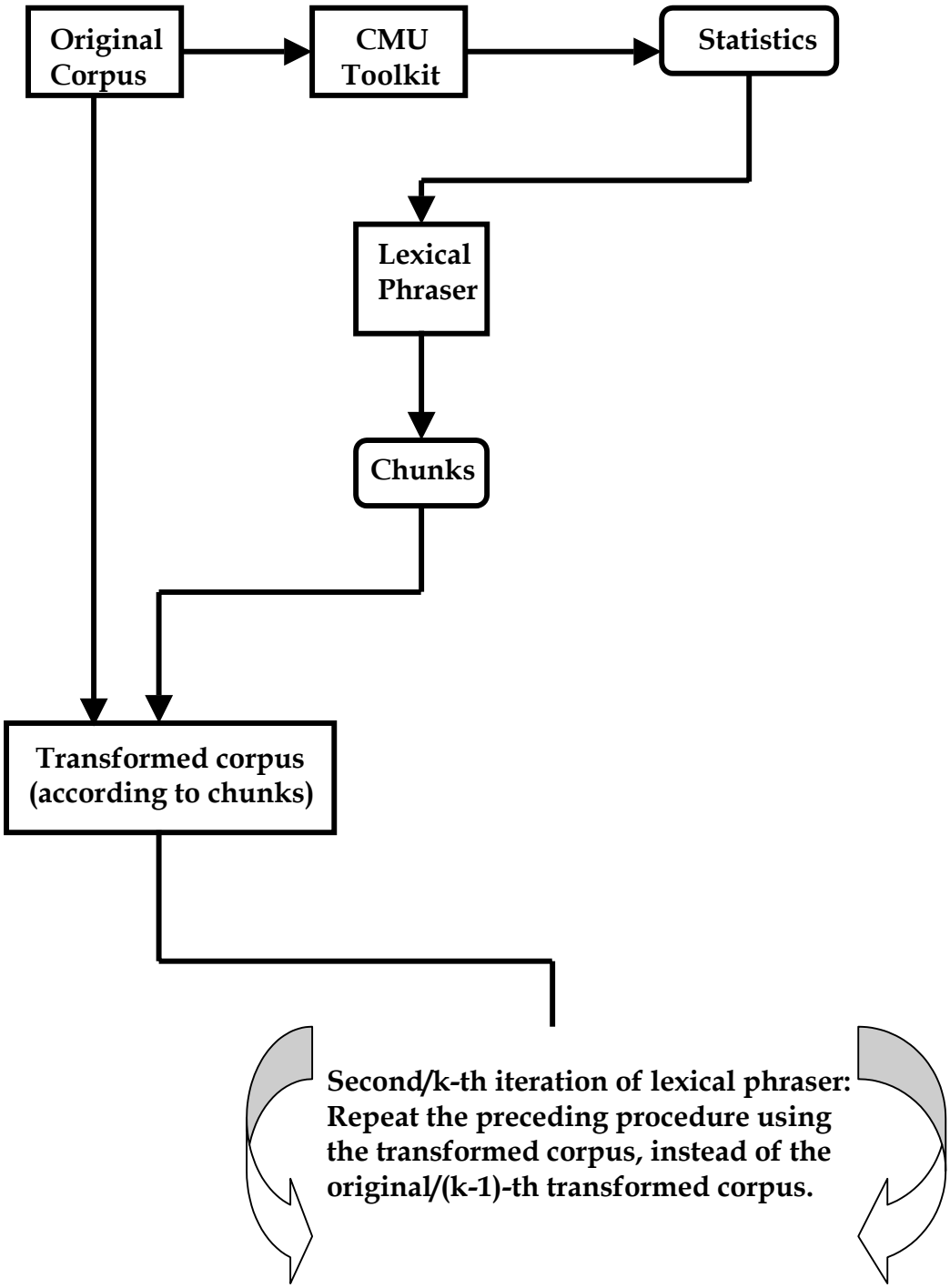


Figure 4.5: Multiple iterations of lexical phraser. At the end of iteration the corpus is transformed according to the resulting chunks.

As is shown in Figure 4.5, initially we use the CMU toolkit to build a bigram language model (unigram probability, unigram backoff and bigram probability are calculated). The lexical phraser uses these results in order to associate the several lexical units calculating the mutual information between them. At the end of the iteration we retain the *n* chunks with the largest mutual information. Then the previously used corpus is transformed

according to the *n* chunks. Lastly, the lexical phraser can repeat the same procedure based on the updated corpus (the language model is retrained), inducing new chunks. Otherwise, the updated corpus is passed to the component of semantic generalizer that follows.

The results produced by the CMU toolkit: In the following figure we present a representative part of the .arpa file which is generated by the CMU toolkit, containing the required statistics for the calculation of the mutual information.

```
##############################################################
## Copyright (c) 1996, Carnegie Mellon University, Cambridge University,
## Ronald Rosenfeld and Philip Clarkson
##############################################################
. . . . . .
p(wd2|wd1) = if(bigram exists) p_2(wd1,wd2)
                    else          bo_wt_1(wd1)*p_1(wd2)
All probs and back-off weights (bo_wt) are given in log10 form.
. . . . . .
\1-grams:
p_1     wd_1 bo_wt_1
\2-grams:
p_2     wd_1 wd_2 bo_wt_2
\1-grams:
. . . . . .
-2.8061 AIRPORT    -0.3163
-3.0791 AIRPORTS   -0.3822
-3.9822 ALASKA     -0.4736
. . . . . .
\2-grams:
. . . . . .
-0.0039 LAS VEGAS 0.0036
-0.1761 LAST FLIGHT -0.2573
-0.5006 LATE AFTERNOON -0.1022
. . . . . .
```

Figure 4.6: A sample from .arpa file produced by CMU toolkit. Notice that probabilities (left column) and backoff weights (right column) are given in log_{10} form. Witten Bell discounting method was applied.

As said before, each time the corpus is being transformed, an updated .arpa file must be generated. The only "tricky" point of this file is the fact that the probabilities and backoff weights are given in log_{10} form. In order to get the desired value we simply apply an "antilog" function. For example, the probability of the bigram "LAS VEGAS" is $10^{-0.0039}$.

Mutual Information: For measuring word association, we used the information theoretic concept of mutual information, as is expressed in a weighted, point-wise form by the following equation:

$$MI(w_1,w_2)=\tfrac{1}{2}(p(w_1,w_2)+p(w_2,w_1))log\frac{\tfrac{1}{2}(\,p(w_1,w_2\,|\,\lambda_F)+p(w_2,w_1\,|\,\lambda_B)\,)}{p(w_1)\,p(w_2)} \qquad (4.1)$$

,where λ_F denotes the usual text order and λ_B denotes the reversed text order. In general, $p(w_1,w_2) = p(w_2,w_1)$. However for estimating $p(w_1,w_2)$ we use $p(w_1\,|\,w_2)\,p(w_2)$ or $p(w_2\,|\,w_1)\,p(w_1)$, according to λ_B and λ_F respectively.
Equation (4.1) is similar with (2.10). The only difference between them is that the equation (2.10) takes into account only the left contexts. Equation (4.1) is a form of mutual information that is computed according left and right contexts. In the next paragraphs we will show how this can be implemented.
The joint probability of two consecutive words, $p(w_1,w_2)$, is calculated through the chain rule.

In [10], a different approach is applied for the estimation of joint probabilities regarding the calculation of mutual information. For a pair of words w_1 and w_2 the joint probability is estimated by counting the number of times that w_1 is followed by w_2 in a window of x words, $f_x(w_1,w_2)$, and normalizing by the size of corpus. (For example for expressions such as, w_1 *and* w_2, the size of window equals to three.) The window size parameter allows a search at different scales. Smaller window sizes identify fixed expressions and other relations that hold over short ranges. Larger window sizes highlight semantic concepts and other relationships that hold over larger scales.

In our work we used a window of one value for two main reasons: (a) the training sentences are characterized by limited length, and (b) the primary mission of the lexical phraser is to form chunks that will replace their members in later iterations within the corpus, ensuring that the chunk will behave as an individual lexical unit.

Equation (4.1) is a symmetric version of mutual information. As noted in chapter 2, mutual information, in an initial form, is defined to be:
$MI(w_1,w_2) = log\frac{p(w_1,w_2)}{p(w_1)\,p(w_2)}$. Joint probabilities are supposed to be symmetric: $p(w_1,w_2) = p(w_2,w_1)$, and thus, mutual information is also symmetric: $MI(w_1,w_2) = MI(w_2,w_1)$.

Let's consider the initial form of mutual information as an association ratio between two lexical units. Technically, this association ratio is not symmetric, since $f(w_1,w_2)$,used for the calculation of $p(w_1,w_2)$, encodes linear precedence. (Recall that $f(w_1,w_2)$denotes the number of times that w_1 appears before w_2.) In order to use a symmetric form of the association ratio, according to the nature

of mutual information, we introduce in equation (4.1) the term $p(w_2,w_1)$, which encodes the reverse precedence than $p(w_1,w_2)$ does. Practically, reversing the training corpus and training a second bigram language model through CMU toolkit, we calculate the required statistics.
In particular, the lexical phraser is the implementation of the equation (4.1). As was mentioned, at the end of chunker's iterations the n chunks with the largest mutual information are retained.

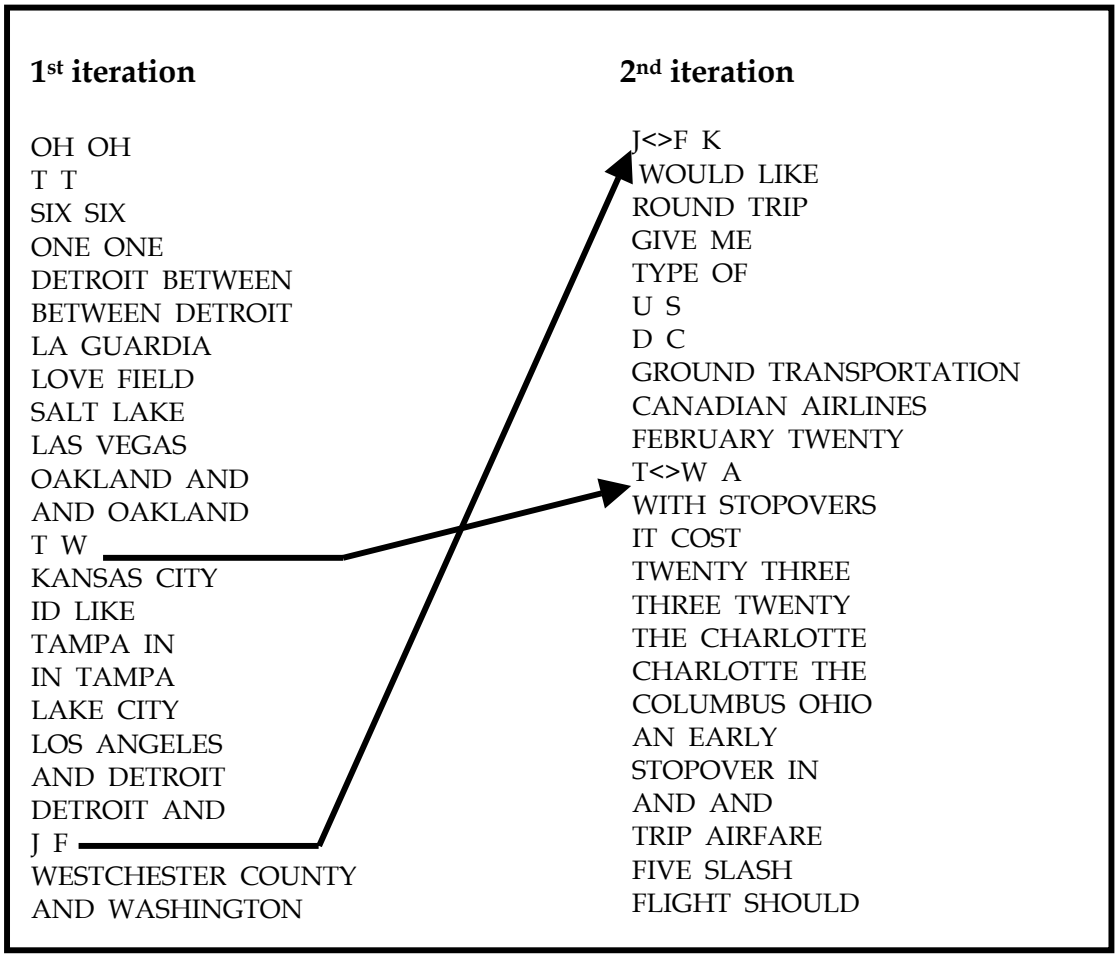
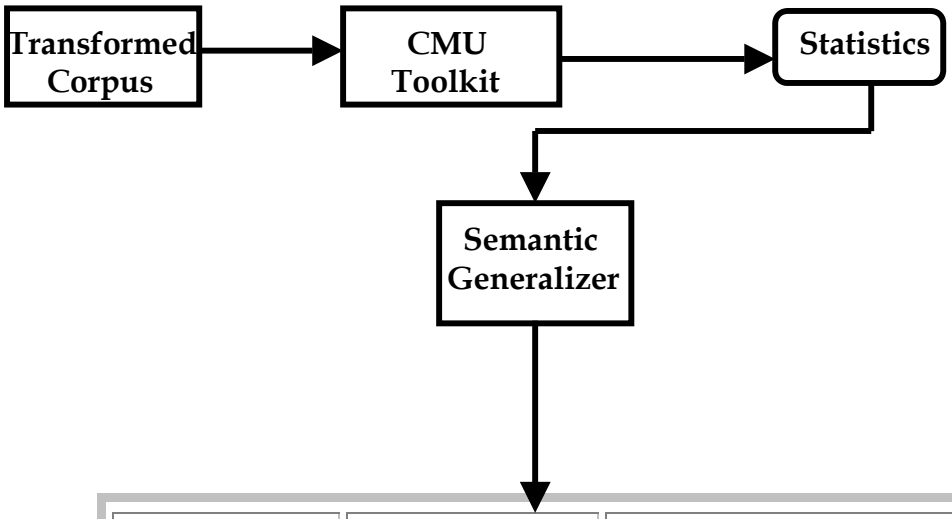


Figure 4.7: Top chunks in descending order. The chunker iterated twice.

The lexical phraser can operate on its output during later iterations. As is shown in Figure 4.7, "J<>F<>K" and "T<>W<>A" are induced completely at the end of second iteration. Having described the general functionality of lexical phraser, a critical question arises concerning the output list: how many candidate chunks should be selected? It is clear that in the lower positions of the list, the association between the lexical units is less tight, in contrast to higher positions. The answer is subject to careful observations through experimentation. As noted in [5], 20 to 40 is a reasonable number for small corpora. If we select, for example, fewer than 10 chunks, certain commonly co-occurring lexical units, such as "*WOULD LIKE*", will not be combined. More than 50 chunks may cause so many nested chunks that even an entire sentence can be transformed to a chunk. This fatal situation is more likely in corpora containing sentences of short length.

4.4 Semantic Generalizer

The next step is the induction of semantic classes. The semantic generalizer has as input the transformed corpus according to the induced chunks. As in the case of lexical phraser the required statistics are taken from the .arpa file. The output of the lexical phraser is a long list of semantically similar pairs of lexical units.



Left Member	Right Member	Distance/Similarity Score
LexicalUnitA	LexicalUnitB	xxx
LexicalUnitC	LexicalUnitD	xxx
LexicalUnitB	LexicalUnitE	xxx
LexicalUnitA	LexicalUnitG	xxx
…	…	…

Figure 4.8: The output of the semantic generalizer.

4.4.1 The criterion of similarity

In order to construct the list of semantically similar pairs we apply a dissimilarity /similarity measure. Each member of the pair is semantically related with the other member, in a degree expressed by the corresponding dissimilarity /similarity score.

Our hypothesis is that the words, which occur in the same bigram lexical context, have similar semantic meanings.

Context thresholding: We used a context threshold of three. A context threshold of three means that a word is not considered for merging into semantic group unless it occurs in the corpus at least three times with two other words in the right and left contexts. This eliminates singletons, such as cases where the human subject uses a single word to answer a system query. In general, context thresholding restricts candidates to those words well represented in a broader lexical context.

This list reflects the only available source of knowledge to us, since the dissimilarity /similarity measure takes into account the lexical environment where lexical units occur, without part of speech tagging.
In some sense, the functionality of the applied dissimilarity /similarity metric is very close in spirit to the pattern matching.

We used the following dissimilarity/similarity measures:

(a) Kullback-Leibler distance (KL)
(b) Information-radius distance (IR)
(c) Manhattan- norm distance (MN)
(d) Vector product similarity (VP)

They were calculated taking into account the left and right context, as is noted in equations 3.5(a-d), 3.8(a-d), 3.11(a-b) and 3.14(a-b), respectively. In order to get right context's statistics, we reversed the corpus and we trained the corresponding bigram language model.

We can say that the implementation of those metrics is the core of the semantic generalizer. In the next chapter we compare them based on some experimental results.

4.4.2 Grouping semantically similar lexical units

At first glance, the format of list does not propose a direct criterion for clustering. However, an initial idea is to find a common feature among the

pairs. Hence, pairs, which have the same member, can be clustered into the same semantic class.
The algorithm for the clustering works as follows:

[Step 1]. take the first pair from the list

[Step 2]. create the first class and put the members of the first pair into this class

[Step 3]. take the next pair from the list and do:

(i). if none of members is included in the previous class/es, then create a new class and include both of them in it. Return to step 3 if the last pair of list is not taken else stop.
(ii). if the pair has only one member that is included in a class, then put the other member into this class if is not included in any of the previous classes. Return to step 3 if the last pair of list is not taken else stop.
(iii) if both of members are included in a class, then ignore this pair. Return to step 3 if the last pair of list is not taken else stop.

