Extending the Log Likelihood Measure to Improve Collocation Identification

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Bridget T. McInnes

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

December 2004

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of master's thesis by

Bridget T. McInnes

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Dr. Ted Pedersen

Name of Faculty Adviser

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

I would like to to take this time to acknowledge and thank the people who helped make this thesis possible.

To Dr. Ted Pedersen, for the guidance he has provided throughout the course of this work. I appreciate the time and energy you spent in teaching me how to conduct research. To the rest of my thesis committee – Dr. Rich Maclin and Dr. Barry James – I appreciate your careful reading of this thesis and thoughtful comments.

To Dr. Serguei Pakhomov and Dr. Guergana Savova for many insightful conversations during the development of the ideas in this thesis. Your patience, despite my many, many questions, is greatly appreciated.

To the Computer Science Department at the University of Minnesota Duluth. Specifically Dr. Carolyn Crouch, Dr. Donald Crouch, Linda Lucia, Linda Meek and Jim Luttinen for their support.

To my father, Gordon McInnes, for his endless encouragement and faith in me. Thank you for your continual interest in my research and allowing me to use you as a sounding board for my thoughts and ideas. Your insight is invaluable as well as much appreciated.

E Labore Dulcedo.

Contents

List of Figures

List of Tables

Abstract

Automatically identifying collocations in a text can be useful for applications such as machine translation and building lexicons or knowledge bases. This thesis presents an extension of the Log Likelihood ratio (G^2) to automatically identify collocations that consist of more than two words. G^2 is the ratio between how often an Ngram occurs compared to how often it would be expected to occur given a hypothesized model.

In the 2-dimensional case, i.e., collocations that consist of only two words, the only possible model is that of independence. G^2 calculates the observed count of an Ngram and compares it to the count that would be expected if the words were statistically independent. The score that G^2 produces reflects the degree to which the observed and expected values diverge. Calculating the expected values based on independence is commonly carried over to the three dimensional case but as the dimensions grow, so does the number of possible models.

Our approach calculates the G^2 of an Ngram for each of the different possible models and iteratively determines what model best represents the Ngram. The score the models return allow us to rank the Ngrams such that an Ngram that has a high G^2 score is a collocations while one with low G^2 score is not.

To calculate G^2, various co-occurrence and individual frequency counts of the words in an Ngram are needed. Traditionally, the method used for obtaining frequency counts for Ngrams is to count the number of times they appear in a corpus and store them. However, this becomes very limiting because of memory constraints that make it infeasible to process large data sets and the difficulty in obtaining reliable counts from corpora for rare words. This problem has lead to an increasing discussion on the feasibility of using the World Wide Web as a corpus. We explore the use of "hit" counts returned by the search engines Google and Alta Vista as the co-occurrence and frequency counts of an Ngram in order to calculate the G^2 score.

1 Introduction

The goal of this thesis is to automatically identify collocations from a text using an extension of the Log Likelihood Ratio (G^2). As an introduction, we would like to present terms and concepts that will be used throughout this thesis, as well as an overall picture of what this thesis is about.

Collocations, in simple terms, are units of words in which if the words were separated they would have a different definition than the unit itself. Some examples of collocations are: “a little bit”, “United States of America”, and “school bus”. These are strings of words that represent a single concept but whose individual components represent a different concept. Identifying collocations in a text becomes difficult even for humans. For humans, identifying collocations is a subjective task depending upon the domain in which the words appear and the context in which they are used; for a program, it is even more difficult. Programs do not have the contextual clues, domain knowledge and intuitiveness that we have to make these decisions.

Words in collocations tend to appear next to each other in text more often than what we would consider random. For example, “school” followed by “bus” occurs more often in general text than “school” followed by “bit”. This detail may make it possible for us to automatically identify these units through statistical measures of association such as the G^2. To calculate this ratio, we need to break our text into groups of words called Ngrams. Ngrams break up text into smaller chunks for processing where ”N” is an integer that represents the number of words in the chunk. We can then calculate the G^2 for the Ngrams to determine if they are collocations.

Typically, the G^2 determines if an Ngram is a collocation by taking the ratio between how often it occurs in a corpus compared to how often it would be expected to occur based on the model of chance (independence). A G^2 score of zero implies that the data fits perfectly with the hypothesized model; meaning that the Ngram exhibits complete independence between its tokens. Basing a hypothesized model solely on the model of independence results in high log likelihood scores for Ngrams that exhibit only partial dependence because they can not be completely represented by the independence model. In this thesis, we propose an extension of the G^2 by incorporating all possible hypothesized models; not just the model of independence. A high G^2 score for all of the hypothesized models indicate that the words in the Ngram do not exhibit independent behavior and a low score in at least one of the models indicates that there exhibits some sort of independence between the words in the Ngram.

This approach is evaluated by determining how well the extension of G^2 can identify collocation and how well it performs compared to other measures that have been used in the past to try and solve this problem. We compare our results to the frequency approach, the standard G^2 and the C-value approach proposed by Frantzi, Ananiadou and Hideki [11].

To use G^2, co-occurrence and frequency counts are needed for each of the Ngrams. These counts are the number of times that an Ngram exists in a corpus. Methods used for obtaining these counts typically involve iterating over the corpus and storing their individual counts in memory. This is very limiting due to memory constraints that make it infeasible to process very large data sets. Even with large data sets, rare and unusual words are not often observed. We explore the possibility of using the World Wide Web as a means of gathering these counts. The counts are obtained by posing an Ngram as a query and using the "hit" count returned by a search engine as the frequency count.

Automatically identifying collocations can be used for applications such as building lexicons or knowledge bases; information retrieval and machine translation. Previously, we have worked with developing a system to align words in parallel text which are two text that are exact translations of each other. We believe, if we could identify the collocations in the texts then groups of words could be aligned rather than just individual words; possibly improving translation techniques. New words are constantly being created especially in areas that have a specialized language like the medial domain. Identifying collocations automatically in literature would allow the automatic update of specialized lexicons and knowledge bases.

The contributions of this thesis are an extension of the Log Likelihood Ratio to evaluate trigrams and 4-grams and an improvement to the measure so that it can be used to extract three and four word collocations with better with better accuracy from a text. We have defined an schema to evaluate how well the approach works. We have also extended and evaluated various data structures that can be used to store and analyze trigrams and 4-grams. We have proposed an approach to collect frequency counts for various size Ngrams using the World Wide Web.

2 Background

Defining a $collocation$ is a challenge because it is not very well defined. [24] There are various definitions that have been used, for example 1) "a phrase consisting of two or more words that correspond to some conventional way of saying things" [19], 2) "distinctive entities requiring inclusion in a lexicon because their meanings are not unambiguously and derivable from the meanings of the words that compose them" [15] and 3) "the occurrence of two or more words within a short space of each other in a text" [10]. These definitions can be vague and imprecise for our use because they leave to much room open for interpretation. We believe to precisely define a collocation, we need to state ist behavior. Therefore, we define a collocation to be a unit words that exhibit non-compositionality, non-substitutablity and limited modifiability.

Non-compositionality is such that the meaning of the collocation can not be derived by looking at the words individually, non-substitutability means that the words in the collocation can not be substituted for another and still hold its exact meaning. Non-modifiability is that the collocation can not be modified with additional lexical material [19]; Wermter and Hahn [24] showned that some limited modifiability does exists in collocations.

There exist three main types of collocations: $idiomatic\,phrases$, $narrow\,collocations$, and $fixed\,phrases$ [24]. Idiomatic phrases are collocations in which none of the words in the collocation directly contribute to the overall meaning of the collocations itself; for example, "under the weather". Narrow collocations are those in which at least one word in the collocation contributes to the overall meaning of the collocation; for example "little black book". Fixed phrases are those in which all the words contribute to the overall meaning of the collocations, an examples of these would be "liver failure".

In this section, we introduce the idea of an $Ngram$ as a means of breaking text into smaller sized units in order to help analyze word segments. These units can be analyzed using statistical methods to determine whether the words in the unit makeup a collocation. This section focuses on defining an Ngram and describing how they can be obtained from a corpus and what methods are there for analyzing them.

2.1 Ngrams

Ngrams are defined as a contiguous or non-contiguous sequence of words, often called tokens, that occur in some proximity to each other in a corpus. Contiguous Ngrams, typically referred to as just Ngrams, are Ngrams whose tokens occur directly next to each other in a corpus while non-contiguous Ngrams, referred to as positional Ngrams, are Ngrams whose tokens are located within a specified window of each other rather than directly next to each other. For example, consider the phrase:

$$\texttt{to be or not to be} \tag{1}$$

The tokens would be: ”to”, ”be”, ”or”, ”not”, ”to”, and ”be”. The unique tokens, ”to”, ”be”, ”or”, and ”not” (called $types$) are the unigrams (1-grams) of the corpus. A bigram (2-gram) is a sequence of two tokens in a corpus and a trigram (3-gram) as a sequence of three tokens (Table 1).

The terminology describing Ngrams is also applicable when describing positional Ngrams. The positional bigrams, for our above sentence using a window size of three, can be seen in Table 2. The use of positional Ngrams can increase the number of observed Ngrams seen in a corpus that would not be otherwise be identified. For example, if our corpus contained the phrase ”Prime Minister Margaret Thatcher”, when determining all the trigrams in the corpus, we would extract ”Prime Minister Margaret” but miss ”Prime Minister Thatcher” if the positional Ngrams were not taken into consideration. Although, it has been shown that contiguous Ngrams are more likely to be collocations than non-contiguous Ngrams [5].

Table 1: Ngrams

unigrams	to	be	or	not
bigrams	to be	be or	or not	not to
trigrams	to be or	be or not	or not to	not to be

Table 2: Positional Bigrams (Window Size 3)

to be	to or	be or	be not	or not	or to	not to	no t be

2.2 Statistical Analysis

Tokens in a collocation tend to occur together more often than one would expect by chance. Statistical measures of association can be performed to determine the likelihood the tokens in an Ngram occur together more often than normal. For example, if “cardiac” is continually followed by ”infarction”, we can say that the tokens ”cardiac” and ”infarction” are closely associated with each other, meaning they occur together more often than random. Statistical measures give us a way to quantify this association.

Statistical measures are computed using various co-occurrence and individual frequency counts of an Ngram. The frequency counts of Ngrams can be conveniently displayed in a contingency table. In the following sections, we will discuss how contingency tables are represented, the notation associated with the tables and measures of association can be performed using information from the contingency tables.

2.2.1 Contingency Tables

Table 3 shows a contingency table for bigrams and the standard notation that is used. The cell n_{11} is the joint frequency of the bigram, the number of times the tokens in a bigram are seen together. The cell n_{12} is the frequency in which token1 occurs in the first position but token2 does not occur in the second position and the cell n_{21} is the frequency in which token2 occurs in the second position of the bigram but token1 does not occur in the first position. The cell n_{22} is the frequency in which neither token1 nor token2 occur in their respective positions in the bigram. The cells, n_{1p}, n_{p1}, n_{2p} and n_{p2} represent the marginal totals which are the number of times a word does/does not occur in the first or second position of the bigram. Lastly, the cell n_{pp} is the total number of bigrams found in the corpus.

Table 3: Contingency Table for Bigrams

	token2	$\neg$ token2	Totals
token1	n_{11}	n_{12}	n_{1p}
$\neg$ token1	n_{21}	n_{22}	n_{2p}
Totals	n_{p1}	n_{p2}	n_{pp}

Contingency tables can be created for Ngrams of any size n, although they become more complicated as n increases because the number of marginal counts increase by 2^n. An example, of a contingency table for trigrams can be seen in Table 4. The cell n_{111} contains the frequency of token1, token2 and token3 occurring together in their respective positions. The cell n_{112} contains the frequency in which token1 and token2 occur in their respective positions but token3 does not. The cells n_{121}, n_{122}, n_{211}, n_{212}, n_{221}, and n_{222} also represent similar frequency counts where 1 indicates that the token in that position is present and 2 indicates that it is not. The cells n_{11p}, n_{12p}, n_{21p}, n_{22}, n_{pp1}, and n_{pp2} represent the marginal counts and the cell n_{ppp} contains the total number of Ngrams.

Table 4: Contingency Table for Trigrams

		token3	¬ tokens3	Totals
token1	token2	n_{111}	n_{112}	n_{11p}
token1	¬ token2	n_{121}	n_{122}	n_{12p}
¬ token1	token2	n_{211}	n_{212}	n_{21p}
¬ token	¬ token2	n_{221}	n_{222}	n_{22p}
Totals		n_{pp1}	n_{pp2}	n_{ppp}

2.2.2 Measures of Association

The data in a contingency table can be used to evaluate an Ngram using measures of association. Four measures that are commonly used for Ngram statistics are Pearson's Chi-Squared (X^2) and the Log Likelihood Ratio (G^2) [7], Pointwise Mutual Information [3] and the Dice Coefficient [6]. These measures take into consideration what values one would expect to see in a contingency table versus what values are actually observed in a corpus. The values that we expect to see are estimated based on a hypothesized model which in this case is the independence model; is the hypothesis that the tokens in the Ngram happen to co-occur purely by chance. The values that one would expect to see based on a hypothesized model are called the expected values. To calculate the expected values, based on the model of independence, the product of the marginal total is divided by the total number of Ngrams in the text; seen in Table 5 where m_{ij} is the expected value.

Table 5: Contingency Table for Expected Values

	token2	¬ tokens2	Totals
token1	$m_{11} = \frac{n_{1p} * n_{p1}}{n_{pp}}$	$m_{12} = \frac{n_{1p} * n_{p2}}{n_{pp}}$	n_{1p}
¬ token1	$m_{21} = \frac{np1 * n_{2p}}{n_{pp}}$	$m_{22} = \frac{n_{p2} * n_{2p}}{n_{pp}}$	n_{2p}
Totals	n_{p1}	n_{p2}	n_{pp}

The X^2 and G^2 measures use these expected values to compare the significance of seeing commonly seen events versus rarely seen events. They are defined as:

$$G^2 = 2 * \sum_i^j n_{ij} * log(n_{ij}/m_{ij})$$

$$X^2 = \sum_i^j \frac{(n_{ij}-m_{ij})^2}{m_{ij}}$$

where n_{ij} are the observed frequencies of an Ngram and m_{ij} are the expected frequencies of an Ngram assuming that the Ngram is independent. X^2 and G^2 can be extended to determine the association between Ngrams for any size n. The distribution for X^2 and G^2 is χ^2 when the corpus size is large, the number cells in the contingency table is fixed and the expected values for each of the cells in the contingency table are large [22].