The proposed algorithm groups semantically similar lexical units as is shown in Figure 4.9.

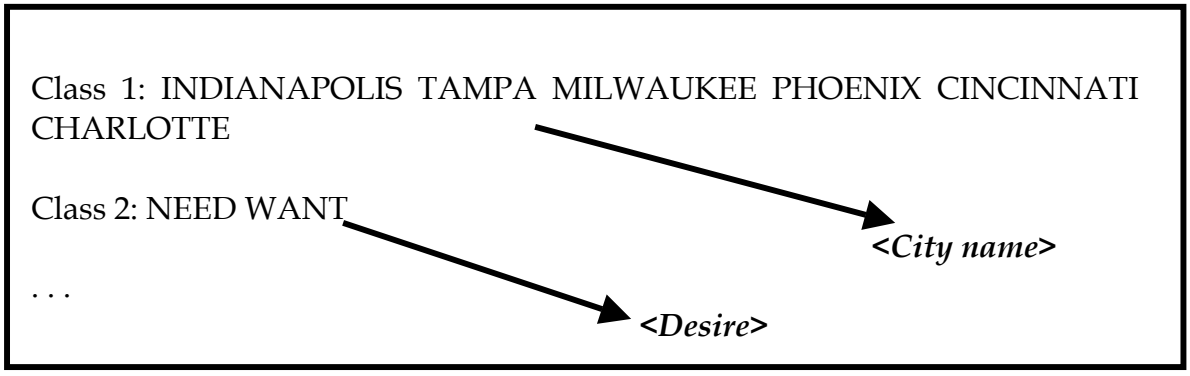


Figure 4.9: The format of output of the proposed clustering algorithm.

The derived clusters are given a label, which indicates the current number of the class, since there are no available sources for semantic labeling (the actual label we used in our work is of the form "cx^1classx^2c", where x^1 indicates the number of system's iteration and x^2 indicates the number of class). This fact underlines the nature of our work for automatic induction of semantic classes. Class 1 can be viewed as set of lexical units, which belong to the concept of "city name". Similarly, Class 2 expresses, in some sense, the general concept of "desire".

The clustering algorithm may group wrongly a lexical unit in cluster due to step 3(ii). That happens usually when the algorithm examines a pair that is positioned away from the top of the output list.

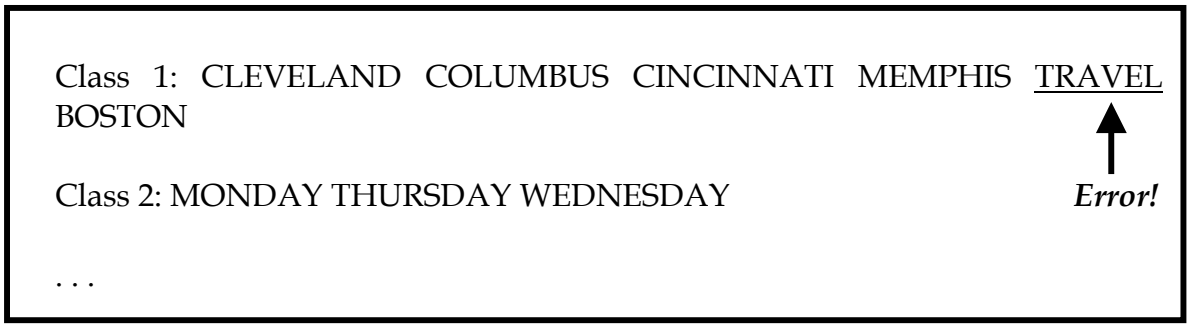


Figure 4.10: An example of wrong clustering

Figure 4.10 illustrates an example of wrong clustering. Obviously the word "TRAVEL" does not belong semantically to the semantic class "city name".

"Trust only your neighbors"- Definition of search margin: While we are constructing a cluster we can limit our search to the nearest pairs. This means that the clustering algorithm will examine the candidate pairs, which are within a predefined range of list's lower positions, from the last grouped word's position. We call this range "search margin". The next example shows how this range affects the construction of clusters.

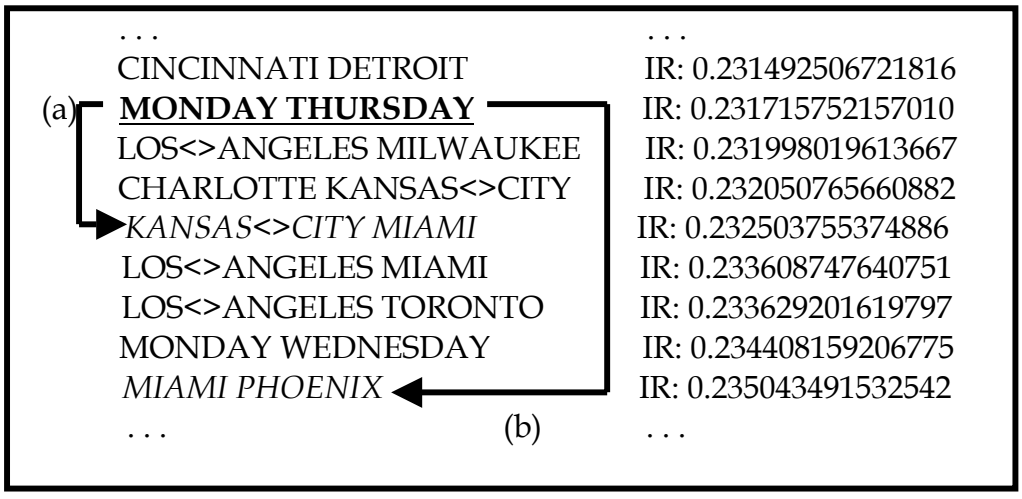


Figure 4.11: Constructing the class that corresponds to the concept of "day name". In case (a) search margin = 3. In case (b) search margin = 7.

Assume that the clustering algorithm examines the pair "MONDAY THURSDAY" presented in Figure 4.11 and executes step 3(i). A new class is created consisting of "MONDAY" and "THURSDAY". In case (a), the

algorithm is allowed to examine the lower part of the list, searching up to the pair "KANSAS<>CITY MIAMI". Thus the resulting class is not changed. In case (b) the search margin is extended up to the pair "MIAMI PHOENIX". This extension adds the word "WEDNESDAY" to the class, since step 3(ii) is executed. This constraint favors the precision of our system but decreases the system's recall. In the next chapter we present several quantitative results concerning the impact of search margin to system's performance.

The described algorithm is based only in a simple observation regarding the same member that two pairs may have. The quantity and quality of the resulting classes depends on the size of the list. Generally the pairs, which are near to the top of the list, are composed of lexical units that are strongly related. In contrast, the lower pairs in the list have more poor semantic relationship. The balancing art is to have a large list of pairs in order to derive enough semantic classes, preserving at the same time their semantic homogeny. To achieve this goal, six clusters were generated per iteration.

"Check the selected neighbors": In order to preserve the semantic homogeny of the derived classes, especially when the search margin is large, we adopted a criterion that removes the dissimilar members from each class. It is described in the next paragraph 4.4.3.

4.4.3 Checking the members of each class

An important issue is to find a criterion to filter out the "idle lexical units", which, for some reason, are included in an almost homogenous derived class. The term "idle lexical unit" refers to a lexical unit, which has not strong semantic relationship with the other members of the same class. This idea preserves the semantic compactness of the semantic classes. So, the criterion, which provides the entry of a lexical unit to the semantic class, must ensure that the lexical unit is located near to the semantic centroid of the class. A way to form such a criterion for each member of the class is to find its semantic distance from the average semantic score of the other class members. If this distance is greater than a defined threshold, then the member must be removed from the class.

Assume a semantic class $C = \{ x_1, x_2, \ldots, x_n \}$.
For each $i = 1 \ldots n$ we calculate:

$$D_1 = \frac{\sum_{k=1}^{n} \sum_{l=k+1}^{n} dist(x_k\ x_l)}{½n(n-1)} \qquad (4.2)$$

$$D_2 = \frac{\sum_{k=1}^{n} dist(x_k\ x_i\ )}{(n\text{-}1)} \qquad (4.3)$$

The *dist* function denotes the semantic distance between x_k and x_i, which is calculated by the measure used to construct the list of pairs. It is obvious that the distance when $k = i$, equals to zero because is the distance between two instances of the same lexical unit.

For example D_1 and D_2 for x_3, which is included in class $C_1 = \{ x_1, x_2, x_3, x_4, x_5 \}$, are:

$D_1 = [(dist(x_1\ x_2)+dist(x_1\ x_4)+dist(x_1\ x_5)) + (dist(x_2\ x_4)+dist(x_2\ x_5)) + dist(x_4\ x_5)] / 10$
and
$D_2 = (dist(x_1\ x_3)+dist(x_2\ x_3)+dist(x_4\ x_3)+(dist(x_5\ x_3)) / 4$

An intuitive interpretation of *D*'s: D_1 provides a relative average estimation about the semantic closeness of all class' members with x_3 to be excluded. Additionally, D_2 gives an average estimation regarding the semantic distance between x_3 and the residue members. In the following paragraphs we will see how we can combines the *D* values in order to build a compact background for our system.

When the KL, IR and MN are used as similarity measures, a member is removed from a class if $D_2 > m\ D_1$.
When the VP is used as a similarity measure, a member is removed from a class if $D_1 > m\ D_2$.

Experiments showed that satisfactory results are achieved if constant *m* ranges around 5.

4.4.4 Checking the compactness of each class

Another critical issue is the estimation of the quality of each class if it is viewed as a core that consists of several semantically related lexical units. Intuitively this can be compared to the atom's structure. It is important to guarantee tight bonds between the lexical units in order to have classes with semantically compact core.

For each class having *n* members, included in a set, *S*, we define a compactness rate, $Comp._{rate}$:

$$Comp._{rate} = \frac{\sum_{i \in S} D_{1(i)}}{\sum_{i \in S} D_{2(i)}} \qquad (4.4)$$

The compactness rate for each class is the rate of the sum of members' D_1 to the sum of members' D_2. D_1 and D_2 are computed by equations (4.2) and (4.3), respectively.

This rate provides a relative estimation about the semantic compactness of the class. Our experimental experience suggests that classes with $Comp._{rate} > 1=p$ must be ignored.

4.4.5 Merging classes

Due to the nature of the clustering algorithm it is possible some lexical units that express the same concept, to be clustered in distinct classes. For example consider the following classes that were induced after the first operation:

c1class1c*LAYOVER STOPOVER
c1class2c*EARLIEST LATEST
c1class3c*FIND RENT
c1class4c*ALSO BE
c1class5c* INDIANAPOLIS TAMPA MILWAUKEE PHOENIX CINCINNATI
c1class6c* CHARLOTTE PITTSBURGH DALLAS MINNEAPOLIS

It is obvious that "c1class5c" and "c1class6c" express the same concept. Thus, it is more reasonable to merge these classes before the next iteration. In general the merging of classes, which were induced during the same iteration, provides some advantages:

(a) it is provided a more precise, realistic representation of the corresponding meaning since the lexical units that express the same meaning are not included in discrete groups.
(b) the merging eliminates the need of computation of the similarity between the merged classes at the next iteration.
(c) considering the first advantage, the statistics that are calculated at the beginning of the next iteration are more sufficient. So, during the next iteration, the chunker and the semantic generalizer can perform computations using more reliable data.

The key idea for merging classes deals with measuring a similarity distance between them. The distance between two classes, $D_{1,2}$, is defined to be the average distance between their members.

Suppose that we have the classes: *c1class1c* = $\{x_1, x_2\}$ and *c1class2c* = $\{y_1, y_2, y_3\}$. The distance between them is calculated as:

$$D_{1,2} = [(dist(x_1 y_1)+dist(x_1 y_2)+dist(x_1 y_3)) + (dist(x_2 y_1)+dist(x_2 y_2)+dist(x_3 y_3))] / 6$$

The *dist* function is the dissimilarity/similarity measure, which is used by the semantic generalizer.

Now, we are able to define a rate, $Merge_{thr.}$, for merging classes. We will use the distance D_2, introduced in equation (4.3). $D_{2(i)}$ characterize every induced class, *i*, indicating, in some sense, the semantic density of its members. It is calculated as the sum of its members' D_2 divided by their count. This threshold is defined as:

$$Merge_{thr.} = \frac{1}{2} \frac{D_{2(1)} + D_{2(2)}}{D_{1,2}} \qquad (4.5)$$

Through multiple experimental tests, we concluded that the appropriate merging is taken place between classes if they have $Merge_{thr.}$ greater than (0.7=c) + *q*. The *q* value is a constant that is depended on the changing of the N-gram statistics during system's iterations. Initially equals to zero. We found that reasonable results are obtained if it is incremented by 0.075 per iteration.

In our example, classes "c1class5c" and "c1class6c" were appeared to have $Merge_{thr.} > 0.7 + 0$. Consequently their members were grouped into a single cluster, forming a new class:

c1class(56)7c*INDIANAPOLIS TAMPA MILWAUKEE PHOENIX CINCINNATI CHARLOTTE PITTSBURGH DALLAS MINNEAPOLIS.

The numbers, which are between parentheses, indicate the participating classes. These classes are not considered anymore during the remaining procedure. The eventual set of induced classes is presented in the next paragraph.

4.4.6 After the end of first operation: what follows

Here are the final results generated by the semantic generalizer's first operation:

c1class1c*LAYOVER STOPOVER
c1class2c*EARLIEST LATEST
c1class3c*FIND RENT
c1class4c*ALSO BE

c1class(56)7c*INDIANAPOLIS TAMPA MILWAUKEE PHOENIX CINCINNATI CHARLOTTE PITTSBURGH DALLAS MINNEAPOLIS

Clearly the amount of induced semantic classes and their members is not sufficient. Consequently the corpus must be reparsed and the whole experimental procedure must be repeated. This operation is performed by the third component, as is shown in Figure 4.1.

4.5 Reparsing the corpus

The corpus is reparsed after the semantic generalization (induction of semantic classes). All instances (members) of each of the classes are replaced with the appropriate class. Let's summarize the gradient transformations that are taken place in a sentence included the ATIS corpus, up to this point (until the end of the first operation of the semantic generalizer):

1. The original sentence taken from the corpus is:
"I WOULD LIKE TO FIND A FLIGHT FROM CHARLOTTE TO LAS VEGAS THAT MAKES A STOP IN SAINT LOUIS"
2. The lexical phraser chunked the words: LAS,VEGAS and SAINT,LOUIS. So the referred sentence is changed to:
"I WOULD<>LIKE TO FIND A FLIGHT FROM CHARLOTTE TO LAS<>VEGAS THAT MAKES A STOP IN SAINT<>LOUIS"
3. The semantic generalizer induced the following classes:
c1class1c*EARLIEST LATEST
c1class2c*ALSO BE
c1class3c*WANT WOULD<>LIKE
c1class5c*INDIANAPOLIS TAMPA SAINT<>LOUIS PHOENIX KANSAS<>CITY CINCINNATI CHARLOTTE MINNEAPOLIS LAS<>VEGAS c1class6c*BURBANK NASHVILLE
The sentence of step 2 is transformed as follows:
"I c1class3c TO FIND A FLIGHT FROM c1class5c TO c1class5c THAT MAKES A STOP IN c1class5c"

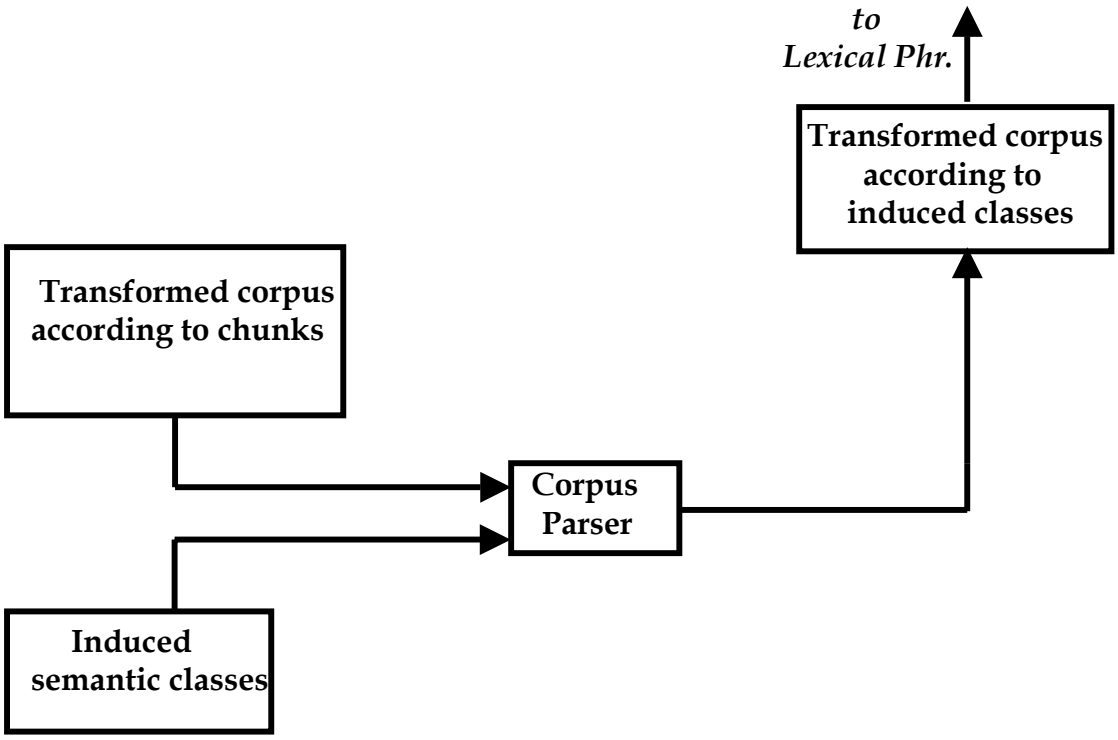


Figure 4.12: The corpus is reparsed after semantic generalization.