Pointwise Mutual Information (PMI) was proposed by Church and Hanks [3] as another way to determine the association of two words. This measure compares the probability of the words in an Ngram occurring together with the probability of the words occurring independently. This measure takes into consideration only a particular point in a large distribution. It looks only at the joint frequency of the bigram normalized over the total number of Ngrams. PMI is defined for bigrams as:

$$PMI = log\frac{n_{11}}{m_{11}}$$

where n_{11} is the known joint frequency, and m_{11}, is the expected joint frequency. This measure can be easily extended for any size n by taking the log of the observed joint frequency over the expected joint frequency of the Ngram. Limitations for this measure include over-rating events that occur in a corpus only once.

The Dice Coefficient [6] does not depend on the expected values of an Ngram. This measure only depends only on the frequency of the Ngram and the frequency of the individual words in the Ngram. This measure is basically twice the joint frequency of the tokens in the Ngram over the sum of their individual frequencies. Therefore, the Dice Coefficient is high when the tokens in the Ngrams occur together more often then they do separately. The Dice Coefficient is defined for bigrams as:

$$Dice = 2 * \frac{n_{11}}{n_{1p}+n_{p1}}$$

where n_{11} is the joint frequency of the bigram and n_{1p} and n_{p1} are the marginal totals. This measure also easily extends to Ngrams of any size, for example, the Dice Coefficient for trigrams can be estimated as:

$$Dice = 2 * \frac{n_{111}}{n_{1pp}+n_{p1p}+n_{pp1}}$$

where n_{111} is the joint probability, n_{1pp} is the number of times where token1 occurs in the first position, n_{p1p} is the number of times token2 occurs in the second position and n_{pp1} is the number of times token3 occurs in the third position.

2.3 Data Structures

There are a variety of data structures that can be used to identify Ngrams and their frequencies. A common data structure used for these type of problems is a hash table. A hash table is a direct address table that contains a key and an associated value. The time to find an element matching an input key is **O(1)**, for a good hash function, allowing for fast retrieval of Ngrams. The disadvantage to hash tables is the amount of memory that is needed to store Ngrams obtained from large corpora. In the following sections, we will discuss two possible alternative data structures, suffix arrays and masks, that have been used to store Ngrams.

2.3.1 Suffix Arrays

Suffix arrays were first introduced as a method to conduct string searches by Manber and Myers [18] and independently as Pat arrays by Gonnet [13]. It was demonstrated that the space requirements for suffix trees became greater than those of suffix arrays as the alphabet or token size increases by $O(\mid \Sigma \mid)$ where $\mid \Sigma \mid$ is the size of the alphabet. Suffix trees preceded suffix arrays and used a tree structure to store the data rather than an array. Suffix arrays have been show to have a definite advantage over suffix trees in terms of space when using English words rather than English characters. The data structure was then introduced as a way to obtain variable length Ngrams as well as their term and document frequencies by Church and Yamamoto [26] who showed that the number of sub strings in a corpus of size N was equal to $\frac{N(N+1)}{2}$. This allows Ngrams where $n > 2$ to be obtained using less space than other storage mechanisms such as hash tables.

The suffix array data structure entails the creation of two arrays where each element contains a token in the corpus. The first array is a token array containing the entire text that is to be processed, which we will refer to as the corpus array. The second array contains the index of each token in the corpus array, which we will refer to as the suffix array. To get a better idea of what the corpus and suffix arrays represent we will use a simple example corpus similar to the example seen in Church and Yamamoto [26]. In our example, each token is stored in its own indice in the corpus array as seen shown in Table 6. The suffix array is created to store each of the indices that exist in the corpus array, as seen in $InitialSuffixArray$ of Table 6. A simple way to describe this is that the suffix array now contains integers 0 through N, the size of the corpus array, representing the positions of all the tokens in the corpus array. The space requirements for these arrays can be estimated by $2 * N * B$ where N is the number of characters in the corpus and B is the number of bits each token takes.

Table 6: Suffix Array

Corpus Array	to	be	or	not	to	be
Initial Suffix Array	0	1	2	3	4	5
Sorted Suffix Array	5	0	3	2	4	1

Table 7: Sorted Suffix Array

Suffix Array Indice	Corpus Array Indice	Ngram
0	5	be
1	1	be or not to be
2	3	not to be
3	2	or not to be
4	4	to be
5	0	to be or not to be

Now that we have our corpus and suffix array created, there needs to be a way to arrange all possible unique Ngrams together in an order in which they can be accessed easily. To do this the suffix array is sorted in alphabetical order. This is done by sorting the elements in the suffix array based on what element they correspond to in the corpus array. Remember that that the corpus array contains the actual character and the suffix array contains the indice to where that character is located in the corpus array. An example of this can be seen in the Table 6.

The sorted suffix array represents all of the possible Ngrams that exist in the corpus with all like Ngrams situated next to each other. This form allows the unique Ngrams to be easily accessed as seen in Table 7.

The sorted suffix array allows for the ability to calculate the frequency of an Ngram very easily. If we look at the sorted suffix array in Table 7 the frequency of any Ngram can be determined by knowing the indice of the first and last occurrence of that Ngram in the suffix array. This allows us to determine frequency of that Ngram in the corpus by subtracting the indice of the first occurrence from the last occurrence and then adding one see in the following formula:

$$Ngramfrequency = i - j + 1 \tag{2}$$

where i and j are the first and last occurrence of the Ngram in the suffix array. For example, if we look at Ngram "to be" from our example corpus, the first occurrence of the Ngram is at indice 4 in the suffix array and last at indice 5. Therefore, using the Formula 2, we calculate a frequency of two for the Ngram "to be".

To calculate the frequency of all of the Ngrams, Church and Yamamoto [26] use a secondary array, the size of the corpus, to store the longest common prefixes, lcp, of adjacent Ngrams in the suffix array. Each indice, i, in the lcp array indicates the longest common prefix of the corresponding Ngram at position i and $i - 1$ in the suffix array as seen in Table 8. This secondary array was shown by Manber and Myers [18] to compute string searches in $O(P + logN)$ where P is length of the common Ngram in the corpus of size N. This implementation decreases the amount of time it takes to calculate the term frequency for all the Ngrams in the corpus but increases the amount of memory that is needed to implement suffix arrays because an additional array of size $N + 1$ is needed.

Table 8: LCP array

suffix[0]	be						lcp[0] = 1
suffix[1]	be	or	not	to	be		lcp[1] = 0
suffix[2]	not	to	be				lcp[2] = 0
suffix[3]	or	not	to	be			lcp[3] = 0
suffix[4]	to	be					lcp[4] = 2
suffix[5]	to	be	or	not	to	be	lcp[5] = 0

2.3.2 Masks

The Mask algorithm was then introduced by Gil and Dias [12] to store positional Ngrams from a corpus using minimal space requirements by adapting methods from suffix arrays described by Manber and Myers [18]. Gil and Dias [12] show that the number of positional Ngrams Δ can be calculated for a corpus of size

N and a window size of $2F + 1$. Therefore, storing these Ngrams in a hash table become infeasible, since as the window size increases the number of Ngrams also increases.

$$\Delta = (N - 2F)x \left(1 + F + \sum_{k=3}^{2F+1} \sum_{i=1}^{F} \sum_{j=1}^{F} C_{j-1}^{i-1} C_{j}^{k-i-1}\right) \tag{3}$$

Gil and Dias [12] create a corpus array similar to the corpus array described by Church and Yamamoto [26], introduced in the previous section. A secondary array is then created, this array stores the following information: the document number, for when Ngrams from multiple documents are to be collected, the starting position of the Ngram in the corpus array and a mask representation of the Ngram. The mask consists of a bit array. Remember that positional Ngrams are non contiguous Ngrams within a windows size. Therefore the bit array is an array, the size of the window, that stores a zero or one depending on whether the token in that position exists in the Ngram. The one indicates that the token in that position is included in the Ngram and zero indicates that the token in that position is not in the Ngram.

Table 9: Positional Ngram Corpus Array

corpus	to	be	or	not	to	be
array indice	0	1	2	3	4	5

If we consider our trivial example corpus "to be or not to be" where each token is an element in the array, our corpus array can be seen in Table 9. The positional Ngrams for this corpus, using a window size of three are "to be", "to or", "be or", "be not", "or not", "or to", "not to" and "not be". If we consider the positional Ngram "not be", assuming our example is our first document, the document number we will set to one since we are considering the example as one document. The starting position would be at indice three in the corpus array. The bit mask would consists of a bit array the size of the window, in this case three. The first element in the bit mask would represent "not", the second element would represent "to" and the last element would represent "be" as seen in Table 9. The elements in the mask would contain a zero or a one indicating whether or not the associated token exists in the Ngram. The bit mask for this example would contain the values 101, where the first one represents the token "not", the zero represents the token "to" and the last one represents "be". An array containing the document number, starting position and bit mask for each possible positional Ngram in our example corpus can be seen in Table 10.

The frequency for these Ngrams can be obtained in a similar fashion as the suffix array implementation. The array of bit masks is sorted in alphabetical order situating like Ngrams next to each other. The frequency

Table 10: Positional Bigram Representation

Ngram	document number	start position	bit mask
to be	1	1	110
to or	1	1	101
be or	1	2	110
be not	1	2	101
or not	1	3	110
or to	1	3	101
not to	1	4	110
not be	1	4	101
to be	1	5	110

is then calculated using the formula $j - i + 1$, where i and j represent the first and last occurrence of the Ngram in the array. For example, to determine the frequency of "to be", the first occurrence is at indice seven and the last occurrence is at indice eight see in Table 11. Using our above formula, we can calculate the frequency of the Ngram: $8 - 7 + 1 = 2$

Table 11: Sorted Bit Mask Array

element	Ngram	mask
0	be not	12101
1	be or	12110
2	not be	14101
3	not to	14110
4	or not	13110
5	or to	13101
6	to be	15110
7	to be	11110
8	to or	11101

3 Data Structure Implementation

The identification of collocations requires the efficient extraction of Ngrams from corpora in order to analyze the data in smaller segments. Obtaining Ngrams from a corpora is not a trivial tasks. It has been shown that most NLP tasks that require learning algorithms benefit significantly from using larger sources of data. We are limited in the amount of data that can be processed mostly by the amount of memory that is available to us to process the corpus. In this section, we discuss three algorithms that were developed to extract Ngrams and their frequency counts from corpora: suffix arrays, masks and an extended hash table approach. Each of these algorithms discussed in the previous section were implemented using the Perl scripting language. The advantage of Perl is its ability to handle regular expressions which are necessary when defining the form of a token. This advantage weighs heavily in the decision to use Perl because most other languages do not have the ability to define regular expressions as precisely as Perl. In addition, Perl is a very portable anlanguage that runs on most operating systems and platforms.

For most counting problems, hash tables and arrays are most commonly used because of their fast retrieval of data and ease of use. The disadvantage of these data structures is the amount of memory that is needed to store Ngrams from a large data set when $n > 2$ becomes infeasible for most computer systems commonly available today..

This disadvantage was overcome in the masks and suffix array implementation by using the built in Perl function *vec()*. The vec() data structure is a Perl primitive that allows for the compact storage of unsigned integers. The integers are packed as tightly as possible in a typical Perl string. The vec() data structure requires the specification of three parameters: the Perl string in which the integers are to be packed, the offset and the bits. The bits specify how many bits the value can be stored in, hence the parameter name bits. The offset parameter allows us to access an element in the vec() similarly to how you would with an array. For example, an offset of two with a bit parameter of 32 would technically represent the number stored in the vec() string between bit 64 and 96. The Perl vec() structure has the ability to store 67,108,000 integers into memory while a traditional Perl array can only store 8,315,000 integer on a Solaris system with 512 MB of Memory. This additional storage dramatically effects the size of corpus that can be used when experimenting.

The extended hash table approach breaks a text into manageable sized pieces and uses a hash table approach determine the Ngrams from each of the pieces separately rather than keeping all of the possible Ngrams from the entire corpus in memory.

We compared each of our implementations to the count.pl program from the Ngram Statistic Package [1] which is a collection of Perl programs that can be used to analyze Ngrams in text files. The count.pl program in this package is used as a base line to compare with our programs because it determines the Ngrams and their frequencies using a single hash table.

The analysis was conducted using *nyt200102*, *nyt200103*, *nyt200104*, *nyt200105* and *nyt200106* data sets from the New York Newswire Service compiled by the Linguistic Data Consortium (LDC) Documentation for English Gigaword. Each file consists of approximately ten million tokens. More information about the English Giga-Word corpus is described in Section 5. The experiments were run on segments of this corpus consisting of 10, 20, 30 40 and 50 million tokens. It was found that the amount of memory needed to determine all the Ngrams was less than when using a single hash table.