As is illustrated in Figure 4.12 the last step is performed by the component of corpus parser. Then, the "updated" corpus is given, as input, to the lexical phraser, as the iterative demands. The chunks that were not classified into semantic groups are negated as single lexical entities and they do not occur such as in the corpus, which is given to the lexical phraser. In other words, the special symbol "<>" is replaced by the white space.

In order to have a specific view of the above statements we present the total induction of semantic classes for five iterations:

c1class1c*EARLIEST LATEST
c1class2c*ALSO BE
c1class3c*WANT WOULD<>LIKE
c1class6c*BURBANK NASHVILLE
c1class(45)7c*INDIANAPOLIS TAMPA MILWAUKEE PHOENIX KANSAS<>CITY CINCINNATI DALLAS MINNEAPOLIS LOS<>ANGELES
c2class1c*COLUMBUS c1class6c CLEVELAND
c2class2c*LAS<>VEGAS MIAMI
c2class3c*LONG<>BEACH ONTARIO
c2class4c*PAUL PETERSBURG
c2class5c*FIND RENT
c2class6c*OAKLAND WESTCHESTER<>COUNTY
c3class1c*TORONTO c2class2c

c3class6c*EIGHTH SIXTH
c3class(45)7c*MONTREAL TORONTO CHARLOTTE DETROIT
c3class(23)8c*CHICAGO DETROIT MEMPHIS TRAVEL c2class3c BOSTON PITTSBURGH NEWARK c2class1c
c4class1c*MONDAY THURSDAY WEDNESDAY
c4class2c*FRANCISCO JOSE
c4class3c*HOUSTON c3class1c SALT<>LAKE<>CITY ORLANDO TACOMA SEATTLE
c4class4c*LOUIS c2class4c
c4class5c*NEED c1class3c
c4class6c*NONSTOP RETURN
c5class1c*BALTIMORE SAINT<>c4class4c ATLANTA
c5class2c*SATURDAY c4class1c
c5class3c*FOUR NINE
c5class4c*U<>S<>AIR c1class2c
c5class5c*EVENING MORNING
c5class6c*CAR LAYOVER

It is obvious that a class can be a member of a posterior class. For example class "c3class1c" has as member the class "c2class2c". This makes "c3class1c" to have totally three members: "LAS<>VEGAS", "MIAMI" and "TORONTO". Also, notice the structure of "SAINT<>c4class4c" which is included in "c5class1c".
If we follow a backward direction we can see that "c4class4c" includes "PAUL", "PETERSBURG" and "LOUIS". Since the word "SAINT" is chunked with "c4class4c", the final members of "c5class1c" are: "SAINT<>PAUL", "SAINT<> PETERSBURG" and "SAINT<>LOUIS".

4.6 The need for evaluation

The number of system's iterations is a critical issue being under discussion, as well as other parameters such as the number of selected chunks. Moreover a comparison between the referred metrics can lead to useful conclusions. The most appropriate way to estimate the optimal values of these parameters is to make several tests evaluating the system's precision and recall each time. This discussion is the main subject of the following chapter.

4.7 Summary

In this chapter we described the several steps for the implementation of our system, which is performing the automatic induction of semantic classes. Our system's development was strongly based on the procedure, which is presented in [5]. The implementation was done, mainly, in Perl. As experimental corpus we used transcribed utterances from the ATIS domain. Next, we used the CMU toolkit in order to build a bigram language model. This model provided the required statistics for the computations that our

system done. The system works iteratively, inducing semantic classes at the end of each iteration, and consists of three main components: (a) a lexical phraser, (b) a semantic generalizer and (c) a corpus parser. The lexical phraser groups words in a single lexical (and concepts) according to their weighted point-wise mutual information. The next component, semantic generalizer, computes one of the four semantic metrics, KL, MN, IR and VP, and generates rules that map words (and concepts) to concepts using a clustering algorithm. This is done because we assumed that words that occur in the same bigram lexical context, have similar semantic meanings. Also we used a context threshold of three. The semantic generalizer, generates six semantic classes per iteration. Moreover, we tried to apply some methods in order to preserve the semantic homogeny of each class and to merge semantically similar classes that were generated during the same iteration. In the final step, a corpus parser reparses the corpus and replaces all the instances of each semantic class with the name of the appropriate class. According to these changes the CMU toolkit computes the new statistics and the whole procedure is repeated, since, as we mentioned earlier, the system works iteratively.

5 Evaluation

The final step in our work is the assessment of the quality of the output of the system we built. That is, we must measure how well our system induced the semantic classes. The answer to this important question can be a useful guide concerning the accurate understanding of several "aspects" of our system. In the next paragraphs we will specify the nature of these aspects.

In early work on NLP, the system builders themselves assessed the quality of the output of the system. If the output seemed good, then the system was judged a success. In general this approach is not very effective. Recent interest has been focused on the rigorous evaluation of NLP systems. Two widely used techniques for evaluation is to use statistical methods or to use the human judgment [1]. In the last case, usually the human judge must be an expert in the domain of interest. For example, in [5], five non-expert, human subjects were used in order to evaluate the quality of semantic classes that were induced by each of the different metrics (KL, IR, MN, VP). They were high-school students employed at Bell Labs for summer practice.

5.1 Train and test data

Our experimental corpus is based on the training and test sets of the ATIS (Air Travel Information System) domain. ATIS is a common task in the ARPA (Advanced Research Project Agency) Speech and Language Program in the USA. Both sets consists of transcribed human queries where callers phoned in order to make flight reservations and retrieve several information about the flights and airports, like the fare types and even if an airport provides limousine service. During the experimental procedure, in the data there were no punctuation marks. Obviously the ATIS domain is specific because the caller uses a limited vocabulary in order to be informed precisely. The train data consists of 1705 sentences (19208 words). The test data were included in a file named AtisTest94 consisting of 445 sentences (5027 words). However, the train data can be used for the task of the evaluation. Also, we must remind that both train and test data were not part-of-speech tagged. Sample sentences for the two sets are presented in the appendix.

5.2 Evaluation using precision and recall

5.2.1 Definition of precision and recall

In our work in order to evaluate the induced semantic classes we used a statistical approach using the notion of "precision" and "recall", through a

Perl script. The system's accuracy is based on how many of the induced classes are actually semantically correct [1]. For the needs of our study it can by assessed by a "precision" measure:

$$Precision = \frac{\#\ of\ correct\ members\ in\ all\ induced\ classes}{\#\ of\ members\ in\ all\ induced\ classes} \qquad (5.1)$$

Additionally, the proportion of the target items that the system selected is defined using a "recall" measure [2]. In our work, "recall" is defined as:

$$Recall = \frac{\#\ of\ correct\ members\ in\ all\ induced\ classes}{\#\ of\ correct\ members\ occured\ in\ test\ text} \qquad (5.2)$$

The above definitions raise a reasonable question: how a "correct member" is defined?

Firstly, we must recall from the previous chapter that an induced semantic class is a group consisting of lexical units without a true conceptual label. The next step is to examine each member of this class and decide if it is "correct" or not. That is, if it belongs to this class or not.

5.2.2 Hand-labeled semantic classes

In order to perform this examination we used a given collection of conceptually hand-labeled semantic classes, constructed for the ATIS domain. Actually this collection is a set of files. Each file's name describes the concept that a class represents. The file's contents are the lexical units that are assumed to belong to the corresponding class. For each induced class we used one labeled semantic class from this collection in order to make the required comparisons.

For example the file named "month" express the concept of *month* and consists of those lexical units that are appropriate for forming the corresponding class: *JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER* and *DECEMBER*.

We used totally 32 files, which cover 32 different concepts. The files' names are presented in the appendix with a representative example of their contents.

An induced class was assumed to express the same concept as the selected labeled class if the majority (>50%) of its elements was occurred also in the labeled class. So, for an induced class satisfying the criterion of majority, a member was considered as "correct" if it was appeared also in the labeled class. If so, the enumerator of equation (5.1) was increased by one. The collection of labeled semantic classes is presented in the Appendix.

The denominator of equation (5.2) is referred to the common lexical units between the collection of labeled semantic classes and the test text.

5.3 Varying parameters

As was mentioned in the previous chapter, our system has many parameters that can take several values. It is an important issue the experimentation with their possible combinations. In this way the cases where the system achieves satisfactory results can be identified. These parameters are summarized in the following table:

Description of the parameter	Parameter's abbreviation
Number of system's iterations	SysIt
Number of chunker's iterations	ChuIt
Number of chunks that are selected at each system's iteration	#Ch
The used semantic metric	SemMs
Number of pairs of semantically related lexical units that are selected at each system's iteration	#Pairs
The constant m used during the removal of a member from a class	m
The value of p in $Comp._{rate}$ computed by equation (4.3) during the checking of the compactness of semantic classes	p
The value of c and q in $Merge_{thr}$ computed by equation (4.4) during the merging of classes	c, q
The value of search margin used in the grouping of semantically similar lexical units	SchMrg

Table 5.1: System's parameters

Obviously there are a lot of possible combinations between the mentioned parameters, which require an extremely long computational time. Our experimental experience provided to us some kind of estimation regarding the range in which we must perform the evaluation for each parameter. Taking these facts into account, in the following paragraphs we tried to evaluate our system using some representative parameter's values that give the best results, which can be achieved.

5.4 Varying the search margin

The main purpose of the following figures is to illustrate the impact of the search margin (*SchMrg*) to the precision and recall. Also useful conclusions can be reached about the optimal number of selected chunks (#*Ch*) and the number of chunker's iterations (*ChuIt*) for each iteration of the system. The numbers contained in the legend denote the number of chunker's iterations.

The following table contains the parameters that were kept constant:

SysIt	SemMs	#Pairs	m	p	c q
11	IR	150	5	1	0.7 0.075

The following plot shows the precision when the search margin is equal to five. This means, for each class, that the clustering algorithm examined the candidate pairs, which were at the five lower positions, from the last grouped word's position.

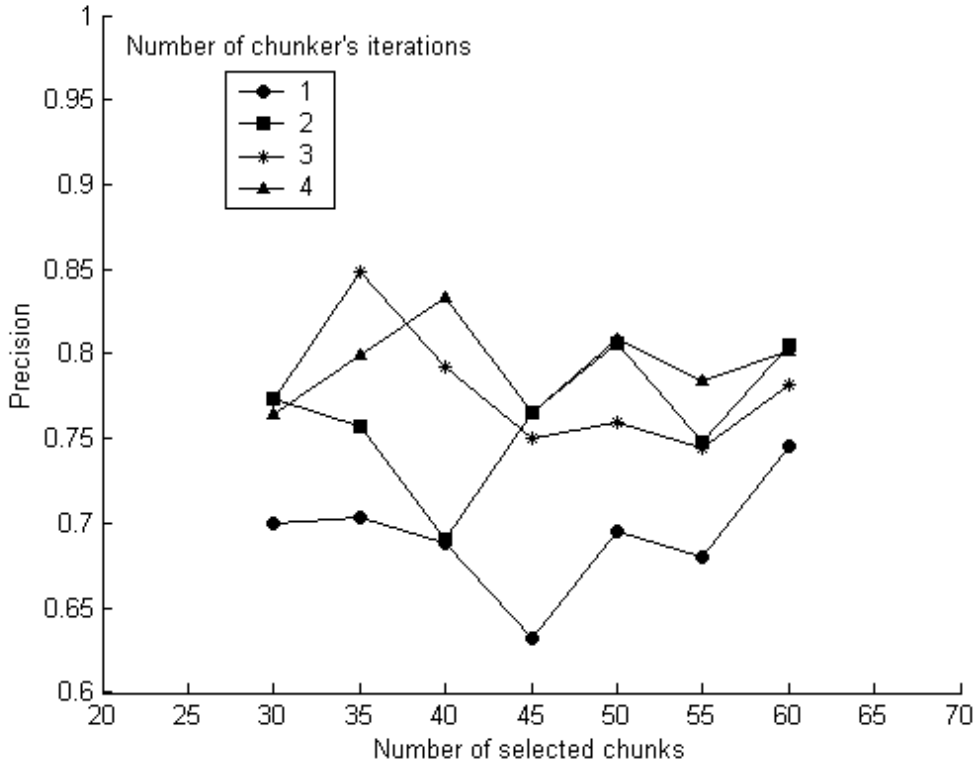


Figure 5.1: Precision for search margin=5

Generally, our experimental experience suggests that this value for the search margin provides sufficient results. We evaluated the results for four different numbers of iterations of the chunker, which are presented in the legend. Moreover, we studied the results for six different numbers of chunks that were kept at each system's iteration. As we can see, 30 chunks were too few and the resulting precision is poor. On the other hand, 35 and 40 chunks per system iteration can increase the precision, which approximates the value of 85%. Contrarily, more than 45 chunks, were too many and probably caused so many nested chunks that entire part of sentences were combined into a single chunk.

This plot can also show the impact of the number of chunker's iterations. It is obvious that only one iteration of the chunker did not manage to form enough chunks. When the chunker iterated twice, the results were better but slightly lower than the desired precision. In contrast, the best results are obtained if we allow three iterations of the chunker.

The following plot illustrates the evaluated recall for the same experiment:

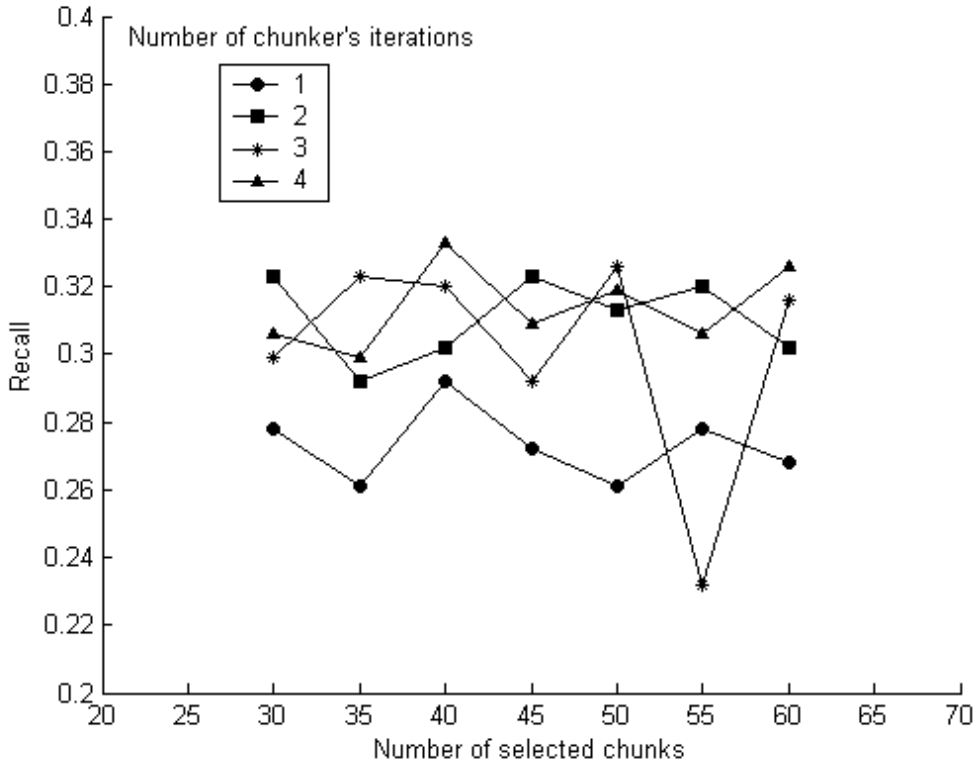


Figure 5.2: Recall for search margin=5

Indeed, one iteration of the chunker gives also poor recall, as is shown in the above plot. As in the case of precision, the highest recall is obtained when we have more than two iterations of the chunker. Concerning the number of

selected chunks per system's iteration, the same situation is repeated as before. That is, 35 and 40 chunks can ensure sufficient recall.

The next plots show the precision and recall for an unbounded search margin. This means that the clustering algorithm has no limits regarding the positions between two pairs, which will be grouped in the same class, while is examining the list of the candidate pairs.

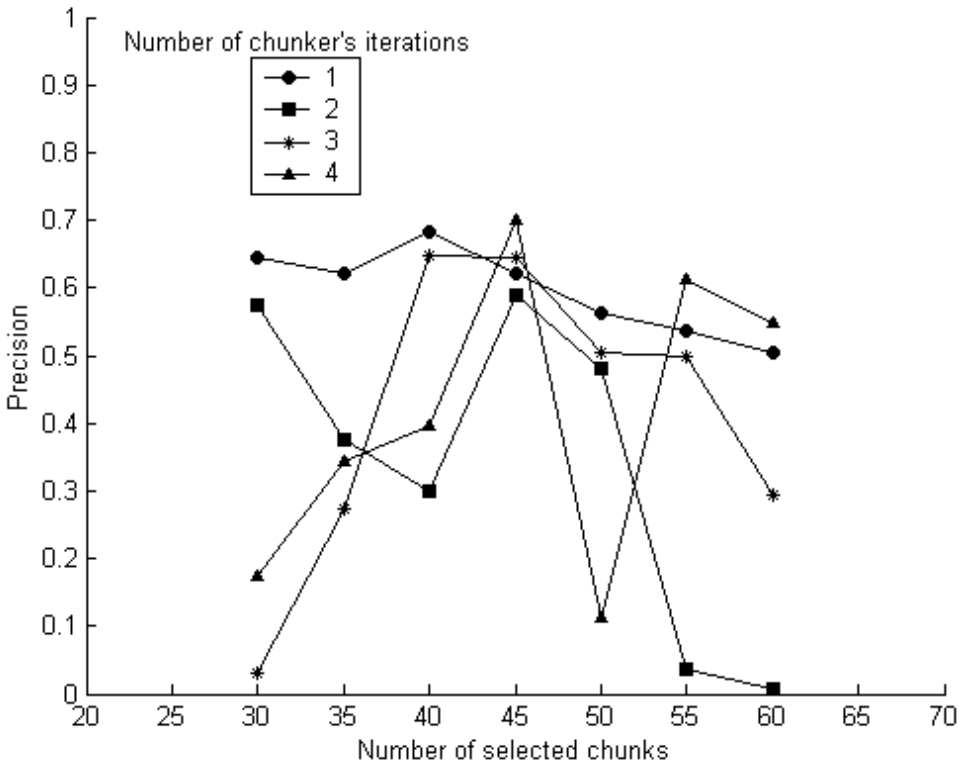


Figure 5.3: Precision for unbounded search margin

Clearly, the unbounded search margin allows the clustering algorithm to group into classes many noisy data. This is fatal situation becomes worst for more than two iterations of the chunker. The presence of those noisy data, grouped in semantic classes, caused the very low, in general, precision and forced the system to have an unstable, unreliable behavior.