3.1 Suffix Array Implementation

Our suffix array implementation is a modification of the algorithm presented by Church and Yamamoto [26]. The modifications are due to language and memory constraints. In our implementation, we are more concerned about memory than speed because we would like to experiment with the largest size corpus as possible. Oursuffix array implementation converts all the tokens in the corpus to integers therefore each type in the corpus has a unique integer [21]. This is done to reduce the amount of memory needed to store the corpus in memory and use the Perl vec() function. Two vec() functions, each the size of the number of tokens in the corpus, are allocated. The first vec() stores the corpus where each element in the vec() is an integer representing the appropriate token. For example, if we consider the corpus fragment:

```
to be or not to be
```

each type in the corpus would be assigned an integer value and stored in a vec(), as seen in Figure 12. A second vec() stores the location of each token in the first vec() in sorted order, seen in Figure 12. The Perl language does not have a built in sorting function for the vec() data structure therefore the vec() is created in sorted order using the following algorithm. In a hash of arrays, each unique token, type, in the corpus

```
sub suffix_array {
  my %w = ();
  # store all the unique integers and their indices from the corpus vec()
  for(0..$N) { push {$w{vec()($corpus, $_, $bit)}}, $_; }
  my $count = 0;
  # for each unique integer sort their indices and store in the suffix vec()
  foreach (sort keys %w) {
    foreach my $elem (sort bysuffix @{$w{$_}}) { vec($suffix, $count++, $bit) = $elem; }
  }
}
```

Figure 1: The Suffix Array Creation Function

is stored with its corresponding locations in the first vec() as seen in Figure 1. The hash is then traversed in sorted order based on the types. For each key in the hash, the corresponding array, which contains the location of where the type exists in the first vec(), is traversed in sorted order and stored in the second vec(). The array is sorted based on the tokens that precedes the type whose locations are stored in the array.

Table 12: Suffix Array Vec()

First vec()	1	2	3	4	1	2
Second vec()	5	0	3	2	4	1

For example, the integer one corresponds to the token “to” which occurs in position zero and four in the first vec(). The array will then contain the integers zero and four. This array is sorted by looking at the first occurrence of the postceding tokens that are not equal, in this case two which corresponds to “or” and the blank space. The array then would order itself as four and two. The code for this can be seen in Figure 2.

The retrieval of the Ngrams and their respective frequencies uses a different approach than what was described by Church and Yamamoto [26] in Section 2 of this paper. Church and Yamamoto [26] use a secondary array called the longest common prefix, *lcp*, array to increase the speed in the determination of the Ngrams and their frequencies. The array requires additional memory which our implementation is trying to

```
sub bysuffix {
    my $z = $a; my $x = $b; my $counter = 0;
    # find the first occurrence where the Ngrams differ
    while(vec($corpus, ++$z, $bit) == vec($corpus, ++$x, $bit) && ++$counter ¡ $max_Ngram_size) {;}
    # check to see what value is greater and return the appropriate value
    return ( vec($corpus, $z, $bit) == vec($corpus, $x, $bit) ? 0 :
            (vec($corpus, $z, $bit) ¡ vec($corpus, $x, $bit) ? -1 : 1));
}
```

Figure 2: Sorting by Suffix

avoid.

In our implementation Ngrams and their respective frequencies are determined by traversing the second vec() and maintaining two offset variables in order to calculate the frequency, the first occurrence and last occurrence of the current Ngram. These are updated whenever the previous Ngram is not equal to the current Ngram. At this point, we know the first and last occurrence of the previous Ngram allowing us to calculate the frequency of the Ngram using the formula described by Church and Yamamoto [26] where the frequency equals the the index of the last occurrence of the Ngram minus the index of the first occurrence plus one. The previous Ngram and its frequency then can be printed out to a file.

The marginal frequencies of the Ngrams can be fully obtained for bigrams and trigrams. For bigrams, the marginal frequencies consist of the number of times each of the individual tokens in the bigram occur in their respective positions. This information is partially obtained during the creation of the suffix vec() where the unigram counts for each type in the corpus are obtained. These counts are then modified to take into consideration the first and last token in the corpus, and tokens that are removed from the bigram count. These frequency counts are available for Ngrams of any size.

A complete set of marginal frequencies can be returned for Ngrams where $n <= 3$. To determine the marginal frequencies for a trigram, $t_1t_2t_3$, there are six frequency counts that need to be returned. The frequency of the individual tokens in their respective positions, which is described above, and the frequencies of t_1t_2, t_1t_3, and t_2t_3. These frequencies of t_1t_2 and t_2t_3 are calculated by determining the first and last occurrence of the Ngram to compute the frequency. Determining the frequency of t_1t_3 is more time consuming and requires to loop through all the Ngrams that begin with t_1 and count the number of Ngrams that have t_3 in the third position. This procedure is time consuming; to help increase the speed, the first occurrence of every token is cached and stored in a secondary vec() data structure if this option is requested.

The complete set of marginal frequencies are not calculated for Ngrams where n is greater than three because as n grows the number of marginal frequencies that need to be obtained increase resulting in an increase in the time needed to obtain each of these frequency counts.

The main advantage to suffix arrays over other types of storage mechanisms, such as arrays and hash tables that hold the each individual Ngram, is that the same suffix array is created regardless of the Ngram size. Therefore, the retrieval of these Ngrams require no additional memory, meaning that the amount of memory it would take to store bigrams using the suffix array implementation is the same amount of memory that it would take to store any size of Ngram. An example of this can be seen in Table 13, where it can be observed that the memory usage for retrieving bigrams and trigrams is the same regardless of the corpus size. This is because the suffix array implementation only involves the creation of two arrays, each of the size of the corpus, regardless of what size Ngrams are being retrieved.

A comparative analysis was conducted between our suffix array implementation and count.pl for identifying Ngrams and their joint frequency counts. We found that the suffix array implementation and NSP were comparable when determining the bigrams from a corpus. As n increased, though, the memory usage for NSP increased while the memory usage for suffix arrays did not. The results can be seen in Table 13. We concluded that for Ngrams where $n \geq 2$, the suffix array implementation had a significant advantage because it allowed for a larger corpus to be used to identify Ngrams. However, the suffix array implementation does not calculate the complete set of marginal values for most n, the majority of known statistics for Ngrams where $n \geq 2$ do not use these values in their estimations. However, the Log Likelihood Ratio does require these marginal counts for its calculation. Our implementation of suffix arrays can be obtained at http://search.cpan.org/ btmcinnes/Array-Suffix-0.3/ and the NSP Package can be obtained at http://search.cpan.org/ tpederse/Text-NSP-0.71/.

Table 13: Suffix Array Memory Usage

		bigrams		trigrams	
Corpus	Corpus Size	Suffix Array	NSP	Suffix Array	NSP
nyt200202	10 million	320 MB	430 MB	320 MB	849 MB
nyt2002023	20 million	620 MB	730 MB	620 MB	1.7 GB
nyt2002024	30 million	980 MB	1.2 GB	980 MB	out of memory
nyt2002025	40 million	1.1 GB	1.4 GB	1.1 GB	out of memory
nyt2002026	50 million	1.4 GB	1.4 GB	1.4 GB	out of memory

3.2 Mask Implementation

Our mask implementation is based on the algorithm described in **Using Masks, Suffix Array-based Data Structures and Multi dimensional Arrays to Compute Positional Ngram Statistics from Corpora** by Gil and Dias [12]. The masks implementation retrieves all contiguous and non-contiguous (positional) Ngrams for a corpus. Initially, like in the suffix array implementation, all the tokens in the corpus to integers and a vec() is created containing all the tokens in the corpus file.

An array containing bit mask representations of all of the Ngrams is then created by traversing the vec() containing all the tokens in the corpus file. Each bit mask represents an Ngram and contains the Ngram document number, the starting position of the Ngram in the vec() containing the corpus and a bit vector of the Ngram. The bit vector is a small vec() the size of the window containing either a one or a zero in each index where the one indicates that the token in that position is included in the Ngram and a zero indicating that is not included in the Ngram. We can see an example of this using the following corpus:

`to be or not to be`

Using a window size of three, if we consider the Ngram “or to”, the document number would be one and the starting position of the Ngram would be two since that is the starting index in which the Ngram occurs in the corpus vec().The bit mask would consist of a vec() of size three, the window size, and contain the elements 101 where the first one corresponds to ”or” which exists in our Ngram, the zero corresponds to ”not” which does not exist in our Ngram and the last one corresponds to ”to” which does exists in our Ngram. The entire mask representing the Ngram ”or to” would be: 12101.

All the positional Ngrams are obtained from the corpus by looping from 0 to $2^{windowsize} - 1$ for each token

```
sub byvec {
   @aarray=(); @barray=();$z=0; $x=0; $counter=0; $aindex = vec($a, 1, $bit); $bindex = vec($b, 1, $bit);
   for my $i(2..$window_size+1) {
      if(vec($a, $i, 1) == 1) { push @aarray, vec($corpus, $aindex, $bit); } $aindex++; }
      if(vec($b, $i, 1) == 1) { push @barray, vec($corpus, $bindex, $bit); } $bindex++; }
   }
   for $z(0..$#aarray) { if($aarray[$z]! =$barray[$z]) { $x = $z; next; } }
   return ( $aarray[$x] > $barray[$x] ? 1 : ($aarray[$x] < $barray[$x] ? -1 : 0) );
}
```

Figure 3: Sort the Window Array

in the corpus vec() because there exists $2^{windowsize} - 1$ possible Ngrams for each window. The index of the nested loop is converted into a binary representation to obtain the bit vector representation of the possible Ngram. The Ngram is then checked to determine if it is an actual Ngram using the following criteria: the Ngram must be within the confines of the corpus, and the total sum of ones in the bit array must be greater than or equal to the minimum Ngram size and less than or equal to the maximum size Ngram. If the criteria for an Ngram is met the document number and starting position of the Ngram in the corpus are stored in the window array with the bit vector representation of the Ngram.

The array of vec() functions is then sorted based on the Ngram representation, similar to how suffix arrays are sorted using Perl's sort function. The positional Ngrams are determined from their respective Ngram vectors, and the two tokens of the Ngram are compared and the appropriate value of one or negative one is returned. The Perl code for this can be seen in Figure 3.

The sorted array vec() functions is then traversed and the positional Ngrams and their frequencies are written to a file. The frequency of any positional Ngram can be determined similar to the way the frequencies are calculated in the suffix array implementation. For example, to determine the frequency of the Ngram "to be" in our corpus fragment "to be or not to be", we find that the first occurrence of "to be" is at index six in the window array and the last occurrence is at index seven as seen in Table 14. The quotient of the indices plus one is two which is the frequency of our Ngram.

Table 14: Frequency of Bigram “to be”

Window Array Index	Positional Bigram	Bigram Vector
6	to be	10110
7	to or	10101

Table 15: Masks Memory Usage

		bigrams		trigrams	
Corpus	Corpus Size	Masks	NSP	Masks	NSP
nyt200202	10 million	630 MB	800 MB	1.0 GB	1.8 GB
nyt2002023	20 million	680 MB	1.0 GB	1.8 GB	Out of Memory

We conducted a comparative study between the positional Ngram implementation and the count.pl program from the NSP Package obtaining only the Ngram and the joint frequencies. For bigrams, a window size of three was used and for trigrams a window size of four. We found that our mask implementation used less memory to obtain the Ngrams and their joint frequency count as seen in Table 15. This would allow more Ngrams to be determined over a larger set of data. The down side to the masks algorithm is that, like the suffix array implementation, the complete set of marginal values are not obtained.

Our masks implementation has the option to identify only contiguous Ngrams if the window size is not set but because the suffix array implementation uses less memory to identify these Ngrams than the masks implementation it has been concluded that our masks implementation should only be used to find positional Ngrams. Our implementation can be found at http://search.cpan.org/ btmcinnes/Text-Positional-Ngram-0.3/.

3.3 Extended Hash Table Implemementation

The extended hash table approach is an extension of the count.pl program found in the NSP package. The basic idea of this algorithm is a large corpus is broken into chunks, the Ngrams are determined over each of the chunks and then combined to obtain a list of all Ngrams and their marginal counts over the entire corpus.

The extended hash table implementation breaks a corpus into manageable size chunks designated by the user. The chunk size should be set to a size that will use less memory than what system has available. This allows the user to determine what size chunks are best for the system that the program is being run on. The Ngrams and their marginal counts are then determined over each of the individual chunks using the count.pl

```
sub extended_hash {
    %Ngram_hash = (); %increment_hash = (); %marginal_hash = ();
    open(FILE, $file); my $file_Ngram = FILE; while(FILE) { store_Ngram( $_ ); }
    open(TEMP1, $master); open(TEMP2, ">$temp_file");
    foreach (keys %Ngram_hash) {
        my $hash_Ngram = print_hash_Ngram($_);
        print TEMP "$hash_Ngram";
    }
    system "mv $temp_file $master_file";
}
```

Figure 4: The extended_hash Function

program from the Ngram Statistics Package (NSP).

The Ngrams from each of the files generated by the count.pl program are combined one at a time to create a master list of all the possible Ngrams and their marginals. The combination is conducted by reading one count file into memory, executing the recombination algorithm, storing the results in a master file and then reading in another count file. The code for this can be seen in Figure 4. This allows for only a subset of the Ngrams to be in memory at any one time reducing the amount of memory needed and keeping the memory load constant.

There are two hash tables that are populated while the count file is being read. The first is a hash of arrays where the key is the Ngram itself and each array contains the marginal totals associated with that Ngram. The second is a table that contains the marginal counts and their corresponding frequencies.

The master file, which will eventually contain a list of all of the Ngrams, is opened and the Ngrams are read in one at a time and processed. The processing consists of first determining if the Ngram was seen in the count file, if so the joint frequency is incremented. Then determine if the any of the marginal counts were seen in the count file. If they were the Ngram from the master file is incremented and stored in the increment hash. If the Ngram did not exist in count file then it is saved to a temporary file. The Perl code for this can

```
sub increment_marginals {
   chomp; my f_Ngram_array = split/<>/, shift; my @f_marginals = split/ /, (pop @f_Ngram_array);
   my $f_Ngram = join ”<>”, @f_Ngram_array;

   #if the Ngram exists in the Ngram_hash increment the frequency
   if( exists $Ngram_hash{$f_Ngram} ) { $  $Ngram_hash $f_Ngram  [0] += $f_marginals[0]; }

   #now check the rest of the marginals if they exists
   for $i (1..$#f_marginals){
      my @combo = split , $combo_array[$i]; my @combo_Ngram = ();
      map  $combo_Ngram[$_] = $f_Ngram_array[$combo[$_]]  0..$#combo;
      if( exists $marginal_hash( (join ”<>”, @combo_Ngram) . ”<>” . $i ) ) {
         $f_marginals[$i] += $marginal_hash{( (join ”<>”, @combo_Ngram) . ”<>” . $i )};
         $increment_hash{((join ”<>”, @combo_Ngram) . ”<>” . $i )} = $f_marginals[$i];
      }
}
```

Figure 5: The increment_marginals Function

be seen in Figure 5.