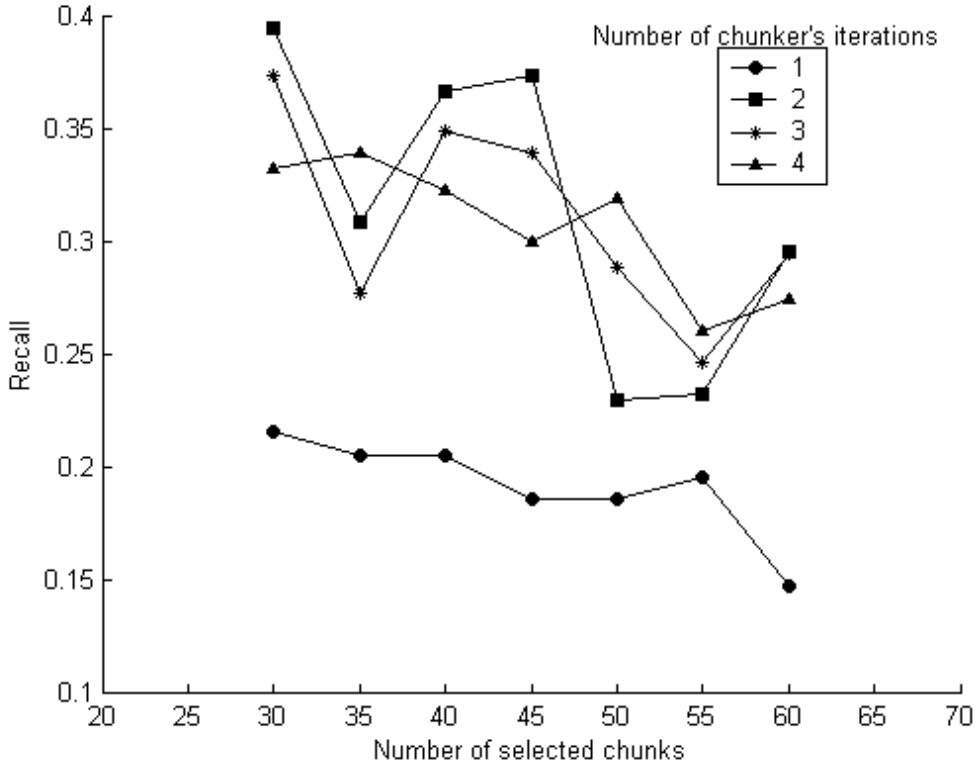


Figure 5.4: Recall for unbounded search margin

The unbounded search margin induced larger, but semantically poorer, classes, giving in many cases recall greater than 35%. This is a desirable result, but combined with the low precision, makes the unbounded search margin inappropriate for our task.

In conclusion, a small value for the search margin can guarantee a stable performance for our system and sufficient precision and recall as well. Additionally we saw that the chunker provides better results if makes more than two iterations, since one or two iterations are not able to form enough chunks for the needs of our system. The best results are obtained for 3 chunker's iterations for each iteration of the system. Lastly, the presented plots suggest that 35 or 40 chunks per system's iteration can give the best possible precision and recall. These conclusions agreed with the corresponding results in [5], where the recommended number of chunks is between 20 and 40.

5.5 Comparison of the four semantic metrics

The main purpose of the following four figures is the comparison of the four semantic metrics with respect to precision and recall. During the evaluation we used five different values for search margin. The zero value denotes the

unbounded search margin. The colors contained in the legend denote precision and recall.

The following table contains the parameters that were kept constant during the experimental procedure:

SysIt	#Pairs	m	p	c q
11	150	5	1	0.7 0.075

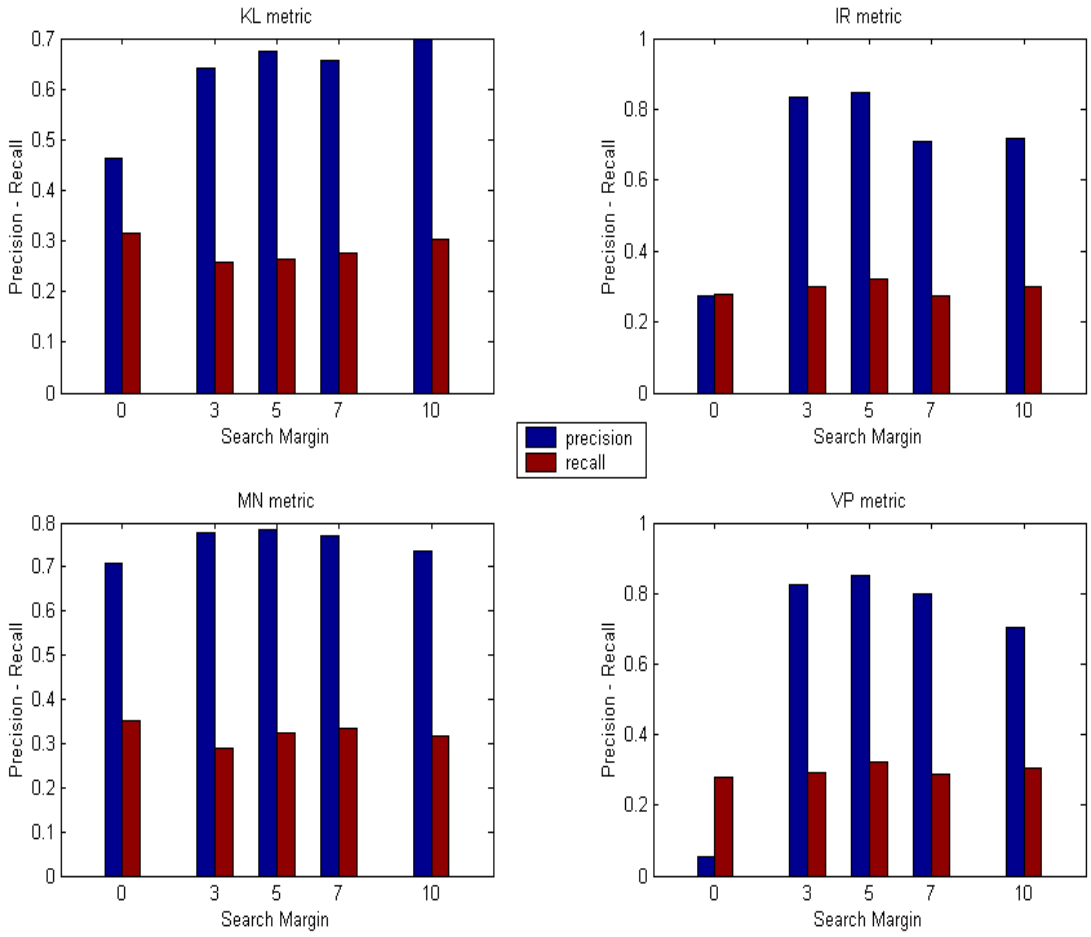


Figure 5.5: Precision and recall for the four metrics

Generally, the bounded metrics perform better than the KL distance, which is unbounded. For example the IR distance is expected to have better performance than the KL, because the denominator for the IR terms is an average of the two probability distributions. Only in the case where *SrcMrg=0* the KL distance gave almost better results compared to the other metrics.

In general, poor precision occur when *SrcMrg=0* because larger classes are constructed and the statistics begin to be worse during the latter iterations of the system.

As is mentioned in [5] about the Travel domain (very similar to ATIS) the KL distance gives the poorest precision. This statement is also verified by our results. The other metrics give qualitatively the same results.

5.6 Merging classes

Recall from 4.4.5 that merging of classes is taking place if the condition $Merge_{thr.} > c+q$ is satisfied.

5.6.1 q value

The *q* value is a variable that is depended on the changing of the N-gram statistics during system's iterations. We found that reasonable results are obtained if it is incremented by a constant value per iteration while it is equal to zero at the first system's iteration. That is needed, because the $Merge_{thr}$ is depended on the semantic distance between the members of the classes which participate in the procedure of merging. The semantic distance between the lexical units generally decreases with increasing the iterations of the system as the closest pairs of lexical units are removed from the corpus. Consequently, this situation raises the need to increase the threshold for the $Merge_{thr}$ by a small amount per iteration, in order to "follow", in some sense, the change of the statistics.

The figure below illustrates the precision and recall for *q=0.025, 0.050, 0.075, 0.1, 0.125, 0.150, 0.175, 0.2, 0.225* and *0.25*.

The following table contains the parameters that were kept constant:

SysIt	ChuIt	SemMs	m	p	c	SchMrg
11	2	IR	5	1	0.7	3

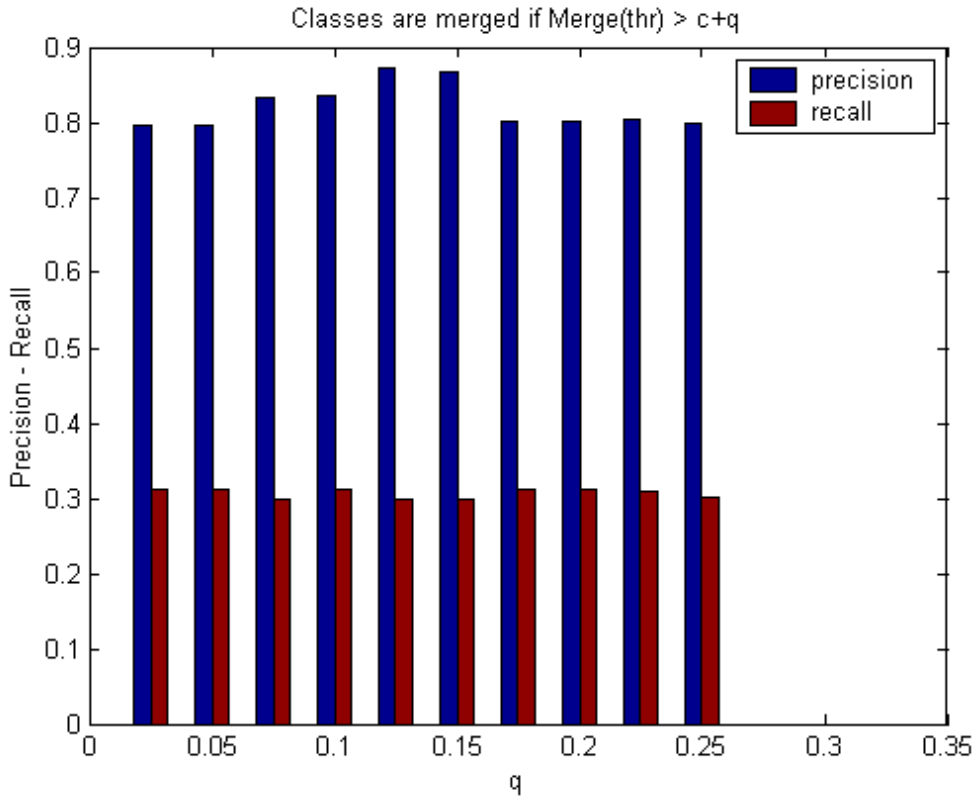


Figure 5.6: Merging classes. The impact of q value

As we can see in the bars above, the increment of the *q* can affect the final results. That is an indication for the changing of the statistics as the system is performing iterations.

Both precision and recall are being favored if the *q* value ranges between 0.075 and 0.125. This parameter seems to be promising since it gives precision greater than 87% for *q=0.125*. Thus, it is important to monitor the changing of the statistics during the iterations in order to tune sufficiently the parameters of the system.

For this experiment the total classes that were formed after merging are few (less than ten). This is an explanation for the limited affect of the *q* to the resulting precision and recall. However, this value may be useful in other domains where the data are more suitable for merging classes.

5.6.2 c value

In contrast, *c* is kept constant during the iterations. In the following figure we present the impact of *c* for five different values: 0.5, 0.6, 0.7, 0.8 and 0.9.

The following table contains the parameters that were kept constant:

SysIt	ChuIt	SemMs	m	p	q	SchMrg
11	2	IR	5	1	0.075	3

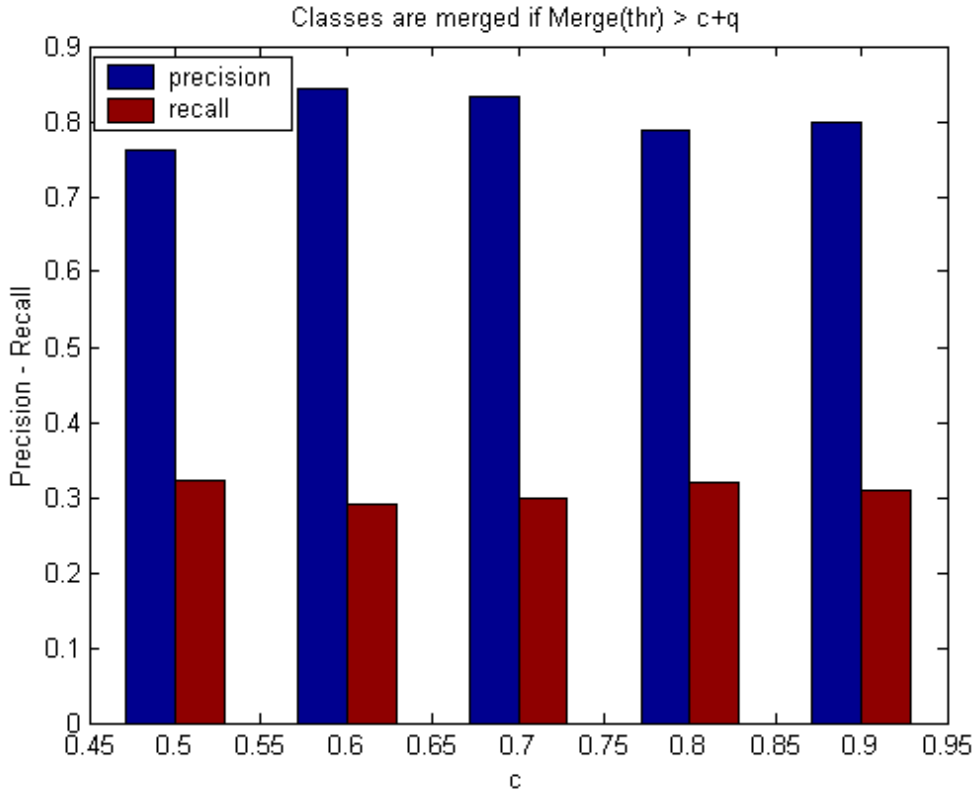


Figure 5.7: Merging classes. The impact of c value

The precision has the highest value when *c=0.6 (84.4%)*. The recall for *c=0.7* is greater compared to the case of *c=0.6*. As in the case of *q* value, the *c* had not a great impact to the results.
So, we can conclude that these values may have a more noticeable affects in semantically broader domains where the need for merging classes is increased. However the studied values for *c* and *q* can serve as useful thresholds for controlling the procedure of merging classes more precisely.

5.7 Checking the compactness of each class

After each iteration every class is being checked regarding its compactness value as was defined in equation (4.4). Through experimentation we found that the results can be slightly better when a class is ignored if $Comp._{rate} > p$. We evaluated the results on ten different values of *p*: 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6 and 1.7.