After all of the Ngrams in the master file are processed, the marginal counts from the Ngrams in the hash table are incremented if needed and printed to the temporary file. When that is finished the temporary file becomes the new master file and the algorithm continues until all the files generated by the count.pl program are processed.

We conducted a comparative analysis between the extended hash table approach and the count.pl program using a corpus of 10 million tokens and 20 million tokens.For each experiment the files were split into 4 chunks and a complete set of marginal values were determined. The extended hash table approach uses the count.pl program to determine the Ngrams over each of the split files, therefore the memory usage for this will not be recorded. It is obviously smaller than the amount of memory used to determine the Ngrams over

Table 16: Extended Hash Memory Usage

		bigrams		trigrams	
Corpus	Corpus Size	split-count	count	split-count	count
nyt200202	10 million	130 MB	430 MB	564 MB	849 MB
nyt2002023	20 million	150 MB	730 MB	600 MB	1.7 GB

the entire file. Therefore, the memory usage displayed for the extended hash algorithm in Table 16 is the amount of memory that is used at the recombination stage which is smaller than the amount of memory used by the count.pl program.

The significant advantage to the extended hash table approach over all the previously discuessed approaches is that the marginal values of the Ngrams can be obtained using less memory than either the masks or the suffix array implementation. Although as stated previously, the majority of known statistics for Ngrams where $n \geq 2$ do not use these values in their estimations, the Log Likelihood Ratio does.

The disadvantage of the extended hash table approach is the amount of time it takes for the program to complete. For the experiment of 10 million tokens, it took 1 hour to complete, while the count.pl program completed in 36 minutes. We conclude that the choice of which program to use should be determined on the amount of data that needs to be processed and the amount of memory that is available to your system.

4 The Log Likelihood Ratio

The Log Likelihood Ratio is a "goodness of fit" statistics that was first proposed by Wilks [25] to test if a given piece of data is a sample from a set of data with a specific distribution described by a hypothesized model. It was later proposed by Dunning [7] as a way to determine if the words in an observed Ngram come from a sample that is independently distributed; meaning they occur together by chance. We can then describe G^2 as the ratio between how often an Ngram actually occurred compared to how often it would be expected occur. In this measure, the observed and expected values for each "cell" in a contingency table are compared.

The G^2 ratio compares the observed frequency counts with the counts that would be expected if the tokens in the Ngram corresponded to our hypothesized model. Typically the hypothesized model has been the model of independence. The model of independence is the probability that two words have occurred together by chance. More formally, it is when the probability that two words occur together is equal to the product of the their individual probabilities defined as:

$$p(word1, word2) = p(word1) * p(word2)$$

A G^2 score reflects the degree to which the observed and expected values diverge. A G^2 score of zero implies that the data fits perfectly into the hypothesized model and the observed values are equal to the expected. Therefore, the higher the G^2 score, the less likely the tokens in the Ngram appear correspond to the hypothesized model.

In this section, we will discuss hypothesized models and how to determine what model best represents an Ngram using G^2. We will then discuss significance testing which has been used to establish a threshold cutoff to determine at what point all the Ngrams above the threshold are collocations and all the ones below are not.

4.1 Hypothesized Models

Calculating G^2 in the 2-dimensional case has only one possible hypothesized model to compare against, the model of independence. Calculating the expected values based on the model of independence is commonly carried over to trigrams and 4-grams but as the dimensions of the contingency table grow so does the number of available models in which the tokens can be compared to. The expected values for a trigram can be based on four models. The first trigram model, in Table 17, is the model of independence previously. The second is the model based on the probability that $word1$ and $word2$ are dependent and independent of $word3$, the third model is based on the probability that $word2$ and $word3$ are dependent and independent of $word1$ and the last model is based on the probability that $word1$ and $word3$ are dependent and independent of $word2$. For 4-grams, the expected values can be based on 14 possible models seen in Table 18.

Table 17: Trigram Models

Model 1 : $P(word1word2word3)/(P(word1)P(word2)P(word3))$
Model 2 : $P(word1word2word3)/(P(word1word2)P(word3))$
Model 3 : $P(word1word2word3)/(P(word1)P(word2word3))$
Model 4 : $P(word1word2word3)/(P(word1word3)P(word2))$

Table 18: 4-gram Models

Model 1: $P(w1w2w3w4)/(P(w1)P(w2)P(w3)P(w4))$	Model 2: $P(w1w2w3w4)/(P(w1w2)P(w3w4))$
Model 3: $P(w1w2w3w4)/(P(w1)P(w2w4)P(w3))$	Model 4: $P(w1w2w3w4)/(P(w1w3)P(w2w4))$
Model 5: $P(w1w2w3w4)/(P(w1)P(w2w3)P(w4))$	Model 6: $P(w1w2w3w4)/(P(w1w4)P(w2w3))$
Model 7: $P(w1w2w3w4)/(P(w1)P(w2w3w4))$	Model 8: $P(w1w2w3w4)/(P(w1w4)P(w2)P(w3))$
Model 9: $P(w1w2w3w4)/(P(w1w3)P(w2)P(w4)$	Model 10: $P(w1w2w3w4)/(P(w1w3w4)P(w2)$
Model 11: $P(w1w2w3w4)/(P(w1w2)P(w3)P(w4))$	Model 12: $P(w1w2w3w4)/(P(w1w2w4)P(w3))$
Model 13: $P(w1w2w3w4)/(P(w1w2w3)P(w4))$	Model 14: $P(w1w2w3w4)/(P(w1)P(w2)P(w3w4))$

The hypothesized models result in different expected values which therefore will result in different G^2 score. The expected values for trigram Model 1 can be estimated using Equation 4 where m_{ijk} is the expected value for its corresponding cell in the contingency table. The parameter, n_{ppp}, is the total number of Ngrams that exist, and n_{ipp}, n_{pjp}, and n_{ppk} are the individual marginal counts of seeing tokens i, j, k in their respective positions in a trigram.

$$m_{ijk} = \frac{n_{ipp} * n_{pjp} * n_{ppk}}{n_{ppp}^2} \tag{4}$$

Calculating the expected values for the other hypothesized models result in a slightly different formula. To understand how we obtain the expected values for these models, it would be beneficial to see how we arrived at Equation 4 for the independence model. Expected values are obtained by calculating the product of the probability of seeing each of the tokens of the Ngram in their respective positions and multiplying that by the total number of Ngrams as seen below:

$$m_{ijk} = n_{ppp} * p_{ipp} * p_{pjp} * p_{ppk}. \tag{5}$$

The probability of seeing a token of a trigram in its respective position is:

$$p_{ipp} = \frac{n_{ipp}}{npp}, \quad p_{pjp} = \frac{n_{pjp}}{npp}, \quad p_{ppk} = \frac{n_{ppk}}{npp} \tag{6}$$

Therefore, substituting the probabilities of each of the individual tokens in Equation 5 with their respective variables in Equation 6, will result in the following formula:

$$m_{ijk} \quad = \quad n_{ppp} * \frac{n_{ipp}}{npp} * \frac{n_{pjp}}{npp} * \frac{n_{ppk}}{npp} \quad = \quad \frac{n_{ipp}*n_{pjp}*n_{ppk}}{n_{ppp}^2}$$

Therefore, the expected values for Model 2, 3, 4 are calculated as:

$$m_{ijk} = \frac{n_{ijp}*n_{ppk}}{n_{ppp}} \qquad m_{ijk} = \frac{n_{pjk}*n_{ipp}}{n_{ppp}} \qquad m_{ijk} = \frac{n_{ipk}*n_{pjp}}{n_{ppp}}$$

where n_{ijp} is the number of times tokens i and j occur in their respective positions, n_{pjk} is the number of times token j and k occur in their respective positions and n_{ipk} is the number of times that tokens i and k occur in their respective positions in the Ngram.

Table 19: Observed Values for “real estate agent”

		agent	¬ agent	Total
real	estate	171	3000	3171
real	¬ estate	2	20805	20807
¬ real	estate	4	2522	2526
¬ real	¬ estate	7157	88567875	88575032
Total		7334	88594202	88601536

Using the above expected value equations, we can calculate the expected values for the trigram “real estate agent” using the observed data from Table 19. The expected values for each of the different models (seen in Table 20, 21, 22, and 23) depend on their respective hypothesized models. G^2 is calculated for each of the models using different expected values which result in a different G^2 scores. For example, when comparing how often the trigram “real estate agent” actually occurred, to how often it would be expected occur given ”real”, ”estate” and ”agent” were independent for Model 1, ”real” and ”estate” were dependent and independent from ”agent” for Model 2, ”estate” and ”agent” were dependent and independent from ”real” for Model 3, and lastly ”real” and ”agent” were dependent and independent from ” estate” for Model 4.

Table 20: Expected Values for Model 1

		agent	¬ agent	Total
real	estate	0.0001	1.5416	1.5417
real	¬ estate	1.9846	23974.4735	23976.4582
¬ real	estate	0.4714	5694.9867	5695.4582
¬ real	¬ estate	7331.5437	88564530.9979	88571862.5417
Total		7334	88594202	88601536

Table 21: Expected Values for Model 2

		agent	¬ agent	Total
real	estate	0.2625	3170.7375	3171
real	¬ estate	1.7224	20805.2776	20807
¬ real	estate	0.2091	2525.7909	2526
¬ real	¬ estate	7331.8062	88567700.1938	88575032
Total		7334	88594202	88601536

Table 22: Expected Values for Model 3

		agent	¬ agent	Total
real	estate	0.0473	1.4944	1.5417
real	¬ estate	1.9374	23974.5208	23976.4582
¬ real	estate	174.9527	5520.5055	5695.4582
¬ real	¬ estate	7157.0626	88564705.4791	88571862.5417
Total		7334	88594202	88601536

4.2 Model Fitting

Model fitting involves determining which model best represents an Ngram. We know when a model is a good ’fit’ the observed values are close to the expected values. Therefore, if an Ngram has a lower G^2 score for a specific model compared to the rest of the models, that model with the lowest G^2 score best represents that Ngram.

Table 23: Expected Values for Model 4

		agent	¬ agent	Total
real	estate	0.0111	1.5306	1.5417
real	¬ estate	172.9889	23803.4693	23976.4582
¬ real	estate	0.4605	5694.9977	5695.4582
¬ real	¬ estate	7160.5395	88564702.0022	88571862.5417
Total		7334	88594202	88601536

For example, using the expected and observed values in the previous section for the trigram "real estate agent", the G^2 score for each of the four models can be seen in Table 24. The model with the lowest G^2 score is Model 2 which is based on the assumption that "real" and "estate" are dependent and independent from "agent". This result is reasonable because "real" with "estate" is describing "agent". Therefore, we could say that the trigram "real estate agent" is best represented by trigram Model 2.

Table 24: G^2 Scores for "real estate agent"

Model 1	46617.8291	Model 2	1904.0684
Model 3	44886.5300	Model 4	45408.7633

As the dimensions of a contingency table grows so does the number of hypothesized models. For trigrams and 4-grams, it is feasible to do an exhaustive search for the best model rather than using a search algorithm such as Forward or Backward Sequential Searching [23]. Therefore, rather than proceeding with a search algorithm, we simply iterate through every possible hypothesized model and do an exhaustive search to determine the 'best fitting' model.

4.3 Significance Testing

Significance testing can be used to assign significance values to G^2 scores based on the χ^2 distribution. G^2 follows a χ^2 distribution when the size of the corpus is large, the number cells in the contingency table is fixed and the expected values for each of the cells in the contingency table are large [22]. The significance values are used to establish a threshold in which ngrams above the threshold are accepted and ngrams below are rejected based on a null hypothesis;

To test the significance, a probability p is computed based on the null hypothesis being true, it is the probability of the observed values being greater than what the hypothesized model would predict. The $pvalue$ is set as a threshold to rejected the hypothesis if the probability is to low or accept if it is not. Typical significance levels for rejection are 0.01 and 0.05. Therefore if the G^2 score is above the threshold for a $pvalue = 0.05$, we are 95% certain that the hypothesized model does not represent our Ngram.

Typically significant Ngrams are identified by converting the G^2 score to a p value based on the χ^2 critical values found in the χ^2 distribution table and the number of degrees of freedom of the Ngram. The number of degrees of freedom indicate the number of values in a distribution that are independent of each other. There is one degree of freedom for each independent parameter in the model. The number of degrees of freedom for a model are used to refine the results of treatments of probability in determining statistical significance. It is dependent on what values are in our contingency table are independent and dependent. The number of degrees of freedom for a model can be calculated using Equation 7 [8].

$$df = \text{(Number of cells in table) - (Number of probabilities estimated for the hypothesis) - 1} \tag{7}$$

Using this definition, the number of degrees of freedom for trigrams under which the G^2 is calculated based on the model of independence (Model 1) is four (Equation 8). The number of degrees of freedom for models under which the expected values are estimated not based on independence are three as (Equation 9). A complete list of the number of degrees of freedom for each combination model for trigrams and 4-grams can be see in Table 25 and 26 respectively.

$$df = (rct) - (r-1) - (c-1) - (t-1) - 1 = 8 - 1 - 1 - 1 - 1 = 4 \tag{8}$$

$$df = (rct) - (r-1) - (ct-1) - 1 = 8 - 1 - 3 - 1 = 3 \tag{9}$$

Table 25: Number of Degrees of Freedom for Trigrams

Model 1 : $P(w1w2w3)/(P(w1)P(w2)P(w3))$	4	Model 2 : $P(w1w2w3)/(P(w1w2)P(w3))$	3
Model 3 : $P(w1w2w3)/(P(w1)P(w2w3))$	3	Model 4 : $P(w1w2w3)/(P(w1w3)P(w2))$	3

Moore [20] questions the use of this type of testing since the distribution of the data is Ziphian and may not be appropriate. It should be noted that he does not question the use of G^2 but the use of significant testing.