The following table contains the parameters that were kept constant:

SysIt	ChuIt	SemMs	m	c	q	SchMrg
11	2	IR	5	0.7	0.075	3

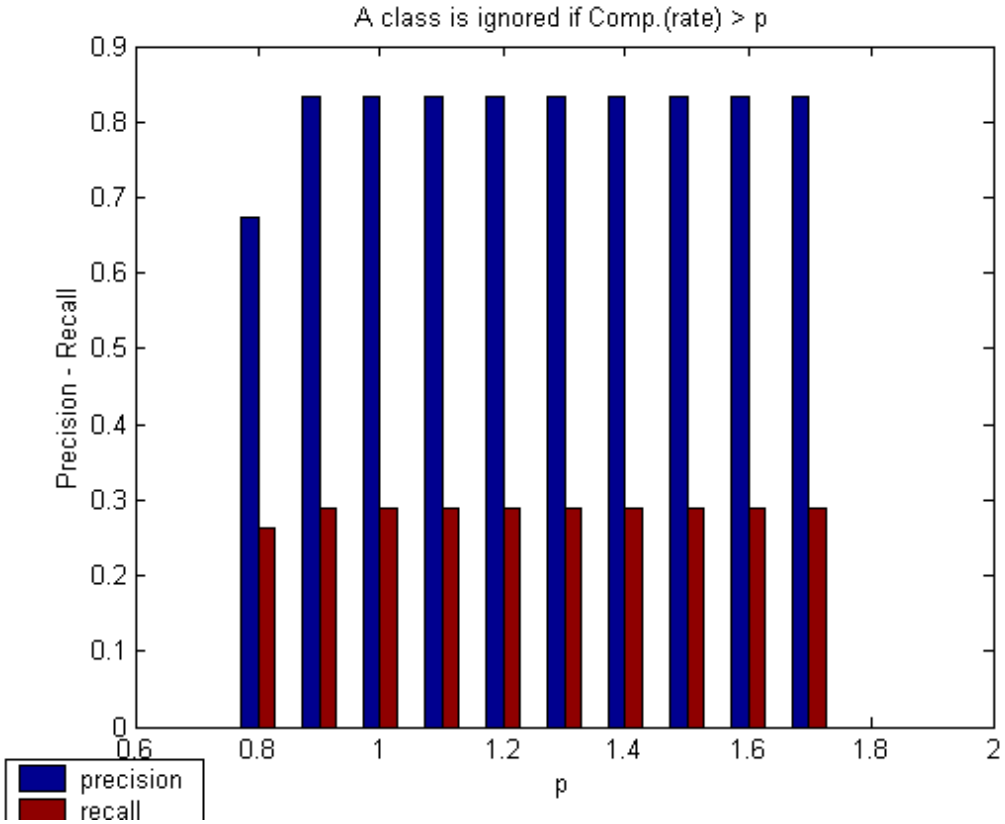


Figure 5.8: Checking the compactness of each class. The impact of p value

We can observe that the value of precision and recall remains constant for $p \geq 0.8$ (83.3% and 28.9% respectively). Despite the fact that values greater than 0.9 are appeared to give same results, it is safer to select for p values near to 0.9. Hence, value such as 0.9 or 1 can behave as a threshold for this purpose.
We must note that this method was considered as an additional constraint concerning the correctness of the results. Its contribution to the overall results cannot be viewed as fundamental in any way. So the role of this idea is actually a kind of support. However, introduces an initial proposal about the estimation of the quality of each class, during the induction of semantic classes at the level of a single system's iteration.

5.8 Removing a member from a class

In section 4.4.3 we introduce the idea of removing a member from a class if it is "located" semantically far from the other members of the class. This idea was an attempt to enhance the semantic homogeny of the induced classes. A

member is removed from a class if $D_2 > m\ D_1$. We evaluated the results on ten different values of *m*: 3, 4, 5, 6, 7, 8, 9, 10, 11and 12.

The following table contains the parameters that were kept constant:

SysIt	ChuIt	SemMs	p	c	q	SchMrg
11	2	IR	1	0.7	0.075	3

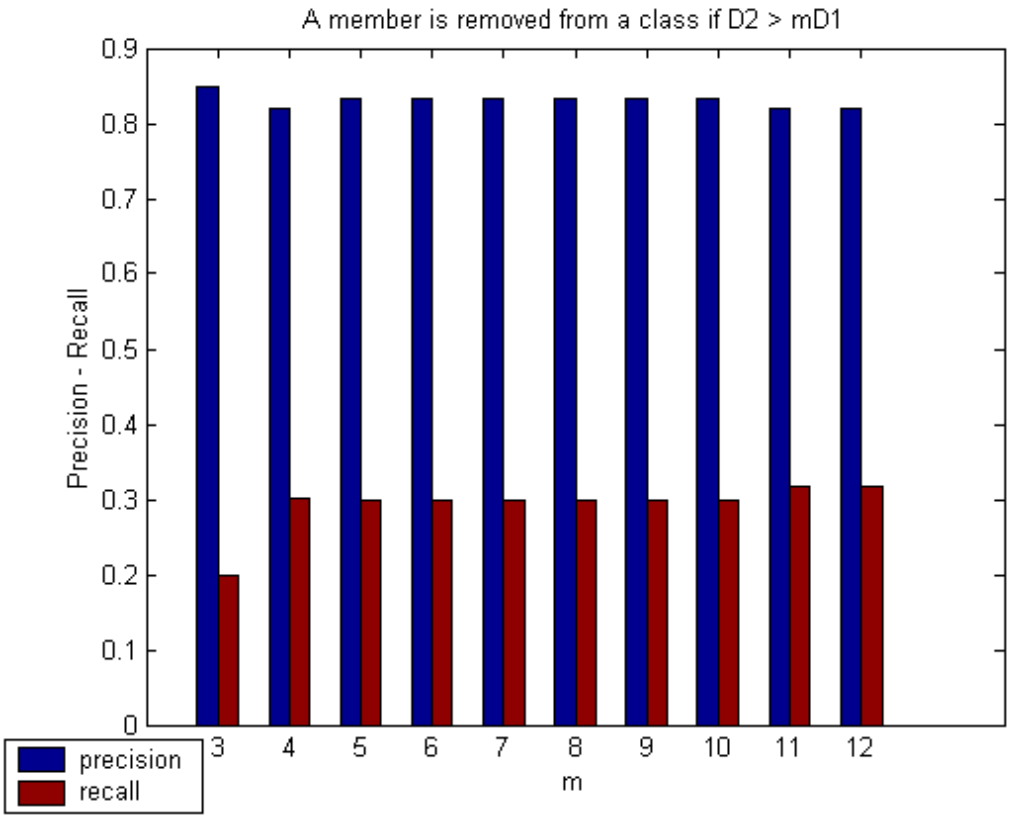


Figure 5.9: The threshold m for removing a member from a class

The results for the last four values of *m* are the same (precision=83.3% and recall=29.9). We observed that the value of 3 was too small, causing the removal of many members. This explains why the precision is higher compared to the next values of *m*. The highest recall is gained for m=4 (30.2%) while the precision equals to 82%. When m ranges from 5 to 10 the results are the same. It is recommended to make a moderate selection of m, avoiding small values (< 3) or values greater than 5. So, good results are ensured if m equals to 4 or 5. The idea of removing a member from a class can be applied further in cases where too many members are grouped into a semantic class. Our experience showed that this method gives interesting results when a class has more than five members. In the cases where a class has less than three members there is no difference.

5.9 How many pairs: introducing a stopping criterion

At each iteration the clustering algorithm operates on a list consisting of semantically related pairs of lexical units. In the previous chapters we mentioned that the length of this list plays a central role regarding the quality of system's results. The selection of a short list favors the precision of our system, while a large list provides greater recall. The following figure illustrates how the length of this list (how many pairs are selected during each iteration, #*Pairs*), affects the average semantic distance of the pairs that are included into the list. Since the IR metric was applied, which is a dissimilarity measure, for each pair we used the inverse of its semantic distance.

The following table contains the parameters that were kept constant:

SysIt	ChuIt	SemMs	m	p	c	q
1	2	IR	5	1	0.7	0.075

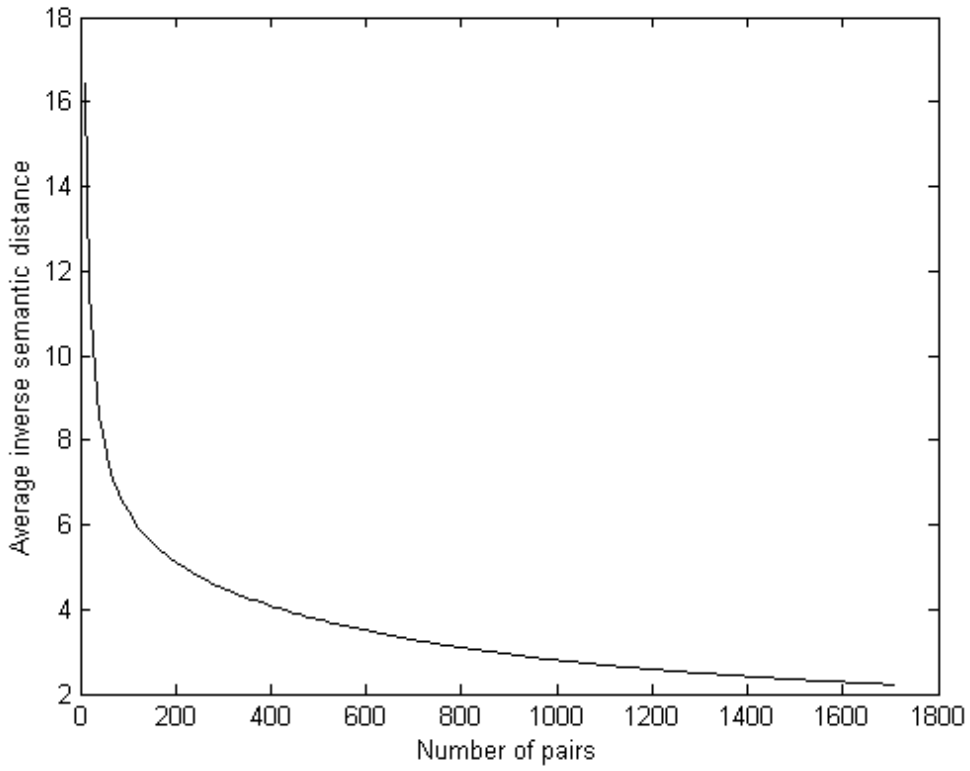


Figure 5.10: The changing of semantic distance according to selected number of pairs

The average similarity between the pairs is decreased as the number of selected pairs grows. It can be said that the average similarity for more than 200 pairs cannot ensure reliable results. Additionally, a very long list of pairs

cannot be accessed up to its lowest pair if a bounded search margin is used. On the other hand less than 100 pairs may be too few in order to achieve sufficient recall. In the most of the cases of our experiments we used 150 pairs; obtaining so, reasonable results.

Automatic selection of the number of pairs for each iteration

The last figure gives an intuitive estimation regarding the optimal number of word pairs. This estimation is essential for each system's iteration. Instead of pre-defining a constant number of pairs for all iterations, we can make, for each iteration, a different estimation for this number. This idea forms a stopping criterion about the number of pairs, which are selected during each iteration. The main characteristic of this technique is that this number is being adjusted to the changing of the statistics during the iterations. Using a more formal expression, we can say that this point, indicating the optimal number of pairs, is located at the point where the derivative of the curve is maximized.

5.10 How many chunks: introducing a stopping criterion

Another critical design issue is the number of chunks that are selected, *(#Ch)*, at each iteration. According to [5], fewer than 10 chunks do not allow to commonly occurring phrases, such as "WOULD LIKE" to be combined. Additionally more than 50 chunks can create many nested chunks. The situation of nested chunking can be harmful for restricted domains, such as ATIS. For two chunker's iterations, the figure that follows shows the average mutual information vs the number of selected chunks.

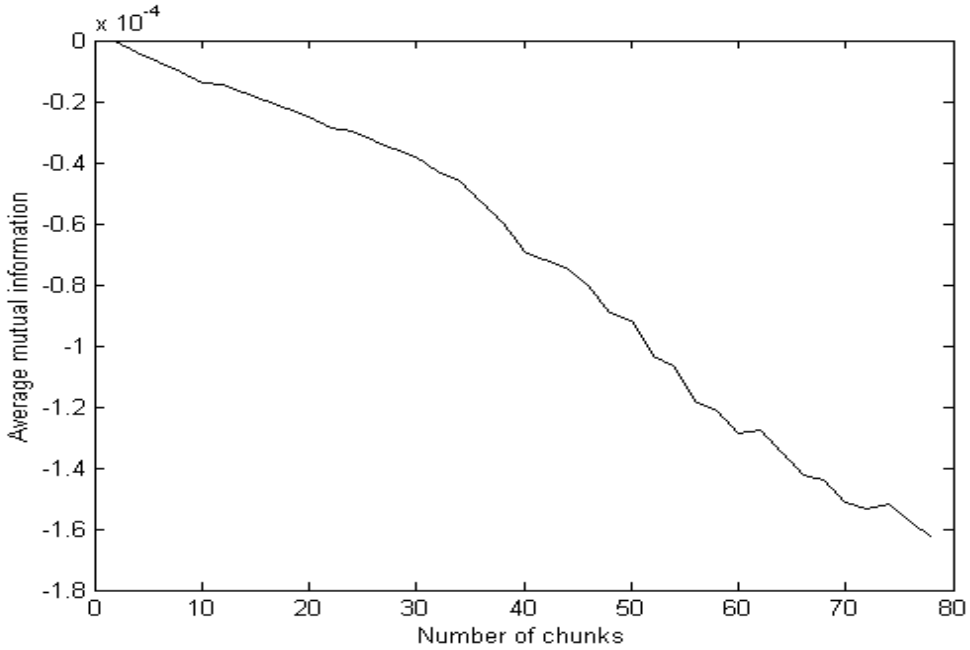


Figure 5.11: The changing of mutual information according to selected number of chunks

Indeed, the difference of mutual information between 30 and 50 chunks is obvious. However, in some cases more than 50 chunks combined with an appropriate search margin can result to sufficient results. Nevertheless, our experimental experience suggests that the number of selected chunks must be greater than 25 and less than 45.

Automatic selection of the number of chunks for each iteration

The same technique can be applied in selecting for each system's iteration the optimal number of chunks. We can define two regions of the curve taking into account its slope. The first region includes, approximately the first forty chunks. Clearly, the average mutual information of the first region is higher compared to the corresponding mutual information of the second region. As before, the point, indicating the optimal number of chunks, is located at the point where the transition from first region to second region is occurred.

5.11 Summary

The nature of our work allows to several parameters' combinations to take place. It would be desired a formal mathematical modeling, with respect to these parameters and some lexical features, such as vocabulary, size of corpus etc. Mainly, our experimental experience was used in order to establish some parameters' values that give sufficient results. We must mention that this experience required a lot of hours of experimentation to be acquired. The presented evaluation sketched a broader framework, in which the performance of the system is "stable". In general, we obtained results quantitatively similar with [5]. We found how much iterations of the chunker can give sufficient results, and we identified a reasonable number of chunks for each iteration of the system. Also we studied the impact of the search margin and we compared the four different semantic metrics.

Additionally we proposed the ideas of merging classes that were induced at the same iteration, removing a member from a class when is semantically "far" from the other members. Lastly we suggested the idea of checking the semantic compactness of each class. These proposals were generated through the needs of our work while we were developing the system. Moreover, we introduced the basis for a stopping criterion about the auto-selection of the optimal number of chunks and pairs at each iteration. Also, the same criterion can be used in order to determine the iterations of the system in order to avoid over-generalization. This issue remains a critical matter under discussion in [5]. We hope that our approach will be a starting point for a deeper study.

Best results and recommended values for the parameters

The search margin is recommended to have a value equal to five. For this value, the highest precision (84.9%) was obtained for 3 chunker's iterations

when 35 chunks were selected per system's iteration. The highest recall (33.3%) was gained for 4 chunker's iterations when 40 chunks were selected per system's iteration. Thus, 3 is a reasonable number for chunker's iterations at each iteration of the system while 35-40 chunks are selected. Also, regarding the four metrics, the IR metric gives the highest precision (84.9%) for *SchMrg=5*. For merging classes the highest precision is given for *q=0.125* (87%). Also, during the same procedure of merging classes the highest precision is achieved for *c=0.6* (84.4%). Checking the compactness of each class the best results are obtained for *p=0.9* (precision=83.3%). In the case of removing a member from a class the best precision and recall are obtained for *m=5* (83.2% and 29.9%)

6 Conclusions and Future Work

6.1 Conclusions

The completion of this work raised several valuable conclusions regarding the nature of human language and how it can be integrated with the probability theory. First of all, we saw that is possible to discover the hidden meaning in the natural language without the support of strict grammatical rules. We used statistical language processing taking into account only the textual context. Based on this consideration we found that the semantic characteristics of text can be identified without the use of a part-of-speech tagger. In contrast the use of a context thresholding of three favored the results by restricting candidate words to those words well represented in a broader lexical content.

Our work was developed according to the proposed procedure, which is presented in [5]. The system was implemented in Perl. Working with Perl, we discovered that it is a powerful programming language and is strongly recommended for rapid prototyping. Perl's integration of regular expressions into the language syntax is particularly powerful for NLP work.

We built a bigram language model using the CMU toolkit. The contribution of this toolkit was invaluable due to its speed and to its options. For the computations of the probabilities, the toolkit used the Witten-Bell discounting combined with backoff.

For the experimental procedure, we used data from the ATIS domain, which is semantically homogeneous. The obtained results showed that the quality of semantic classes is favored if the used corpus is specific. The transcribed utterances of the corpus were short and without sophisticated expressions. This fact was the main reason for the low presence of noisy data and generally made our work easier.

These tools and data served as a fundamental basis for our work. Next, we continued to the implementation of the system. The system consists of three main components: (a) a lexical phraser, (b) a semantic generalizer and (c) a corpus parser. The system worked iteratively inducing six semantic classes per iteration.

We found that the lexical phraser gives sufficient results if it is allowed to make more than two iterations at each iteration of the system. Less than two iterations are not enough in order to form sufficient number of chunks. On the

other hand more than four chunker's iterations can generate too many chunks, causing false substitutions in the corpus. Another critical issue regarding the chunker, was the number of chunks that were selected at each iteration of the system. According to our experimental experience reasonable chunks are obtained for 25 to 40 chunks per system's iteration. More than 50 chunks created so many chunks that entire part of sentences were combined in a single chunk. The proposed number of chunks is similar with the corresponding number suggested in [5].

Moreover, another important subject is the number of pairs that are examined by the clustering algorithm in the semantic generalizer. We found that sufficient results are gained if this number ranges between 150 and 180. We followed a slightly different approach than [5] about the generation of the semantic classes. In [5] each pair is considered as a class and totally five semantic classes are generated at each iteration of the system. In our work, we allowed to the clustering algorithm to generate six classes per iteration, while it is searching the list of pairs according to a search margin. This approach speed up the whole procedure and allows each class to have more than two members per iteration. This method does not perform well for large values of search margin, but if we define a small value for the search margin we can avoid the presence of noise. We found that a search margin less than 6 can give good results.