Table 26: Number of Degrees of Freedom for 4grams

Model 1: $P(w1w2w3w4)/(P(w1)P(w2)P(w3)P(w4))$	11
Model 2: $P(w1w2w3w4)/(P(w1w2)P(w3w4))$	9
Model 3: $P(w1w2w3w4)/(P(w1)P(w2w4)P(w3))$	10
Model 4: $P(w1w2w3w4)/(P(w1w3)P(w2w4))$	9
Model 5: $P(w1w2w3w4)/(P(w1)P(w2w3)P(w4))$	10
Model 6: $P(w1w2w3w4)/(P(w1w4)P(w2w3))$	9
Model 7: $P(w1w2w3w4)/(P(w1)P(w2w3w4))$	7
Model 8: $P(w1w2w3w4)/(P(w1w4)P(w2)P(w3))$	10
Model 9: $P(w1w2w3w4)/(P(w1w3)P(w2)P(w4)$	10
Model 10: $P(w1w2w3w4)/(P(w1w3w4)P(w2)$	7
Model 11: $P(w1w2w3w4)/(P(w1w2)P(w3)P(w4))$	10
Model 12: $P(w1w2w3w4)/(P(w1w2w4)P(w3))$	7
Model 13: $P(w1w2w3w4)/(P(w1w2w3)P(w4))$	7
Model 14: $P(w1w2w3w4)/(P(w1)P(w2)P(w3w4))$	10

5 Experimental Data

The algorithms in this thesis were evaluated using a subsection of the New York Times Newswire Service data available from the English Gigaword Corpus produced by the Linguistic Data Consortium. This English Gigaword Corpus is a comprehensive archive of newswire text data in English that come from four distinct international sources of English newswire: the Agency France Press English Service (AFE), the Associated Press Worldstream English Service (APW), The New York Times Newswire Service (NYT) and the Xinhua News Agency English Service (XIE).

Table 27: English Gigaword Corpus

Source	#Files	GB	Words	#Docs
AFE	44	1.2	170,969,000	656269
APW	91	3.6	539,665,000	1477466
NYT	96	5.9	914,159,000	1298498
XIE	83	0.9	131,711,000	679007

The text data are presented in SGML format where each file is compressed from about 3 MB (1995 Xinhua data) to about 30 MB (1996-7 NYT data) which equates to a range of about 9 to 90 MB when the data are uncompressed. A more detailed look at the size and words count can be seen in Table 27 where *Total MB* is the size of the data when the files are uncompressed, *Words* identifies the number of white space separated tokens after the SGML tags are removed. *#Docs* and *#Files* identify the number of documents and files each data source contains. It is noted in the English Gigaword documentation that the expected use for these files are as input to programs that are geared toward dealing with large quantities of data, for filtering, conditioning, indexing, and statistical summary.

For our experiments, we used the New York Times Newswire Service files nyt200101, nyt200102, nyt200103, nyt200104, nyt200105, nyt200106, nyt200107, nyt200108, nyt200109, and nyt2001010 where each file contains about ten million tokens totaling to approximately 100 million tokens.

5.1 Gold Standard

Evaluation of algorithms that automatically identify collocations is a difficult process. Daille [4] use a test bank approach where Ngrams are determined to be collocations if they exist in a pre-assembled test bank.

This option is infeasible in the general domain because there does not exist a complete list of collocations from the New York Times Newswire Service (NYT) data. Experiments when comparing results to the "compound words" that exist in Word Net did not return accurate results. There were many collocations that were tagged as false negatives because either they did not exist in Word Net of they were a variation of a collocation that did. For example, the collocations *Justice Clarence Thomas*, and *Prime Minister Thatcher* do not exist in Word Net and the collocation *President George Bush* is represented as *President Bush*.

Frantzi, Ananiadou and Hideki [11], Harris, Savova, Johnson and Chute [14] and Justeson and Katz [15] manually determine if an Ngram is a collocation or in their case a term which is a subset of collocations. Justeson and Katz [15] state that determining if an Ngram is a *collocation* is a subjective task that requires looking at the text to determine the author's intent. Therefore, we selected 250 Ngrams from the Gigaword Corpus NYT data and manually determined which Ngrams were collocations to provide a goldstandard to test our algorithm.

The 250 Ngrams were selected from the NYT data by first identifying all possible Ngrams using the Ngram Statistics Package (NSP). These Ngrams are then processed to extract only the Ngrams that occur more than once in the corpus and consist entirely of alpha characters. Given this processed set of Ngrams, we extracted 250 Ngrams using the following method.

The basic idea of the algorithm is to create a bucket for each Ngram that is to be extracted from the Ngram file. Fill each bucket is with $\frac{s}{n}$ Ngrams where s is the number of Ngrams that are in the Ngram file and n is the number of buckets that we are filling. An Ngram is then randomly extracted from each of the buckets ensuring that we obtain a uniform distribution of Ngrams from the file. The algorithm for this can be seen in Figure 6. We then manually identified the collocations from the set of Ngrams to create a goldstandard. To identify what Ngrams were collocations, we used the following definition: A collocation is a unit of words that exhibit non-compositionality, non-substitutablity and limited modifiability [19]. The trigram gold standard created using this method consists of 250 trigrams where 85 of those Ngrams are collocations. The 4-gram gold standard also consists of 250 4-grams where 52 are collocations.

Obtain Ngrams Algorithm

1. Initialize variables

a. Let n be the number of Ngrams we want to extract

b. Let s be the number of Ngrams in the Ngram file.

c. Let $bucket_size = \frac{s}{n}$

2. Foreach Ngram in the Ngram file

a. Store Ngram in an array

b. If the $arraysize = bucketsize$

i. Let $random$ be a random number

ii. Print the array element at index $random$

iii. Reinitialize the array

Figure 6: Obtain Ngrams Algorithm

5.2 Ngram Counts

The joint frequency and marginal values are needed for the Ngrams in the goldstandard in order to calculate the expected values required by the Log Likelihood measure. To calculate all the Ngrams and their counts from the corpus and then extract the Ngrams that exist in our goldstandard is a memory and time intensive process. Since we do not need all the Ngrams from the corpus, only a predetermined subset, we obtained these counts by extracting only the counts needed to calculate the marginal totals and the joint frequencies of the Ngrams in our goldstandard.

This algorithm, seen in Figure 7, was implemented by modifying the the count.pl Perl program in NSP. The count.pl program takes in a text file as input and obtains the Ngrams and their frequencies over the whole corpus. The basic idea to our approach is to preload two of the hash tables in the count.pl program with the Ngrams that we want to collect the counts over. The first hash table stores the Ngrams and collects their joint frequency counts and the second stores the marginals and collects their associated counts. The key to the algorithm is that it checks to see if the Ngram exists in its respective preloaded hash table before incrementing its counts. This allows only the Ngrams that we need to stay in memory while being able to

Find Ngram Algorithm

1. *Load the goldstandard Ngrams into Ngram table*
2. *Foreach goldstandard Ngram in the Ngram table*
    a. *Determine marginal Ngrams*
    b. *Load the marginal Ngrams in the marginal table*
3. *Foreach Ngram in the corpus file*
    a. *If Ngram exists in the Ngram table*
        i. *Increment the frequency for that Ngram*
    b. *Determine the marginal Ngrams for the Ngram*
    c. *If a marginal Ngram exists in the marginal table*
        i. *Increment the frequency for that marginal Ngram*

Figure 7: Find Ngram Algorithm

gather the counts over a larger data set.

6 Extended Log Likelihood Ratio

The Extended Log Likelihood Ratio (Extended G^2) is an extension of the Log Likelihood Ratio. This extension calculates the G^2 score for each possible hypothesized model (Section 4.1) of an Ngram. This approach then uses model fitting techniques (Section 4.2) to determine why hypothesized model best represents the Ngram. This allows us to determine how the tokens in the Ngram relate to each other so that we can determine whether it is a collocation.

6.1 Implementation of the Algorithm

The algorithm was implemented as a Perl module to be used with the statistics.pl program in the Ngram Statistics Package. The statistics.pl program takes as input an Ngram frequency file outputed by the count.pl program and calculates a given statistic. The statistic calculated is supplied as a Perl module that is dynamically loaded into the statistic.pl program.

We implemented two sets of modules to be used with the statistic.pl program; one for trigrams and the other for 4-grams. Each set contains two modules itself, the first module calculates G^2 for each of its hypothesized models. The second module in the set performs model fitting to determine which model best represents the given Ngram. These modules will be available in the Ngram Statistics Package.

6.2 Evaluation of the Algorithm

The Extended G^2 was evaluated by determining how well it identified collocations from a 250 manual tagged gold standard of trigrams and 4-grams described in Section 5.1 using the Ngram counts obtained from the English Gigaword New York Times AP Newswire (NYT) data as described in Section 5.2.

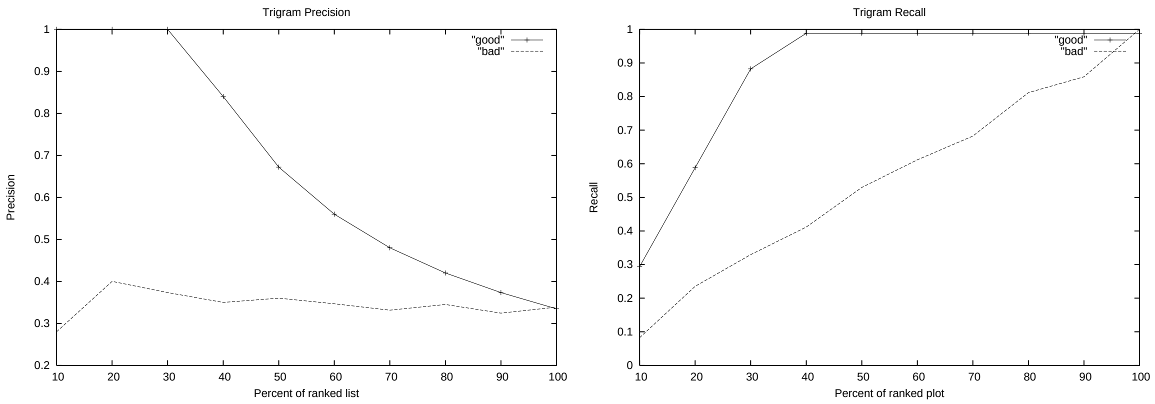
We evaluated our algorithm using the approach indicated by Wermter and Hahn [24]. The precision and recall were calculated at 10% increments on the list of ranked candidates and plotted on a graph for comparison. Precision is calculated by taking the ratio of the number of correctly identified collocations out of the total number of collocations identified (Equation 10). Recall is calculated by taking the ratio of the number of correctly identified collocations out of the total number of collocations given in the gold standard (Equation 11).

$$precision = \frac{\text{number of correctly identified collocations}}{\text{total number of collocations identified}} \tag{10}$$

$$recall = \frac{\text{number of correctly identified collocations}}{\text{total number of collocations in the gold standard}} \tag{11}$$

A non-optimal, “bad”, graph would be a relatively horizontal line indicating that the collocation were dispersed uniformly throughout the list rather than pushed towards the top. An optimal, “good”, graph for precision would be 100% precision until the percentage point in which there would optimally not be any more collocations is reached; at that point a sharp decrease would occur.

For our trigram data, there exists 85 collocations out of the 250 in our gold standard. A “good” graph for our trigram experiments would show that all the collocations from our list of Ngrams ranked by the extended G^2 algorithm would be ranked above the 30% point. Therefore in our precision graph, the ideal would be to see 100% precision until the 30% point and then a sharp decrease from that point on. The recall graph would be 100% recall until the 40% point after which we would see a sharp drop because by that point we would optimally have seen all possible collocations. The precision and recall graphs for the “good” and “bad” trigrams can be seen in Figure 8.



Precision Graph

Figure 8: Trigram Precision and Recall

There exists 52 collocations out of the 250 4-grams in our gold standard. A “good” graph for our 4-gram experiments would show that all the collocations from our list of Ngrams ranked by the extended G^2 algorithm would be ranked above the 10% point. Therefore in our precision graph, the ideal would be to see 100% precision until the 10% point and then a sharp decrease from that point on. The recall graph would optimally, start with a recall of 100% and then sharply drop at the 10% point because again we would optimally have seen all possible collocations by this point. The precision and recall graphs for the "good" and "bad" 4-grams can be seen in Figure 9

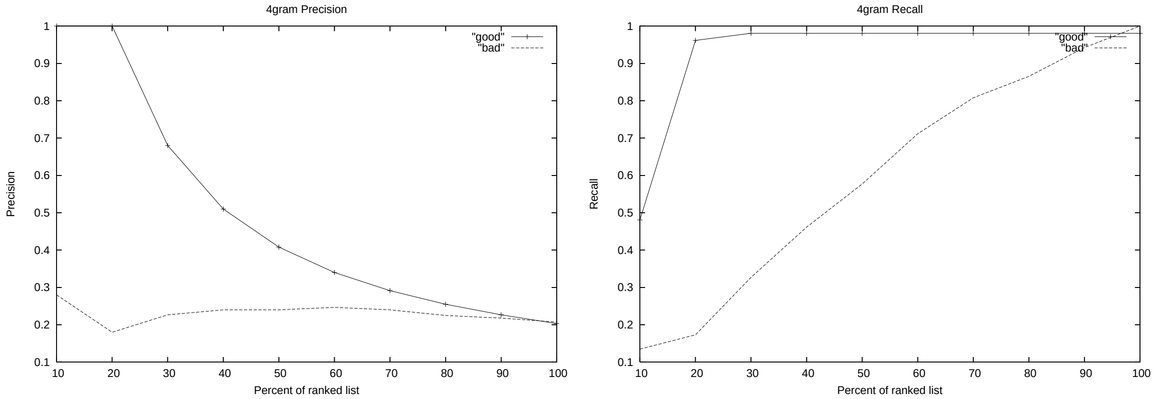


Figure 9: 4-gram Precision and Recall

We compared the extended G^2 algorithm with the frequency based approach, the standard G^2 approach and the C-value approach [10]. A frequency based approach ranks the Ngrams based on the number of times they occur in a text; this approach acts as a simple baseline. The standard G^2 approach that is described in Section 4. The C-value approach which is an algorithm proposed by Frantzi and Ananiadou [10] that identifies collocations and was later extended to the C/NC-value algorithm for term extraction.