The use of the four different semantic metrics showed that the bounded metrics in general have better performance than the unbounded metric. The KL distance is an unbounded metric and gave poor results compared to the other metrics. The best results were obtained by the IR and MN metrics. The same conclusions are also stated in [5]. Also, despite the fact that we used different method than [5] in order to evaluate the precision of the results, we achieved in a lot of cases precision greater than 80% while the corresponding precision in [5] is equal to 78.3%. Lastly, all the induced semantic classes, after 11 iterations of the system, had 125 members, approximately, while in [5] the number of members is equal to 110.

We tried to introduce some additional constraints in order to improve the quality of the results, despite the fact that they are not mentioned in [5]. These constraints were implemented by three methods: removal of "idle" members from classes, merging of classes and checking the compactness of each class. These ideas did not improve remarkably the precision and the recall, but we believe that if they applied in a semantically broader domain, they will show greater positive impact.

Finally we introduced the idea of a stopping criterion. This idea is also suggested in [5] and is about the number of iterations of the system. We found that this criterion can be extended in order to determine, in addition, the optimal number of chunks and pairs that at selected at each iteration. We

suggest that this can be found by studying the changing of mutual information and the used semantic distance during the iterations. Our goal is to build robust mathematical model for this criterion by approximating the curve that describes this changing.

6.2 Future work

Our work is based in a considerable mathematical background combined with several empirical estimations that were acquired during the experimental procedure. This fact allows to several alternative approaches to take place concerning the used measures of similarity as well as for the empirical estimations about some constants. Also, preprocessing methods can be applied to the given corpus.

6.2.1 Preprocessing

In [10] the notion of mutual information is adopted in order to define the proposed measure that estimates word association norms. In our work, the lexical phraser is the responsible module for this task. It uses the given corpus and the derived statistics by the CMU toolkit. But, the given corpus is used directly without any preprocessing. The idea for preprocessing is noticeable in [10] and is divided in two parts:

Preprocessing with a parser: Hindle has found it helpful to preprocess the corpus with the Fidditch parser to identify associations between verbs and arguments [15]. For example, consider the triple subject/verb/object. Hindle computed the mutual information between a verb and its object by counting how often the verb and its object were found in the same triple and dividing by chance.
According to [2] this parser is an example of a broader kind of parsing, called "partial parsing". Partial parsing can refer to various levels of detail of syntactic analysis. The simplest partial parses are limited to finding the noun phrases of a sentence. More sophisticated approaches assign grammatical functions to noun phrases and give partial information on attachments, for example, " this noun phrase is attached to another phrase to he right". The Fidditch parser does not use any tagger because it predates the widespread availability of taggers. Moreover, in [10] is suggested that, in order to measure syntactic constraints, it may be useful to include some part of speech information.
Preprocessing with a part of speech tagger: The widespread use of tagging is founded on the belief that many NLP applications will benefit from syntactically disambiguated text. Given this ultimate motivation for part-of-speech tagging, it is surprising that there seem to be more papers on stand-alone tagging than on applying tagging to a task of immediate interest [2].

Church and Hanks in [10] found that interesting contrasts were identified between verbs associated with a following preposition *to/in* and verbs associated with a following infinitive marker *to/to*, if every word in corpus was tagged with a part of speech.

Most applications require partial tagging after part of speech tagging. However, in many systems, a partial parser is built on the top of a tagger. In these cases, the partial tagging is accomplished by way of regular expression matching over the output of the tagger. An important use of tagging in conjunction with partial parsing is for lexical acquisition.

On the other hand, in [5] the precision of induced classes is appeared to be independent of the POS, in the cases where extant thresholds for bigram and trigram context were required. Despite this fact, the combined use of corpus' preprocessing and context thresholding, seems to be promising.

6.2.2 Migrating from bigrams to trigrams

In [5] is mentioned that there are some cases where the bigram context is too local to capture semantic similarity. This phenomenon was very common in the case of WSJ, which is a large, homogeneous corpus. In the case of ATIS corpus, a specific domain, this phenomenon was observed in the case of pair {airport, seventeenth} in the "noun cluster". For these words the most common lexical context was: {… the (airport or seventeenth) </s>}. The use of trigram context instead of bigrams context was proposed in order to improve the precision of auto-induced classes. At best, the trigram statistics only improved performance by a few percent. It is likely that the used WSJ was too small and the extant trigram counts were too few to make much of a difference. We conclude that the use of trigram statistics gives qualitative results. However, it is recommended to exploit the trigrams statistics in the future, if our training corpus is large enough.

6.2.3 Other measures of word similarity

Many different statistical tests have been proposed to measure the strength of word similarity in natural language texts. These tests attempt to measure dependence between words by using statistics taken from a corpus. Through this work we have made use of four different measures of similarity: Kullback-Leibler, Information-radius, L_1 norm and Vector product. However, many other measures for word similarities exist. Tan et al. in [12] present a comparative study with 21 different measures. We suspect that the experimentation with some other measures would be challenging, at least.

6.2.4 Improvements on clustering algorithm

Soft clustering

Indubitably, the clustering algorithm plays a central role concerning the form of the induced classes. As was described, a number of classes are derived per iteration. Each lexical unit is a member of a specific class. That is, it is not allowed to participate in any other class. This kind of hard clustering, at first glance, seems to make our life easier… But consider the following examples, focusing on the ambiguous predicate *SERVE*:

" WHICH AIRLINES SERVE DENVER?"
"WHICH ONES SERVE BREAKFAST?"

Our purpose here does not deal with word sense disambiguation. We only want to show that a certain lexical unit may belong to more than one semantic class. Thus, an attempt of soft clustering may lead to an enriched set of semantic classes, favoring the recall of our system.

Merging classes from different iterations

We saw that the system can merge two classes, if a condition is satisfied. This holds for classes that were induced during the same iteration. However, many times in later iterations, the system induces new classes that could be merged with previously induced classes. So, it is desired the availability to merge classes induced in different iterations, since the final classes would be broader and closer to the representation of real world.

Putting ideas into practice: The ideas presented in 6.4.1 and 6.4.2 can sketch an extremely promising framework for future improvement. We can combine these ideas while we are trying to minimize a criterion, like perplexity [9]. On the other hand, we must note that such ambitious attempt requires a lot of processing time (counted, probably, in terms of days).

6.2.5 Porting across domains

The design of dialogue systems for a new domain requires semantic classes to be identified and defined. This process could be made easier by importing relevant concepts from previously studied domains to the new one. In [5] and [25] two methodologies are proposed, based on comparison of semantic classes across domains, for determining which concepts are domain-independent, and which are specific to the new task. The proposed concept-comparison technique uses a context-dependent distance measurement (i.e.

KL) to compare all pairwise combinations of semantic classes, one from each domain. The proposed concept-projection method uses a similar metric to project a single semantic class from one domain into the lexical environment of another.

In general, semantic classes, developed for well-studied domains, could be used for a new domain with little modification.

6.2.6 Stemming

Stemming is used especially in Information Retrieval (IR) tasks in which a user needs some information, and is looking for relevant documents [1]. The user's information need is represented by a query or profile, and contains one or more search terms, plus perhaps some additional information such importance weights. Hence, the retrieval decision is made, by comparing the terms of the query with the index terms (important words or phrases) appearing in the document itself. The decision may be binary (retrieve/reject), or it may involve estimating the degree of relevance that the document has to the query. Unfortunately, the words that appear in documents and in queries often have many morphological variants. Thus, pairs of terms such as "foxes" and "fox" will not be recognized as equivalent without some form of natural language processing.

In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of IR applications. For this reason, a number of so-called stemming Algorithms, or stemmers, have been developed, which attempt to reduce a word to its stem or root form. Thus, the key terms of a query or document are represented by stems rather than by the original words. This not only means that different variants of a term can be conflated to a single representative form – it also reduces the dictionary size, that is, the number of distinct terms needed for representing a set of documents. A smaller dictionary size results in a saving of storage space and processing time. For IR purposes, it doesn't usually matter whether the stems generated are genuine words or not – thus, "fox" might be stemmed to "fox" – provided that (a) different words with the same 'base meaning' are conflated to the same form, and (b) words with distinct meanings are kept separate.

Examples of products using stemming algorithms would be search engines such as Yahoo and Google, and also thesauruses and other products using NLP for the purpose of IR.

Regarding our work, the technique of stemming is recommended for cases where the morphological complexity is relatively high. For example, the ATIS domain consists of simple utterances and probably the use of stemming will

not have a great positive impact. In contrast, for corpora like WSJ this technique may become a valuable factor of improvement.

Appendix

A

A brief description of the CMU Statistical Language Modeling Toolkit v2

The Carnegie Mellon Statistical Language Modeling (CMU SLM) Toolkit is a set of UNIX software tools designed to support language modeling. All of the software tools have been written in C.

History

Version 1 of the Carnegie Mellon University Statistical Language Modeling toolkit was written by Roni Rosenfeld, and released in 1994. Version 2 was released in 1996.

How to install the Toolkit

For "big-endian" machines (eg those running HP-UX, SunOS, Solaris) the installation procedure is simple:
change into the src/ directory and type

make install

The executables will then be copied into the bin/ directory, and the library file SLM2.a will be copied into the lib/ directory.

For "little-endian" machines (eg those running Linux, Cygwin) the Makefile must be changed. The Makefile is included in the scr directory. A specific line must be changed, as follows:

```
…
#POSIX_FLAG        = -D_INCLUDE_POSIX_SOURCE
#BYTESWAP_FLAG     = -DSLM_SWAP_BYTES
#FIX_PROT_FLAG  = -D__USE_FIXED_PROTOTYPES__
…
```

"#BYTESWAP_FLAG = -DSLM_SWAP_BYTES" must be changed to
"BYTESWAP_FLAG = -DSLM_SWAP_BYTES":

```
…
#POSIX_FLAG        = -D_INCLUDE_POSIX_SOURCE
BYTESWAP_FLAG = -DSLM_SWAP_BYTES
#FIX_PROT_FLAG  = -D__USE_FIXED_PROTOTYPES__
…
```

Then the Toolkit can be installed as before.

In our work we copied these twelve executables into the directory where we built the language models.

File formats

Text stream: An ASCII file containing text. It may or may not have markers to indicate context cues, and white space can be used freely. (.text)

Word frequency file: An ASCII file containing a list of words, and the number of times that they occurred. This list is not sorted; it will generally be used as the input to wfreq2vocab, which does not require sorted input. (.wfreq)

Word n-gram file: ASCII file containing an alphabetically sorted list of n-tuples of words, along with the number of occurrences (.w3gram, .w4gram etc.)

Vocabulary file: ASCII file containing a list of vocabulary words. Comments may also be included – any line beginning ## is considered a comment. The vocabulary is limited in size to 65535 words. (.vocab)

Context cues file: ASCII file containing the list of words, which are to be considered "context cues". These are words which provide useful context information for the n-grams, but which are not to be predicted by the language model. Typical examples would be <s> and <p>, the begin sentence, and begin paragraph tags. (.ccs)

Id n-gram file: ASCII or binary (by default) file containing a numerically sorted list of n-tuples of numbers, corresponding to the mapping of the word n-grams relative to the vocabulary. Out of vocabulary (OOV) words are mapped to the number 0.(.id3gram.bin, .id4gram.ascii etc.)

Binary language model file: Binary file containing all the n-gram counts, together with discounting information and back-off weights. Can be read by evallm and used to generate word probabilities quickly. (.binlm)

ARPA language model file: ASCII file containing the language model probabilities in ARPA-standard format. (.arpa)

Probability stream: ASCII file containing a list of probabilities (one per line). The probabilities correspond the probability for each word in a specific text stream, with context-cues and OOVs removed. (.fprobs)

Forced back-off file: ASCII file containing a list of vocabulary words from which to enforce back-off, together with either an 'i' or an 'e' to indicate inclusive or exclusive forced back-off respectively. (.fblist)

Discounting strategies

Discounting is the process of replacing the original counts with modified counts so as to redistribute the probability mass from the more commonly observed events to the less frequent and unseen events. If the actual number of occurrences of an event E (such as a bigram or trigram occurrence) is $c(E)$, then the modified count is $d(c(E))c(E)$, where $d(c(E))$ is known as the discount ratio. The CMU Toolkit supports the following discounting strategies:

1. Good Turing discounting
2. Witten Bell discounting
3. Absolute discounting
4. Linear discounting

The tools

1. text2wfreq
Input : Text stream
Output : List of every word which occurred in the text, along with its number of occurrences.

2. wfreq2vocab
Input : A word unigram file, as produced by text2wfreq
Output : A vocabulary file.

3. text2wngram
Input : Text stream
Output : List of every word n-gram which occurred in the text, along with its number of occurrences.

4. text2idngram
Input : Text stream, plus a vocabulary file.
Output : List of every id n-gram which occurred in the text, along with its number of occurrences.

5. ngram2mgram
Input : Either a word n-gram file, or an id n-gram file.
Output : Either a word m-gram file, or an id m-gram file, where m < n.

6. wngram2idngram
Input : Word n-gram file, plus a vocabulary file.
Output : List of every id n-gram which occurred in the text, along with its number of occurrences, in either ASCII or binary format.

7. idngram2stats
Input : An id n-gram file (in either binary (by default) or ASCII (if specified) mode).
Output : A list of the frequency-of-frequencies for each of the 2-grams, … , n-grams, which can enable the user to choose appropriate cut-offs, and to specify appropriate memory requirements with the –spec_num option in idngram2lm.

8. mergeidngram
Input : A set of id n-gram files (in either binary (by default) or ASCII (if specified) format – note that they should all be in the same format, however).

Output : One id n-gram file (in either binary (by default) or ASCII (if specified) format), containing the merged id n-grams from the input files.

9. idngram2lm
Input : An id n-gram file (in either binary (by default) or ASCII (if specified) format), a vocabulary file, and (optionally) a context cues file. Additional command line parameters will specify the cutoffs, the discounting strategy and parameters, etc.
Output : A language model, in either binary format (to be read by evallm), or in ARPA format.

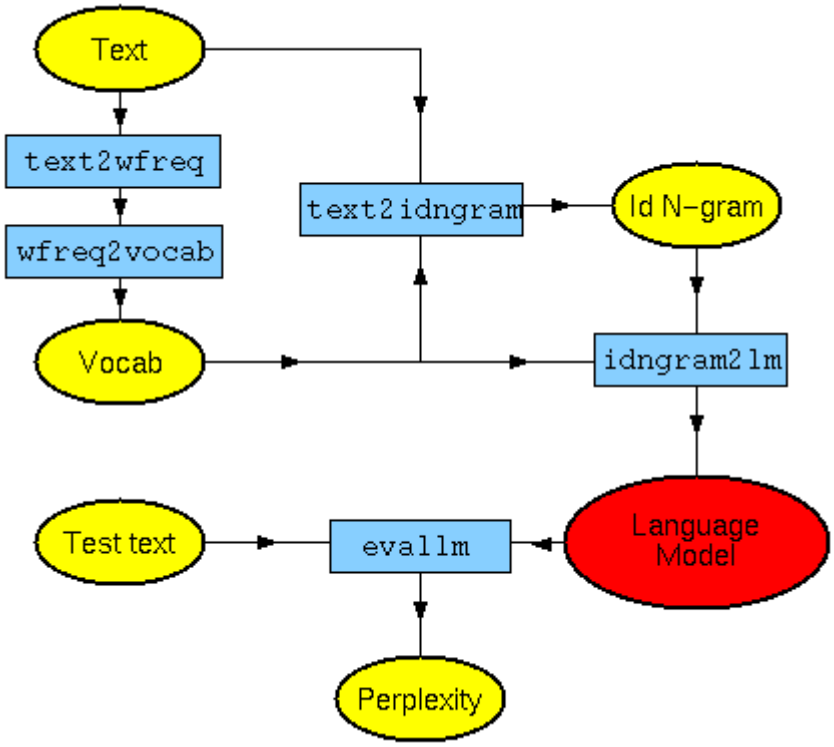
10. binlm2arpa
Input : A binary format language model, as generated by idngram2lm.
Output : An ARPA format language model.

11.evallm
Input : A binary or ARPA format language model, as generated by idngram2lm. In addition, one may also specify a text stream to be used to compute the perplexity of the language model. The ARPA format language model does not contain information as to which words are context cues, so if an ARPA format language model is used, then a context cues file may be specified as well.
Output : The program can run in one of two modes:
compute-PP – Output is the perplexity of the language model with respect to the input text stream.
Validate – Output is confirmation or denial that the sum of the probabilities of each of the words in the context supplied by the user sums to one.

12. interpolate
Input : Files containing probability streams, as generated by the –probs option of the perplexity command of evallm. Alternatively these probabilities could be generated from a separate piece of code, which assigns word probabilities according to some other language model, for example a cache-based LM. This

probability stream can then be linearly interpolated with one from a standard n-gram model using this tool.
Output : An optimal set of interpolation weights for these probability streams, and (optionally) a probability stream corresponding to the linear combination of all the input streams, according to the optimal weights. The optimal weights are calculated using the expectation maximization (EM) algorithm.

Typical usage



The first step in constructing a language model is to define the model's vocabulary:

" cat trainingCorpus.text | text2wfreq | wfreq2vocab –top 20000 > trainingCorpus. vocab "

The option "top" defines the amount of the most common words that the Toolkit uses. In this example the vocabulary file contains the most common 20 000 words.

A context cues files should also be generated:

" echo "<s>" > trainingCorpus .ccs "

Language data is viewed by the CMU Toolkit as a stream of words interspersed with context cues. Context cues are markers, which indicate events such as sentence, paragraph and article boundaries. Useful information is provided to the language model through the use of these markers.