6.2.1 Trigram Results

Figure 10 shows the precision and recall at each percentage point of the list of ranked trigrams using the extended G^2 approach. It can be observed that at the top 10% of the list the precision is at approximately 0.48 with a recall of 0.14. There is a sharp increase in these results at the 20% point with a precision of 0.56 and a recall of 0.32. From that point on we have a steady decline at each percentage point for the precision 10% quicker than hoped for and a rise at each point for the recall. The results are not as good as the optimal results but show that the collocations are being pushed towards the top of the list rather than dispersed randomly amongst the other Ngrams.

A comparison study was conducted on our frequency (baseline) approach of ranking the Ngrams based on the number of times they occur in the corpus. The precision and recall results, seen in Figure 11, initially show similar results to our extended G^2 approach by obtaining a precision of 0.48 with a recall of 0.14 at the 10% point. The precision of the frequency approach after this point drops showing a precision of 0.42 at the 20% point, 14% lower than the extended G^2.

The standard G^2 is calculated based only on the model of independence therefore we conducted a comparison study to determine how well our algorithm performed against this standard approach. The standard approach obtained a precision of 0.32 and a recall of 0.09 at the 10% point 16% lower accuracy than the extended G^2 approach as seen in Figure 12. At the 20% point the standard G^2 shows an increase in precision of 0.40 with a recall of 0.23 but still 16% lower than the extended G^2.

We compared the extended G^2 with the C-value approach proposed by Frantzi and Ananiadou [9] which is described in conjunction with the C/NC-value approach in Section 8. The C-value approach initially obtains a precision of 0.52 and a recall of 0.15 at the 10% point; 4% higher than the extended G^2 (Figure 13). At the 20% point, the precision of the C-value drops 12% lower than the precision of the extended G^2.

6.2.2 4-gram Results

Figure 14 shows the precision and recall at each percentage point of the list of ranked 4-grams using the extended G^2. It can be observed that at the top 10% of the list the precision is at approximately 0.44 with a recall of 0.21. A gradual decrease in precision continues from that point on. The recall gradually increases but never reaches a plateau. This is not as good as our optimal results but still better than our non-optimal. The recall gradually increases performing

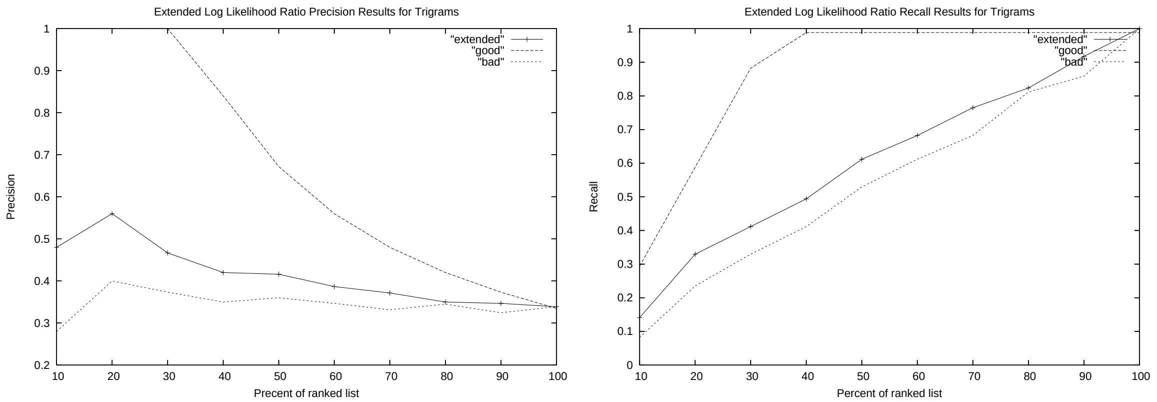


Figure 10: Trigram Precision and Recall for Extended G^2

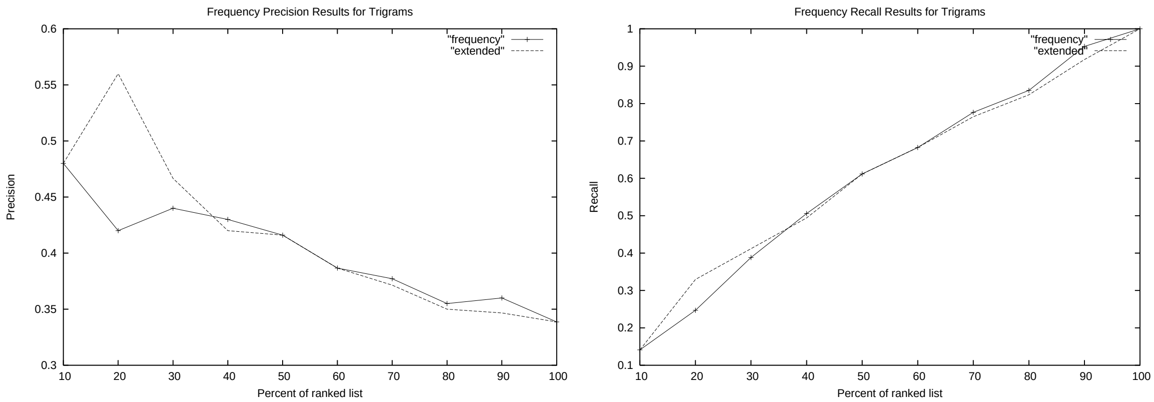


Figure 11: Trigram Precision and Recall for Frequency

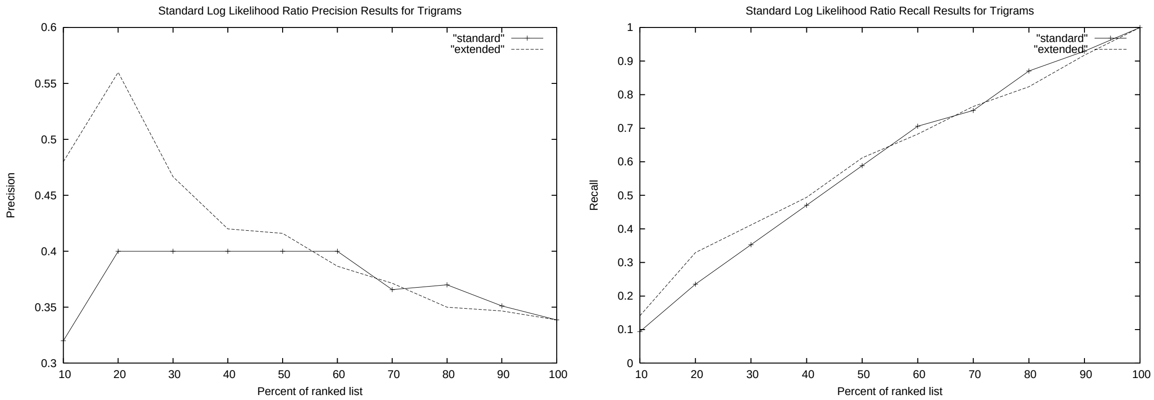


Figure 12: Trigram Precision and Recall for Standard G^2

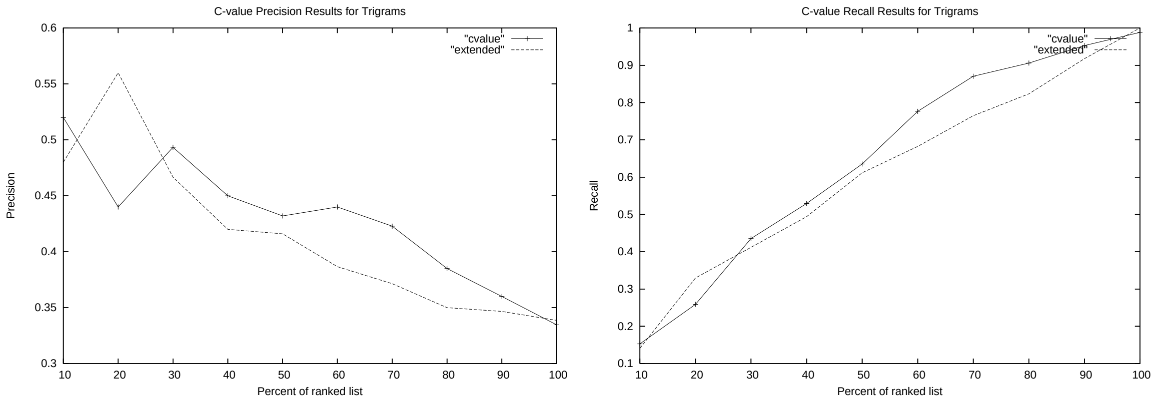


Figure 13: Trigram Precision and Recall for C-value

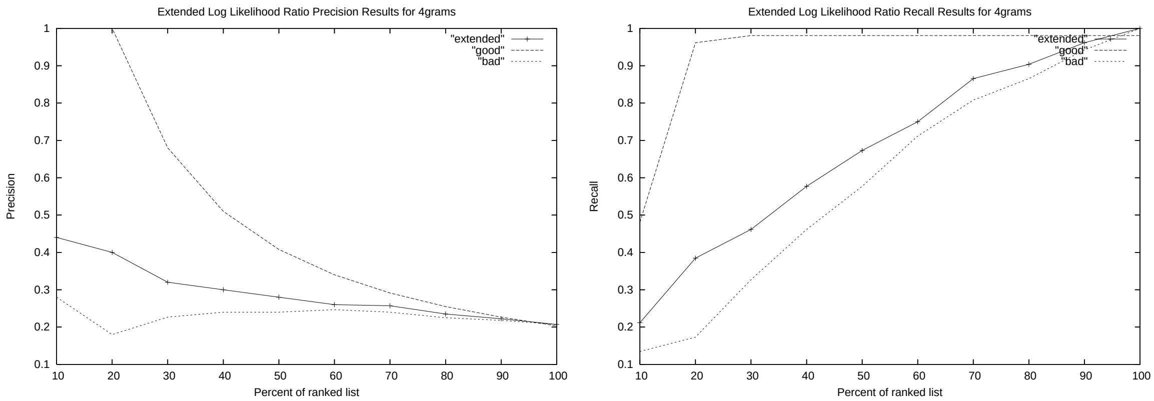


Figure 14: 4-gram Precision and Recall for Extended G^2

We compared the extended G^2 with the frequency approach, seen in Figure 11. The frequency approach obtained a precision of 0.40 with a recall of 0.19 at the 10% point which is lower than the precision obtained by the extended G^2 by 4%. The precision of the frequency approach drops showing a precision of 0.42 at the 20% point, 14% lower than the extended G^2 precision at this point. The standard G^2 performs worse than the extended G^2 with a precision of 0.32 at the 10% point; 8% lower than the extended G^2 at the same point. These results can be seen in Figure 16.

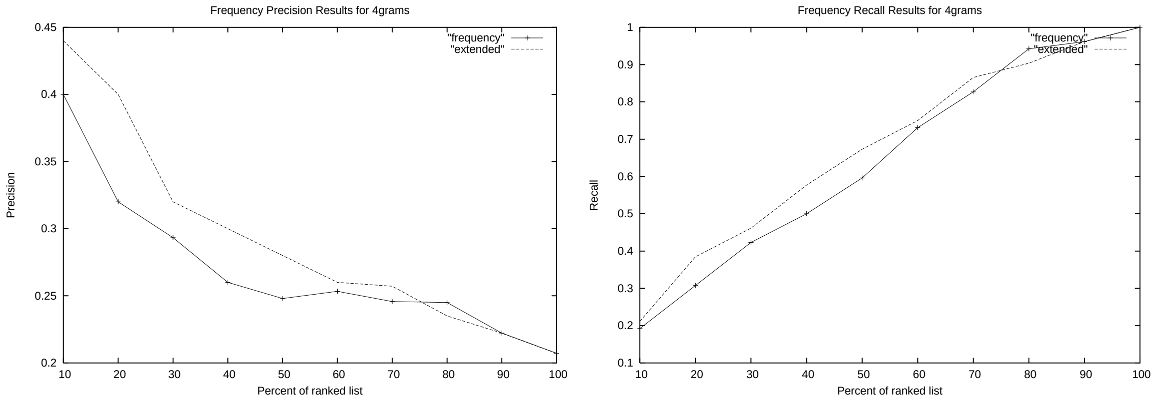


Figure 15: 4-gram Precision and Recall for Frequency

The C-value approach obtains a precision of 0.40 and a recall of 0.19 at 10% point, 4% worse than the precision of the extended G^2 at this point. The precision for this approach drops to 0.30 at the 20% point, 10% lower than the extended G^2 precision.