The next step is to convert the training corpus in to a list of id N-grams :

" cat trainingCorpus.text |text2idngram -vocab trainingCorpus.vocab –buffer 200 –n 3 > trainingCorpus.id3gram"

The "-n 3" option means that we are building a trigram model. The "-buffer 200" option defines the amount of RAM ,in MB, that the Toolkit will grab.

The last step is to convert the id trigram stream into an ARPA language model file:

" idngram2lm –witten_bell –n 3 –vocab trainingCorpus.vocab –context trainingCorpus.ccs –idngram trainigCorpus.id3garm -arpa trainingCorpus.arpa –cutoffs 1 1 "

The discounting method used in the last step is the Witten-Bell discounting. Also in order to reduce the size of a language model, infrequent N-grams are often removed from the model. Cutoffs define this reduction. For the 1996 H4 Broadcast News training corpus the size of the language model for cutoffs "0 0" and "1 1", was 219 MB and 80 MB respectively.

Finally, the quality of the language model can be evaluated using a test text. With respect to the test text the perplexity of the language model is computed:

" evallm –arpa trainingCorpus.test

evalm : perplexity –text test.text "

For a more detailed description of the CMU SLM Toolkit it is recommended to consult the complete manual, which is included to the Toolkit.

B

“Scripting” or “Coding”: how to save time

Perl’s integration of regular expressions into the language syntax is particularly powerful for NLP work. While we were writing the code for this work, sometimes the required code’s processing time was surprisingly large. In these cases the first thing we thought, was to translate the script into C, in order to achieve faster execution time. In the following paragraphs we give a brief explanation why this strategy was rejected:

People often have the idea that automatically translating Perl to C and then compiling the C will make their Perl programs run faster, because "C is much faster than Perl."

The Perl interpreter is running your Perl program. You want a C program that does the same thing that your Perl program does. A C program to do what your Perl program does would have to do most of the same things that the Perl interpreter does when it runs your Perl program. There is no reason to think that the C program could do those things faster than the Perl interpreter does them, because the Perl interpreter itself is written in very fast C.

Suppose your program needs to split a line into fields, and uses the Perl split function to do so. You want to compile this to C so it will be faster. This is obviously not going to work, because the split function is already implemented in C. If you have the Perl source code, you can see the implementation of split in the file pp.c; it is in the function named pp_split. When your Perl program uses split, Perl calls this pp_split function to do the splitting. pp_split is written in C, and it has already been compiled to native machine code.

Now, suppose you want to translate your Perl program to C. How will you translate your split call? The only thing you can do is translate it to a call to the C pp_split function, or some other equivalent function that splits. There is no reason to believe that any C implementation of split will be faster than the pp_split that Perl already has. Years of work have gone into making pp_split as fast as possible.

How to save time: Instead of trying to translate your Perl script into C code, it is more sufficient to study in depth your needs and optimize your code (avoid unnecessary “push”es, use hashes in cases where it is absolutely necessary, etc).

C

Sample sentences from the ATIS domain

SHOW ME ALL FLIGHTS FROM TORONTO
SHOW ME MORNING FLIGHTS FROM TORONTO
SHOW ME ALL NATIONAIR FLIGHTS FROM TORONTO
SHOW ME ALL NATIONAIR FLIGHTS FROM TORONTO
SHOW ME ALL CANADIAN AIRLINES FLIGHTS FROM TORONTO
WHAT CITIES ARE SERVED BY CANADIAN AIRLINES INTERNATIONAL
WHERE DOES CANADIAN AIRLINES INTERNATIONAL FLY TO
WHERE DOES CANADIAN AIRLINES INTERNATIONAL FLY
PLEASE LIST THE EARLIEST LUNCH FLIGHT FROM COLUMBUS TO PHOENIX
PLEASE LIST THE FLIGHTS FROM CHARLOTTE TO LONG BEACH ARRIVING AFTER LUNCH TIME
WHAT FLIGHTS ARE AVAILABLE FROM DENVER TO BALTIMORE FIRST CLASS ON UNITED AIRLINES ARRIVING MAY SEVENTH BEFORE NOON
WHAT FLIGHT GOES FROM DENVER TO BALTIMORE FIRST CLASS ON UNITED AIRLINES ARRIVING ON MAY SEVENTH
PLEASE LIST THE CHEAPEST FLIGHTS FROM DALLAS TO BALTIMORE ARRIVING ON MAY FIFTH MAY SEVENTH
PLEASE LIST THE CHEAPEST FLIGHT FROM DALLAS TO BALTIMORE ARRIVING ON MAY SEVENTH
WHAT IS THE SMALLEST AIRCRAFT AVAILABLE FLYING FROM PITTSBURGH TO BALTIMORE ARRIVING ON MAY SEVENTH
WHAT IS THE SMALLEST AIRCRAFT THAT FLIES FROM PITTSBURGH TO BALTIMORE ARRIVING MAY SEVENTH
PLEASE LIST THE FLIGHTS FROM PITTSBURGH TO BALTIMORE ARRIVING MAY SEVENTH
PLEASE LIST ALL THE TAKEOFFS AND LANDINGS FOR GENERAL MITCHELL INTERNATIONAL
PLEASE LIST ALL THE ARRIVING AND DEPARTING FLIGHTS FROM GENERAL MITCHELL INTERNATIONAL
LIST THE FLIGHTS THAT ARRIVE AND DEPART FROM GENERAL MITCHELL INTERNATIONAL AIRPORT
PLEASE LIST THE FLIGHTS TAKING OFF AND LANDING ON GENERAL MITCHELL INTERNATIONAL AIRPORT
WHAT FLIGHTS TAKEOFF AND LAND AT GENERAL MITCHELL INTERNATIONAL
LIST ALL THE FLIGHTS THAT TAKEOFF FROM GENERAL MITCHELL INTERNATIONAL
LIST ALL THE TAKEOFFS AND LANDINGS AT GENERAL MITCHELL AIRPORT
LIST ALL THE TAKEOFFS AND LANDINGS AT GENERAL MITCHELL INTERNATIONAL
LIST THE TAKEOFFS AND LANDINGS AT GENERAL MITCHELL INTERNATIONAL
LIST ALL THE TAKEOFFS AND LANDINGS AT GENERAL MITCHELL INTERNATIONAL
SHOW ME THE ONE WAY FLIGHTS FROM DETROIT TO NEW YORK
SHOW ME THE ONE WAY FLIGHTS FROM DETROIT TO WESTCHESTER COUNTY
SHOW ME THE MOST EXPENSIVE ONE WAY FLIGHT FROM DETROIT TO WESTCHESTER COUNTY
SHOW ME THE ONE WAY FLIGHT FROM DETROIT TO WESTCHESTER COUNTY WITH THE HIGHEST FARE

D

An example of induced semantic classes for 11 iterations

c1class1c*LAYOVER STOPOVER
c1class2c*EARLIEST LATEST
c1class3c*ALSO BE
c1class4c*WANT WOULD<>LIKE
c1class5c*INDIANAPOLIS TAMPA MILWAUKEE PHOENIX KANSAS<>CITY CINCINNATI
c1class6c*FIND RENT
c2class1c*TORONTO c1class5c
c2class2c*DALLAS MINNEAPOLIS LOS<>ANGELES
c2class3c*BURBANK NASHVILLE COLUMBUS CLEVELAND
c2class4c*GOING J<>F<>K
c2class5c*LAS<>VEGAS MIAMI
c2class6c*LONG<>BEACH ONTARIO
c3class1c*PAUL PETERSBURG
c3class2c*OAKLAND WESTCHESTER<>COUNTY
c3class3c*CHICAGO DETROIT
c3class6c*c2class1c c2class5c
c3class(45)7c*MEMPHIS TRAVEL c2class6c BOSTON c2class3c PITTSBURGH c2class2c NEWARK
c4class1c*MONDAY THURSDAY WEDNESDAY
c4class2c*EIGHTH SIXTH
c4class3c*FRANCISCO JOSE
c4class4c*HOUSTON c3class6c SALT<>LAKE<>CITY MONTREAL c2class4c LA<>GUARDIA ORLANDO
c4class5c*CHARLOTTE c3class3c BALTIMORE
c4class6c*LOUIS c3class1c
c5class1c*c4class4c c4class5c
c5class2c*NEED c1class4c
c5class3c*DEPART LEAVE
c5class4c*NONSTOP RETURN
c5class5c*ATLANTA SAINT<>c4class6c TACOMA DENVER
c5class6c*SATURDAY c4class1c
c6class1c*c5class1c c5class5c
c6class2c*EVENING MORNING
c6class3c*FOUR NINE
c6class4c*U<>S<>AIR c1class3c
c6class5c*LOVE<>FIELD c3class2c
c6class6c*EIGHT THREE
c7class1c*ARRIVE ARRIVES
c7class2c*AMERICAN NORTHWEST
c7class3c*FRIDAY SUNDAY c5class6c
c7class4c*CAN DO
c7class5c*DEPARTING LEAVING
c7class6c*SEATTLE c6class1c
c8class1c*PRICES TYPE<>OF<>AIRCRAFT
c8class2c*AFTERNOON c6class2c
c8class3c*BETWEEN FROM
c8class4c*FIVE c6class6c SIX
c8class5c*UNITED c7class2c

c8class6c*AIRFARE TICKET
c9class1c*TUESDAY c7class3c
c9class2c*AIRPORT BACK FLORIDA KNOW
c9class3c*T<>W<>A c6class4c JULY DELTA
c9class4c*CALIFORNIA OHIO
c9class5c*DIEGO c4class3c
c9class6c*TIMES TOMORROW
c10class2c*GIVE<>ME LIST
c10class3c*GET SEE c5class2c
c10class4c*DAILY c5class4c CONNECTING
c10class5c*CAR STOP
c10class(16)7c*L c1class6c TAKE
c11class1c*AND OR
c11class2c*c6class5c c7class6c
c11class3c*DISTANCE FLIGHTS
c11class4c*AFTER BEFORE AROUND
c11class5c*CITIES c8class1c AIRCRAFT
c11class6c*c7class5c c9class3c

E

More figures of evaluation

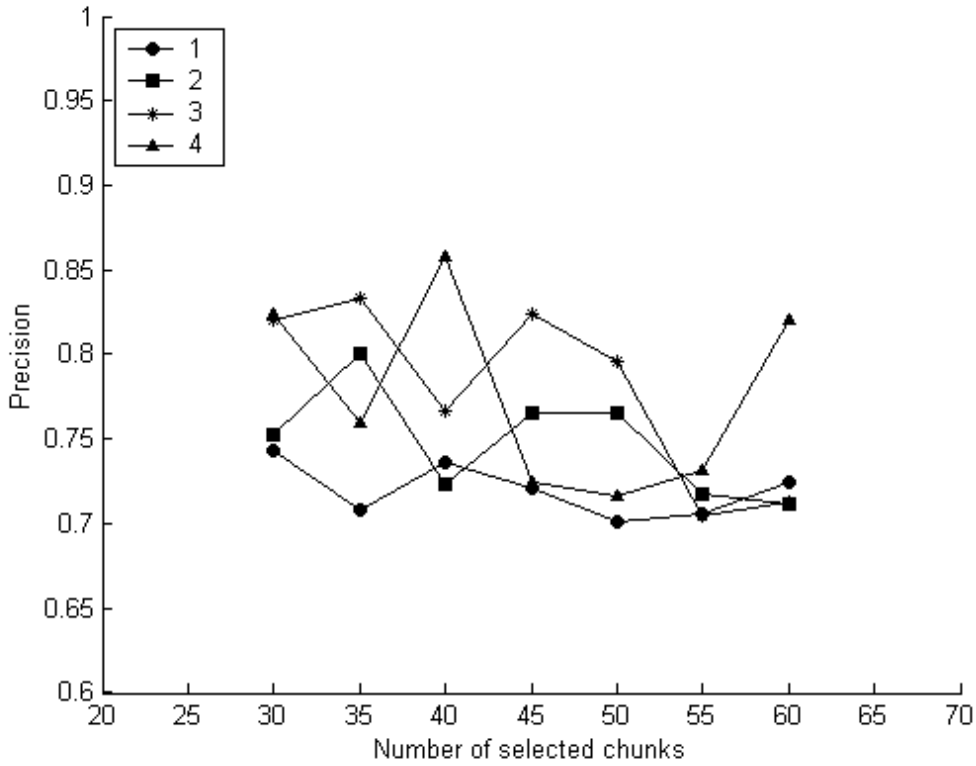
Varying the search margin

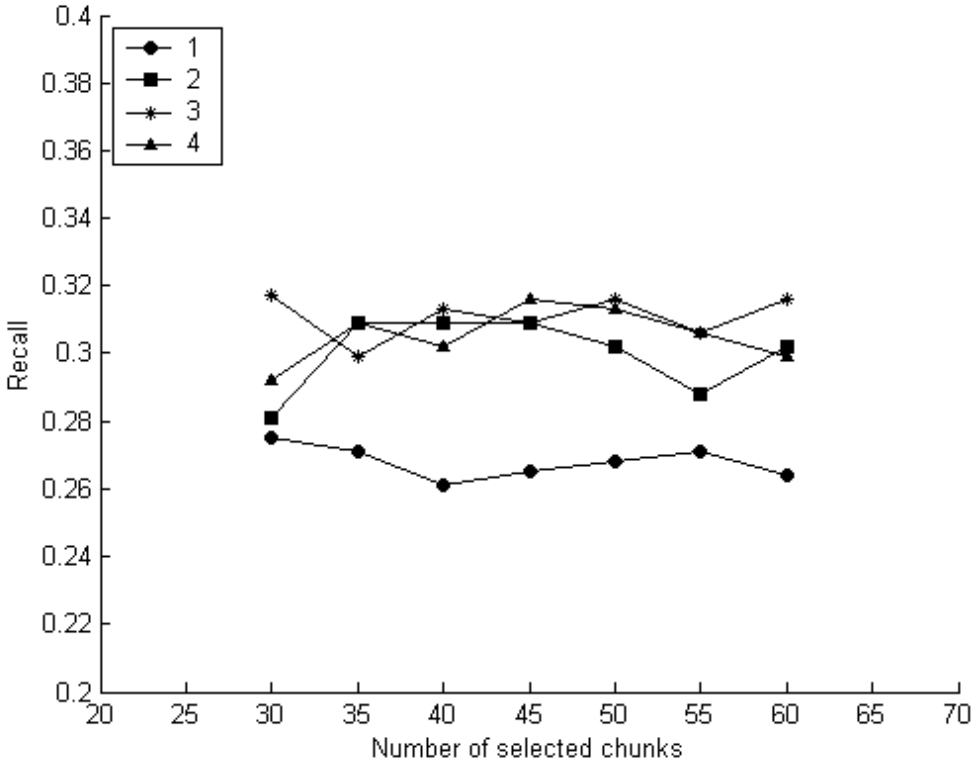
The following table contains the parameters that were kept constant:

SysIt	SemMs	#Pairs	m	p	c q
11	IR	150	5	1	0.7 0.075

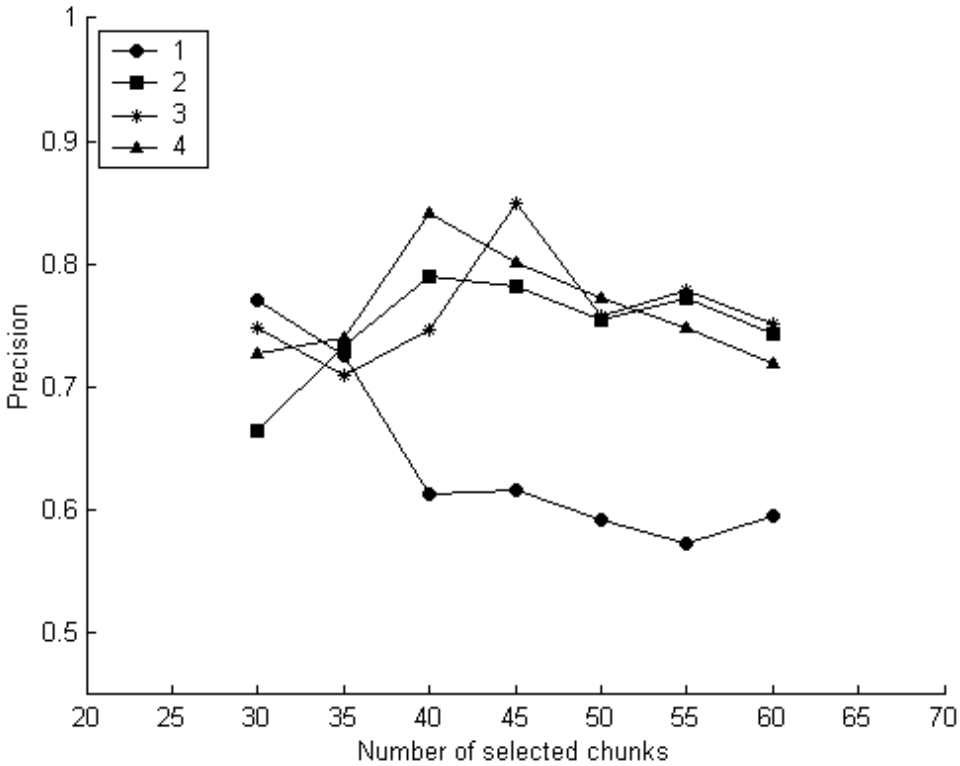
The numbers in the legend, denote the chunker's iterations at each iteration of the system.