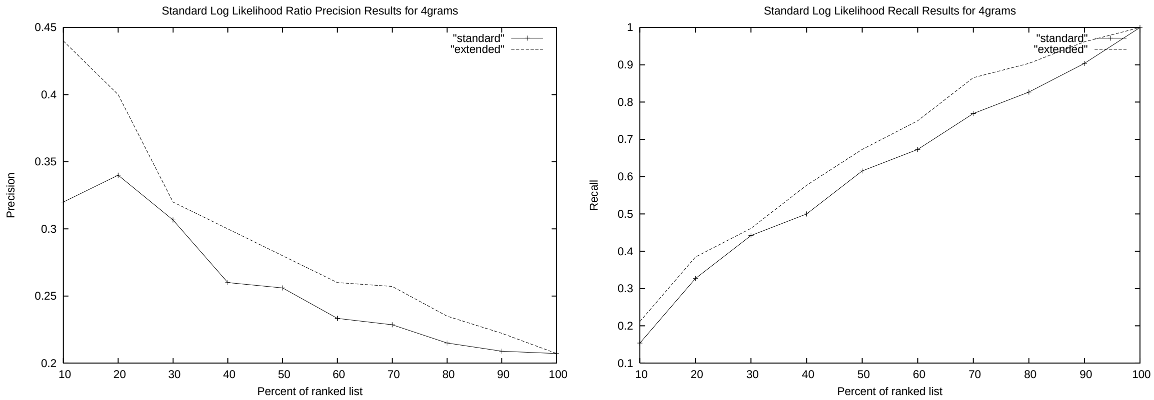


Figure 16: 4-gram Precision and Recall for Standard G^2

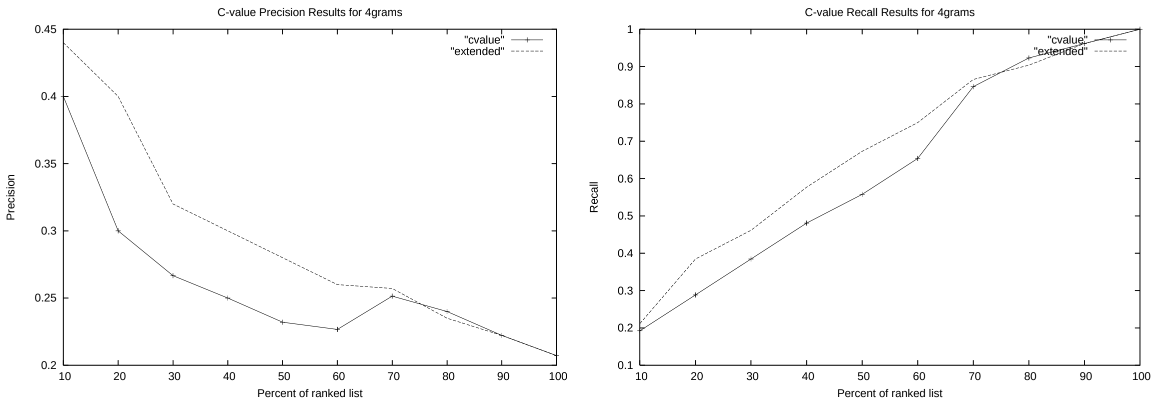


Figure 17: 4-gram Precision and Recall for C-value

6.3 Analysis of Results

To analyze our results, we calculated the F-measure for each of the results to obtain a single score to compare each of the different methods. The F-measure takes into consideration both the precision and the recall scores; it is the harmonic mean of precision and recall. The calculation of the F-measure can be seen in Equation 12.

$$F\text{–}measure = \frac{2 \times precision \times recall}{precision + recall} \tag{12}$$

Figure 18 shows the F-measure results for each of the methods used to identify collocation from trigrams and 4-grams. The results show that the extended G^2 performs better than the frequency and the standard G^2 for both trigrams and 4-grams. The trigram results for the C-value initially how a higher F-measure score than the extended G^2 but after the 20% point the extended G^2 outperforms the C-value approach. The 4-gram results show that the extended G^2 performs better overall than the C-value approach.

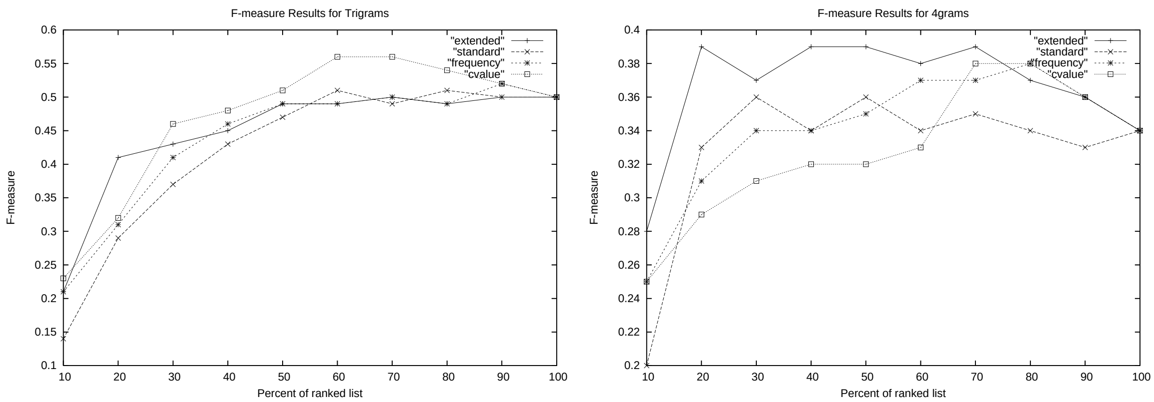


Figure 18: F-measure Results

7 Obtaining Ngram Counts from the Web

Traditional methods for obtaining frequency counts for Ngrams is to count the number of times they appear in a corpus. We discuss in a previous section different data structures and methods that can be used to obtain these counts. Obtaining the counts using these methods become very limiting because there always exists a finite number of Ngrams that we can maintain in memory. Calculating the counts for Ngrams over a large corpus is important for many algorithms because the accuracy under which they perform increases as the amount of data increases [2].

There has been increasing discussion of using the World Wide Web as a corpus [17]. This would increase the amount of available data and through the use of search engines such as Google and Alta Vista provide a memory efficient way to obtain frequency counts through the "hit count" that is returned with every query. The use of the hit count as joint frequency and marginal counts of bigrams have been discussed by Keller and Lapata [16] who showed that the hits counts obtained from the web for bigrams are correlated to the term frequency obtained from the British National Corpus. We suggest that this concept can be extended to gather the marginal counts as well as the joint frequency of trigrams and 4-grams from the web in order to perform the Log Likelihood Ratio (G^2).

7.1 Web Count Algorithm

The basic idea of the web count algorithm is for each Ngram in the input file first obtain the joint frequency by querying the Ngram itself and second to break the Ngram into its marginal sub-pieces and query them individually to obtain the marginal counts. For example, if we look at the trigram "New York Times", the query to obtain the joint frequency would be the trigram itself "New York Times". The queries to obtain the marginal counts would be "New York", "New * Times" and "York Times" respectively. As seen in Table 28, each query was placed in quotes and a $*$ was used as a place holder to obtain the marginal count where the tokens in the query are not contiguous.

The query "New * Times", obtains the count in which "New" is in the first position and "Times" is in the third position. Without the "*" as a place holder in the second position, the results returned by Google are those in which "New" and "Times" are contiguous. This can be observed by looking at the top five headline results returned by Google for each of these queries on June 15, 2004 in Table 29. The top five results for

Table 28: Marginal Counts for "New York Times"

Query	Frequency Count
"New York Times"	6,970,000
"New"	766,000,000
"York"	109,000,000
"Yankees"	123,000,000
"New York"	15,700,000
"New * Times"	8,090,000
"York Times"	7,150,000

the query "New Times" returned phrases in which the tokens "New" and "Times" were directly next to each other while the top five results for the query "New * Times" did not.

Table 29: Query Results using Google

Query "New Times"	Query "New * Times"
Phoenix New Times	The New York Times
New Times – newtimes.com	New York Times Learning Network
Miami New Times	New York Times Company
New Times	The New York Times News Services
Syracuse New Times Net	The New York Times Travel

There exists a difference in the hit count returned by Google for each of these queries as well. The hit count returned for the query "New * Times" is considerably higher than the count returned for the query "New Times". The hit count returned by the query "New * Times" more accurately corresponds to the hit count for the query "New York Times" because hit count for this marginal must be greater than or equal to the hit count return by the query "New York Times".

Table 30: Hit Count Returned by Google

Query	Hit Count
"New * Times"	8,090,000
"New Times"	601,000

7.2 Implementation of the Algorithm

The algorithm was implemented for the search engines Google and Alta Vista. The implementation using these search engines are similar, the differences is how the hit counts are returned. Each of the two packages have a Perl script that takes in an input file of Ngrams and calls a Perl module that accesses the appropriate search engine and returns the hit count for that Ngram. The program to obtain the Ngram counts using Google is called GoogleGrams.pl and AltaVistaGrams.pl to obtain the Ngram counts using Alta Vista. An Ngram file is read in by the program, for each of the different combination of the Ngram as described above the frequency count is obtained from the module accessor variable, *$handler*. The only difference between the Google and AltaVista programs is that the handler variable accesses. As seen in Figure 19 the call to getCount:

$$my\ \$frequency = \$handler \rightarrow getCount(\$marg);$$

will rely on what module the handler accessors refers to either the Google::Count.pm or the AltaVista::Count.pm module.

The Google::Count.pm Perl module is used to obtain the hit count from the Google search engine for a set query. This module requires the Google Web API service license, WSDL file and the SOAP::Lite Perl module in order perform automatic querying. The code for this module can be seen in Figure 20. The *$service* variable is an accessor to the SOAP::Lite module(or library) which initializes your contact with Google using the WSDL file as follows:

$service = SOAP::Lite → service('file:GoogleSearch.wsdl');

The $service variable can then be used to automatically query a search term or in our case an Ngram using the *doGoogleSearch* function from the SOAP::Lite module.

The AltaVista::Count.pm Perl module is similar in functionality to the Google::Count.pm module. It requires the LWP module to access the search engine. The *$browser* access variable sends the query to the search engine and put the results in the *$response* access variable in which the content of the query page can be accessed from. A search is done on the content of the returned page to retrieve the hit count of the query. This number is stripped of comma's and returned.

```
while(<FILE>) {
    print DST "$_<>"; Ngram = split/<>/;
    for (0..$#combination_array) {
        # get the combination
        my @combo = split/ /, $combination_array[$_]; my @marginal = ();
        # check if a split and set the combination array
        my $prev = $combo[0]; $marginal[0] = $Ngram[$combo[0]];
        # get the combination Ngram
        for (1..$#combo) {
            if($combo[$_] != $prev+1 ) { $marginal[$_] = "* " . $Ngram[$combo[$_]];}
            else { $marginal[$_] = $Ngram[$combo[$_]]; }
            $prev = $combo[$_];
        }
        my $marg = join " ", @marginal;                # set the combination Ngram
        my $frequency = $handler → getCount($marg); # get the marginal count
        print DST "$frequency ";                       # print to the destination file
    }
}
```

Figure 19: Basic Search Engine Code

```
# Google getCount Function
sub getCount {
    my $self = shift; my $word = shift; my $query = '"' . $word . '"';
    $result = $service → doGoogleSearch(
              $key,          # key
              $query,        # search query
              0,             # start results
              10,            # max results
              "false",       # filter: boolean
              "",            # restrict (string)
              "false",       # safeSearch: boolean
              "",            # lr
              "latin1",      # ie
              "latin1"       # oe
    );
    return ($result → estimatedTotalResultsCount);
}

# Alta Vista getCount Function
sub getCount {
    # code idea found in Perl & LWP
    my $self = shift; my $word = shift;
    # set the query and send it
    my $query = 'http://www.altavista.com/sites/search/web?q%22'' .$word.''%22&klXX';
    my $browser = LWP::UserAgent → new; my $response=$browser → get($query);
    # get the web page content and retrieve the document number
    my $webpage = $response → content;
    if($webpage = m/>AltaVista found (.+) results/ig) { $num = $1; }
    $num = s/,//g; return $num;
}
```

Figure 20: Google::Count.pm and AltaVista::Count.pm getCount Functions

Table 31: Ngram Counts

Marginal Ngram	Corpus Count	Google Count	Alta Vista Count
New York Times	32,169	3,060,000	37,700,000
New	73,461	323,000,000	981,000,000
York	59,042	46,500,000	165,000,000
Times	35,989	52,400,000	230,000,000
New York	58,489	7,110,000	158,000,000
New * Times	32,175	3,480,000	152,000,000
York Times	32,237	3,090,000	37,900,000

Table 32: Marginal Counts for "New York Yankees"

Query	nomenclature	Frequency Count
"New York Yankees"	$n111$	514,000
"New"	$n1pp$	323,000,000
"York"	$np1p$	46,500,000
"Yankees"	$npp1$	1,370,000
"New York"	$n11p$	7,110,000
"New * Yankees"	$n1p1$	526,000
"York Yankees"	$np11$	508,000

7.3 Evaluation of the Algorithm

To analyze the results obtained from the search engines, a comparison was conducted between them and the corpus counts. The marginal counts from the trigram "New York Times" returned by the New York Times AP Newswire (NYT) data obtained from the English Gigaword corpus, and the results returned by Google and Alta Vista can be seen in Table 31.

We found cases, when analyzing the counts, where the marginal counts returned by the search engine for the Ngram were sometimes lower than the hit count of the Ngram itself. This situation can not happen when obtaining the marginal counts for an Ngram from a corpus because the number of times that the first token in an Ngram is seen in the first position must be equal to or greater than the frequency count of the Ngram. This difficulty arose when looking at the marginal counts returned by the Google Search Engine for the trigram "New York Yankees". In Table 32, it can be seen that the marginal value from $np11$, which is the hit count for the query "York Yankees", is less than the joint frequency count, $n111$.

The total number of Ngrams that exist in Google and Alta Vista need to be estimated in order to apply statistical measures of association to the Ngram counts returned by these search engines. This was estimated by determine how many times the word "the" occurs in both the English Giga-Word corpus versus the hit count for the search engine. The word "the" occurs 2,590,480 times in New York Times Newswire files nyt200102 through nyt200106 totaling to set of 47,410,427 tokens. The query for "the" returned a hit count of 5,860,000,000 for Google and 562,012,313 for Alta Vista on 10 June 2004. Therefore, if we take the ratio of the number of times "the" occurs in the corpus and the number of tokens in the corpus; multiply that by the hit count for "the", we can get an estimate on the number of Ngrams that exist in each of the search engines. The Google search engines contains approximately 117,200,000,000 Ngrams and Alta Vista contains approximately 11,240,246,260.