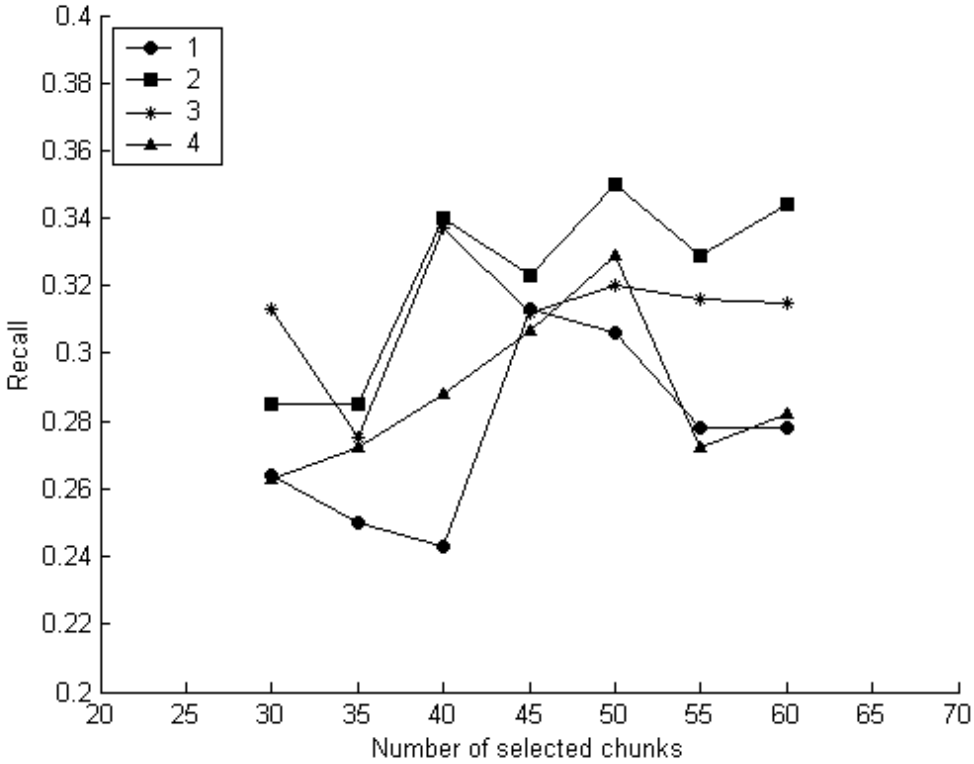
- SchMrg=3



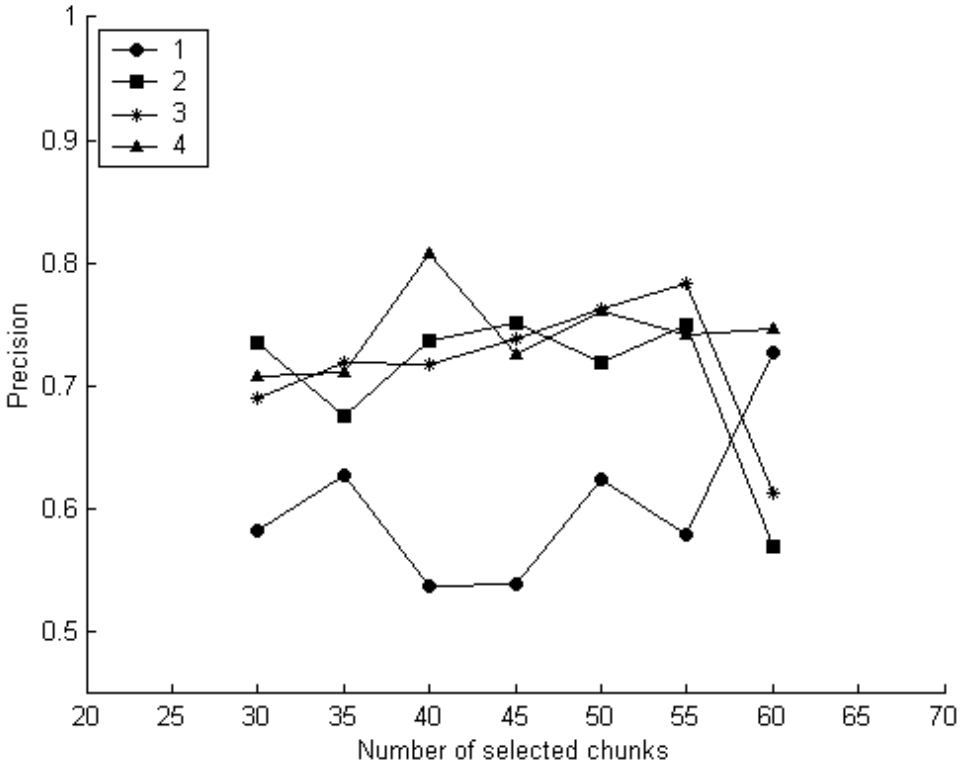

0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
Recall
1
2
3
4
20
25
30
35
40
45
50
55
60
65
70
Number of selected chunks

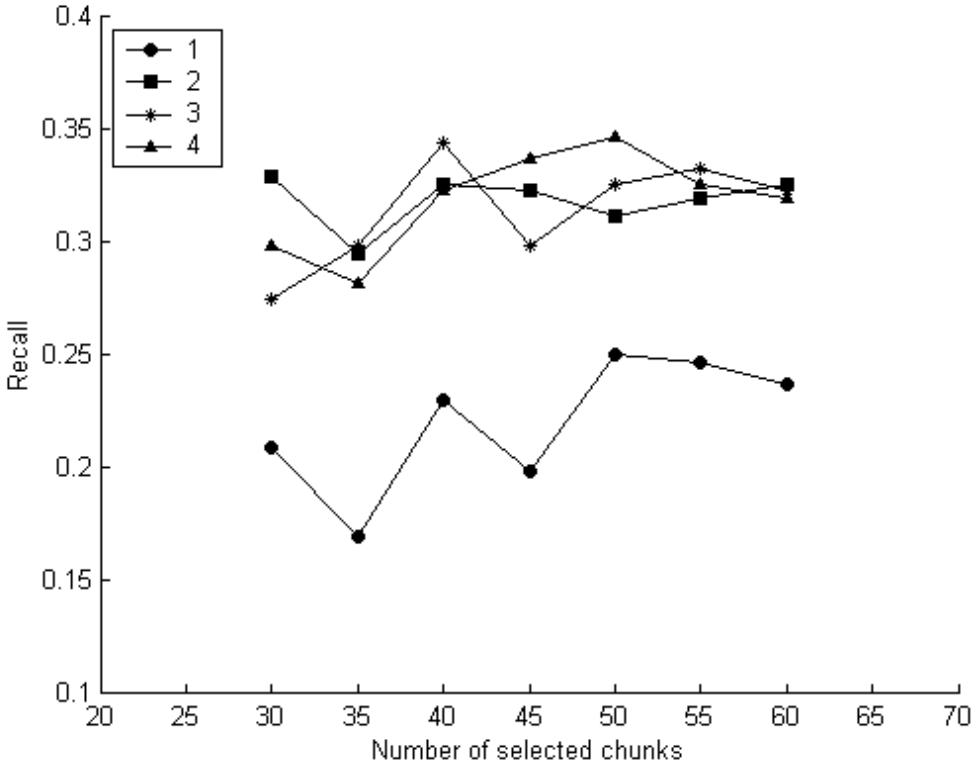
- SchMrg=7


1
0.9
0.8
0.7
0.6
0.5
Precision
1
2
3
4
20
25
30
35
40
45
50
55
60
65
70
Number of selected chunks


0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
Recall
1
2
3
4
20
25
30
35
40
45
50
55
60
65
70
Number of selected chunks

- SchMrg=10


1
0.9
0.8
0.7
0.6
0.5
Precision
1
2
3
4
20
25
30
35
40
45
50
55
60
65
70
Number of selected chunks

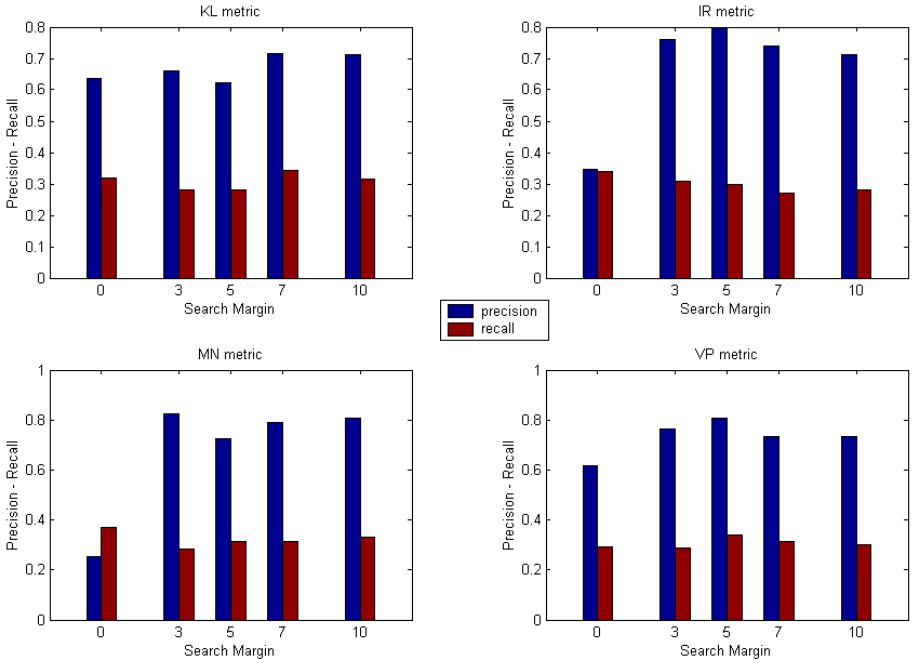


Comparison of the four semantic metrics

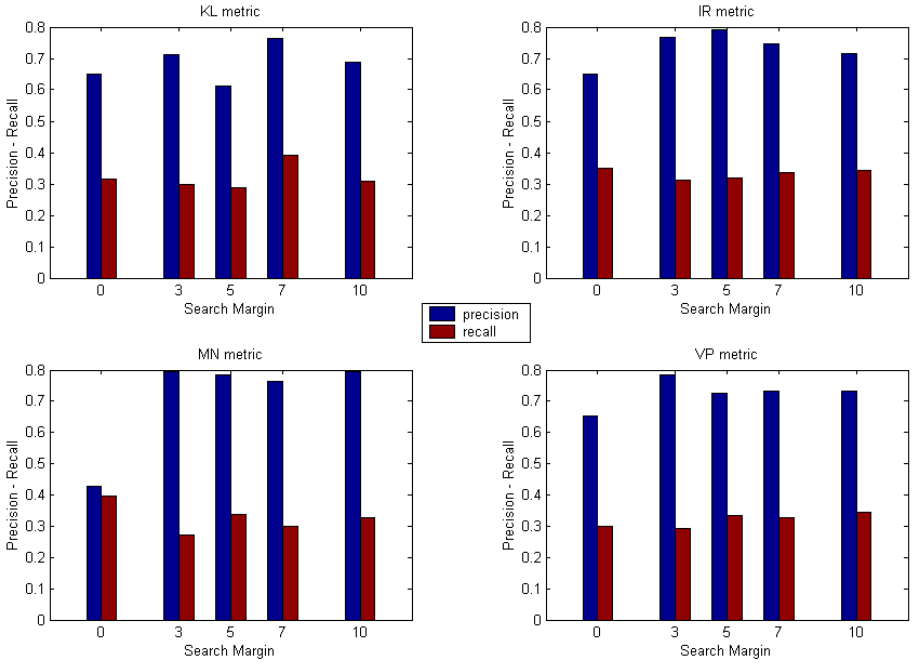
The following table contains the parameters that were kept constant during the experimental procedure:

SysIt	#Pairs	m	p	c q
11	150	5	1	0.7 0.075

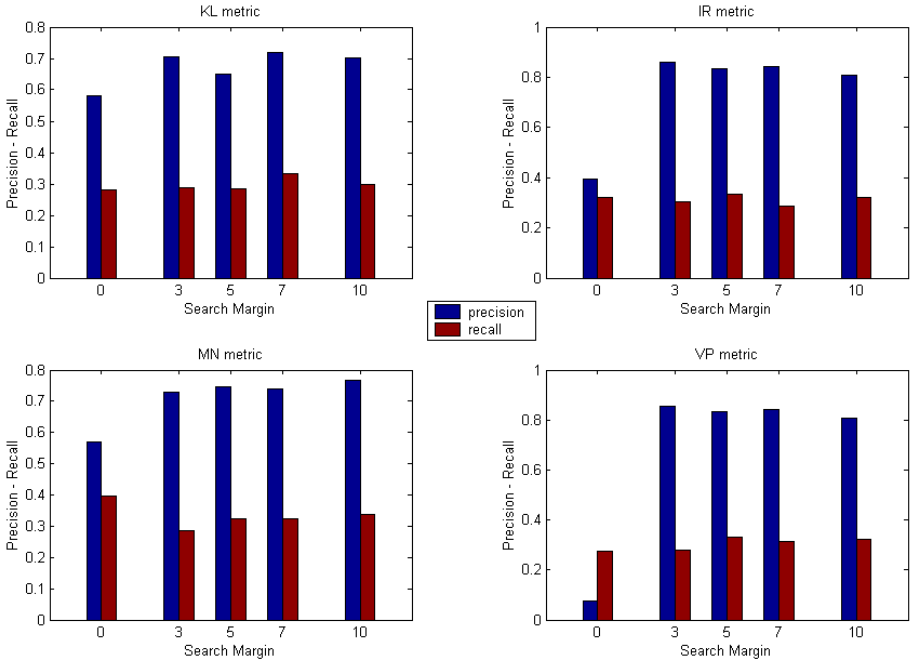
- ChuIt=4, #Ch=35



- ChuIt=3, #Ch=40



- ChuIt=4, #Ch=40


KL metric
IR metric
MN metric
VP metric
Precision - Recall
Search Margin
precision
recall
0
3
5
7
10

Bibliography

[1] Jurafsky, D., Martin, J.H., 2000. *Speech and Language Processing*. Prentice Hall. Upper Saddle River.

[2] Manning, C.D., Schutze, H., 2000. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge.

[3] Klavans, J.L., Resnick, P., 1996. *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*. The MIT Press, Cambridge.

[4] Cover, T.M., Thomas, J.A., 1991. *Elements of Information Theory*. John Wiley, New York.

[5] Pargellis, A., Fosler-Lussier, E., Lee, C.-H., Potamianos, A., Tsai, A., 2004. *Auto-induced Semantic Classes*. Speech Communication. 43, 183-203.

[6] Pargellis, A., Fosler-Lussier, E., Potamianos, A., Lee, C.-H., 2001. *A comparison of four metrics for auto-inducing semantic classes*. In: Proc. Automatic Speech Recognition and Understanding Workshop, Madonna di Campiglio.

[7] Siu, K.-C., Meng, H.M., 1999. *Semi-automatic acquisition of domain-specific semantic structures*. In: Proc. Sixth European Conf. on Speech Comm. and Tech., Budapest, vol. 5, pp 2039-2042.

[8] Fosler-Lussier, E., Kuo, H.-K.J., 2001. *Using semantic class information for rapid development of language models within ASR dialogue systems*. In: Proc. IEEE Internat. Conf. on Acoustics, Speech and Signal Proc., Salt Lake City.

[9] Riccardi, G., Bangalore, S., 1998. *Automatic acquisition of phrase grammars for stochastic language modeling*. In: Proc. of ACL Workshop on Very Large Corpora, Montreal.

[10] Church, K.W., Hanks, P., 1990. *Word association norms, mutual information and lexicography*. Computational Linguistics , vol. 16(1), pp 22-29.

[11] Solsona, R.A., Fosler-Lussier, E., Kuo, H.-K.J., Potamianos, A.,Zitouni, I., 2002. *Adaptive language models for spoken dialogue systems*.

[12] Tan, P.-N., Kumar, V., Srivastava, J., 2002. *Selecting the right interestingness measure for association patterns*. In: Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp 32-41.

[13] Terra, E., Clarke, C.L.A. *Frequency estimates for statistical word similarity measures*.

[14] Dagan, I., Lee, L., Pereira, F., 1997. *Similarity-Based Methods for Word Sense Disambiguation*. In: Proc. 35th Annual Meeting of the ACL, with EACL 8.

[15] Hindle, D., 1983. *Deterministic of Syntactic Non-fluencies*. In: Proc. 23rd Annual Meeting of the ACL.

[16] Clarkson, P., Rosenfeld, R., 1996. *Statistical language modeling using the CMU-Cambridge toolkit*.
(http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html)

[17] Shannon, C.E., 1948. *A mathematical Theory of Communication*. The Bell System Technical Journal, vol. 27, pp 379-23, 623-656.

[18] Witten, I. H., Bell, T.C., 1991. *The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression*. IEEE Transactions on Information Theory, vol. 37(4).

[19] Pantel, P., Lin, D. *A Statistical Corpus-Based Term Extractor*.

[20] Lee, L.J., 1997. *Similarity-Based Approaches to Natural Language Processing*, PhD thesis. Harvard University, Cambridge, Massachusetts.

[21] Young, J.Z., 1971. *An introduction to the study of man*. Oxford University Press.

[22] Chomsky, N., 1986. *Knowledge of language. Its nature, origin and use*. Praeger Publishers.

[23] Χειμωνάς, Γ., 1985. *Ο Λόγος. Μάθημα έβδομο και τελευταίο: ο χρόνος και το σύμβολο*. Εκδόσεις ΡΑΠΠΑ.

[24] Dagan, I. *Contextual word similarity*.

[25] Pargellis, A., Fosler-Lussier, E., Potamianos, A., Lee, C.-H., 2001. *Metrics for measuring domain independence of semantic classes*. In: Proc. 7th European Conf. on Speech Communication and Technology, Aalborg, Denmark.