The calculation of G^2 using the statistic.pl program returned two types of errors when using the frequency counts returned by the search engines. These errors returned the following warning messages:

1. Warning message: About to take log of negative value.
2. Warning message: Frequency value of Ngram must not exceed the marginal totals.

The first error occurred three times using the Alta Vista counts for trigrams and nine times for 4-grams. It occurred using the Google counts once for trigrams and six time for 4-grams. An example of the error message can be seen below for the trigram *these<>edgy<>unorthodox<>*.

Warning from statistic library!
Warning code: 242
Warning message: About to take log of negative value for cell (2,1,2)
Skipping Ngram these<>edgy<>unorthodox<>2450 418000000 1210000 933000 1220000 959000 2450

This error occurred when calculating the expected value $m212$ which is the number of times that "edgy" occurred in the second position. The corresponding observed value is a negative value therefore the check in our program to ensure that our observed values are not less than zero through an error. We can trace back through to see how this happened.

The marginal total are read in for the Ngram and stored in the following variables: $n111 = 2450$, $n1pp = 418000000$, $np1p = 1210000$, $npp1 = 933000$, $n11p = 1220000$, $np11 = 959000$, and $n1p1 = 2450$. The notation for these marginal counts are described in Section 2. From these values we can calculate all the

observed values for this Ngram: $n112 = n11p - n111 = 1220000 - 2450 = 1217550$
$n211 = np11 - n111 = 959000 - 2450 = 956550$
$n212 = np1p - n111 - n112 - n211 = 1210000 - 2450 - 1217550 - 956550 = -966550$

At this point we can see that the value for $n212$ is a negative number which is impossible. This arises from the hit count returned for $n112$ is larger than the hit count returned for $n1p1$. This is impossible because $n112$ is the number of times we see “these” and “edgy” occur in the first and second positions in an Ngram and $np1p$ is the number of times that “edgy” occurs in the second position of the corpus. The token “edgy” has to occur at least the same number of times alone as it does with another token. The other two Ngrams had similar errors.

The second error occurred twice for 4-grams when using the counts returned by Alta Vista and twenty times for 4-grams using the counts returned by Google and five times for trigrams. An example of this error can be seen below for the 4-gram *basic<>square<>foot<>price<>*.

Warning from statistic library!
Warning code: 202
Warning message: Frequency value of Ngram (522000) must not exceed the marginal totals.
Skipping Ngram basic<>square<>foot<>price<>522000 99400000 47200000 49300000 0 2980000 4190000 17100000 6960000 9950000 9350000 1080000 1460000 1440000 2020000

This error occurred because the joint frequency of the Ngram “basic square foot price” returned by the search engine with a frequency count of $522,000$ is larger than the frequency count of the marginal *price* which was zero. It is mathematically impossible to see “basic square foot price” $522,000$ times and then never see “price” in the fourth position because we know that we just say at least $522,000$ times.

7.4 Results

The counts retrieved from the search engines Google and Alta Vista were evaluated by determining how well the counts can be used to identified collocations using the standard G^2 based on the model of independence from the 250 manual tagged gold standard of trigrams and 4-grams described in Section 5.1. These results were compared to the results obtained using the corpus counts described in the Section 5.2.

The results are evaluated using the same technique in which we evaluated the Extended Log Likelihood Ratio in Section 6. The precision and recall were calculated at 10% increments on the list of ranked candidates and plotted on a graph for comparison between the web counts and corpus counts. Precision is calculated by taking the ratio of the number of correctly identified collocations out of the total number of collocations identified (Equation 13). Recall is calculated by taking the ratio of the number of correctly identified collocations out of the total number of collocations given in the goldstandard (Equation 14).

$$precision = \frac{\text{number of correctly identified collocations}}{\text{total number of collocations identified}} \tag{13}$$

$$recall = \frac{\text{number of correctly identified collocations}}{\text{total number of collocations in the gold standard}} \tag{14}$$

The results obtained using the corpus counts from trigrams show a precision of 0.32 and a recall of 0.09 at the 10% point seen in Table 21. The corpus counts increase in precision at the 20% point rising to 0.40 with a recall of 0.23. The results for the 4-grams show a precision of 0.32 at the 10% point but increase to 0.34 at the 20% point.

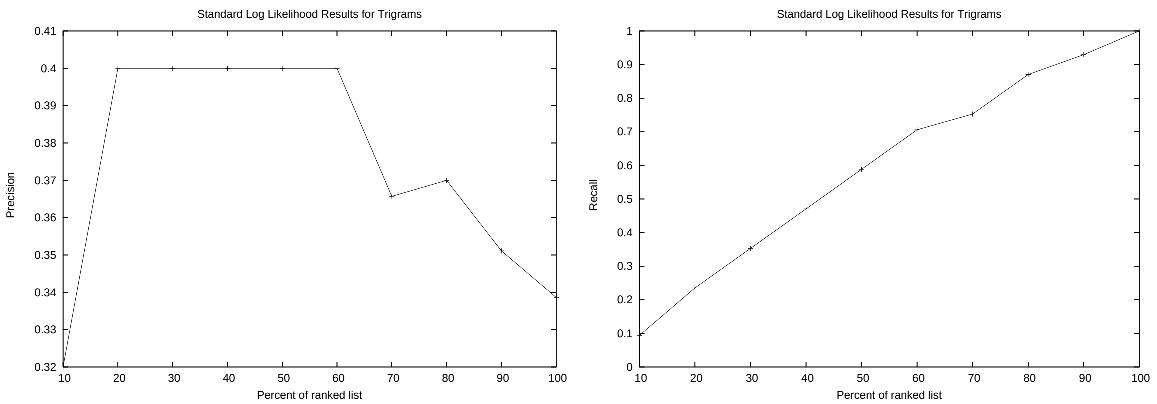


Figure 21: Trigram Precision and Recall for Standard G^2

7.4.1 Alta Vista Results

Figure 23 shows the precision and recall at each percentage point of the list of ranked trigrams using the standard G^2. It can be observed that at the top 10% of the list the precision is at approximately 0.28 with a recall of 0.08. The precision is 4% lower than the precision obtained using the corpus counts. There is a sharp increase in the precision for these results from 0.28 to 0.30 between the 10% and 40% point but the

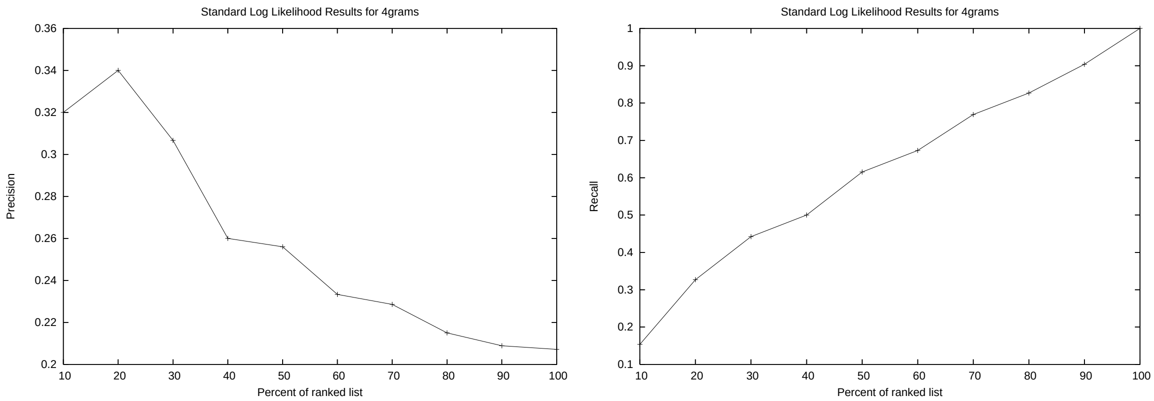


Figure 22: 4-gram Precision and Recall for Standard G^2

precision using the corpus counts remains higher. The G^2 did not identify the collocations using the Alta Vista counts as well as it did using the corpus counts.

The 4-gram results seen in Figure 24 confirm the results seen in the trigram data. The Alta Vista counts have slightly better 4-gram results than trigram results but still do not perform better than the corpus counts. The precision at the 10% point is 0.32 with a drop to 0.23 at the 40% point.

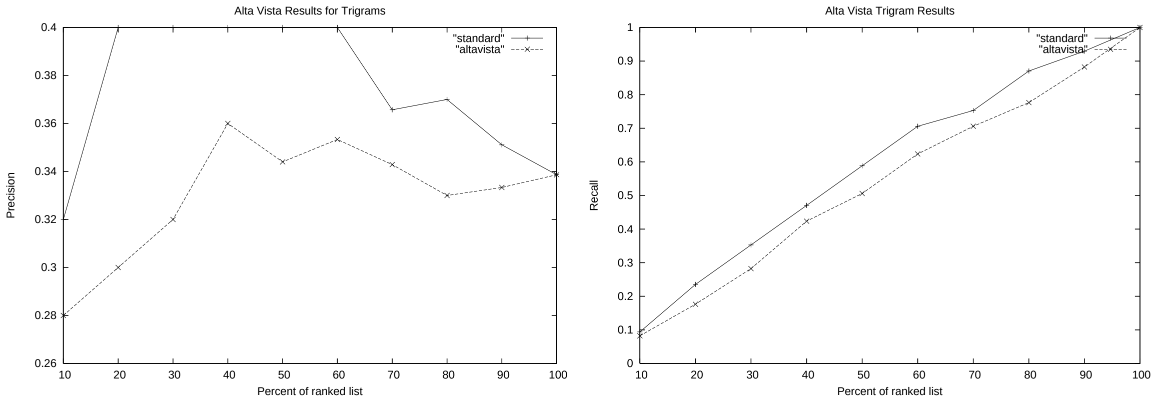


Figure 23: Trigram Precision and Recall for Alta Vista Results

7.4.2 Google Results

The results obtained using the hit counts returned by the Google show an improvement in precision and recall regardless of the counting discrepancies returned by the search engine. Out of the six Ngrams whose G^2 scores could not be calculated three of them were collocations. There is a 0.52 precision at the 10%

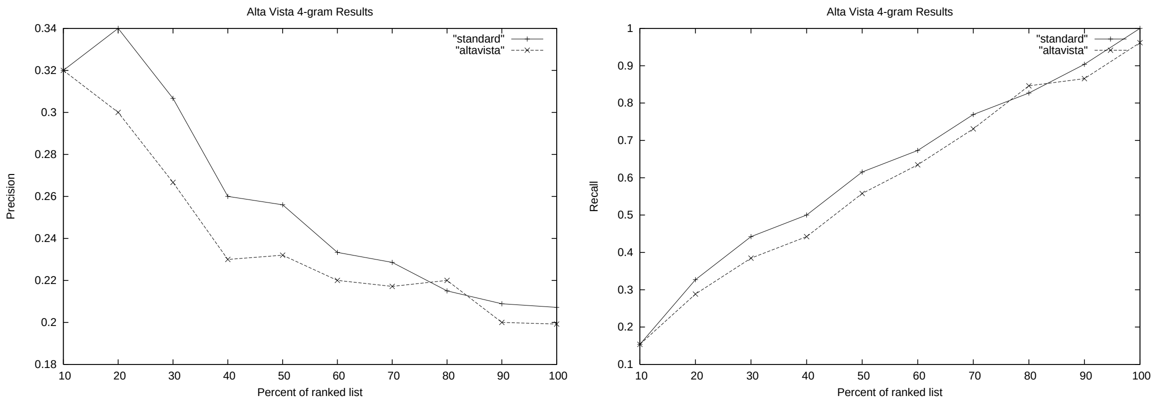


Figure 24: 4-gram Precision and Recall for Alta Vista Results

point, 20% greater than the precision obtained when using the corpus counts. There is a drop at the 20% point with a precision of 0.40 which is the same precision seen in the corpus counts at this point.

The google counts for the 4-grams do not show as significant improvement as with the trigram results but the precision at the 10% point is 0.32 the same as the corpus counts. The google counts see a rise in precision over the corpus counts at the 20% and 30% point and then returns to equal precision at the 40%.

The recall graph increases until it reaches the 60% point and then plateaus. This is because 26 collocations were not able to be processed due to the miscalculation errors described above.

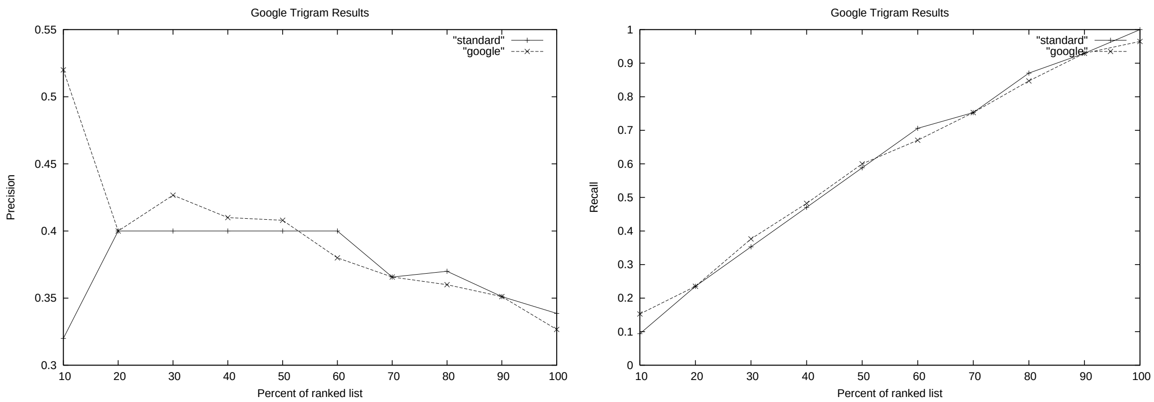


Figure 25: Trigram Precision and Recall for Google Results

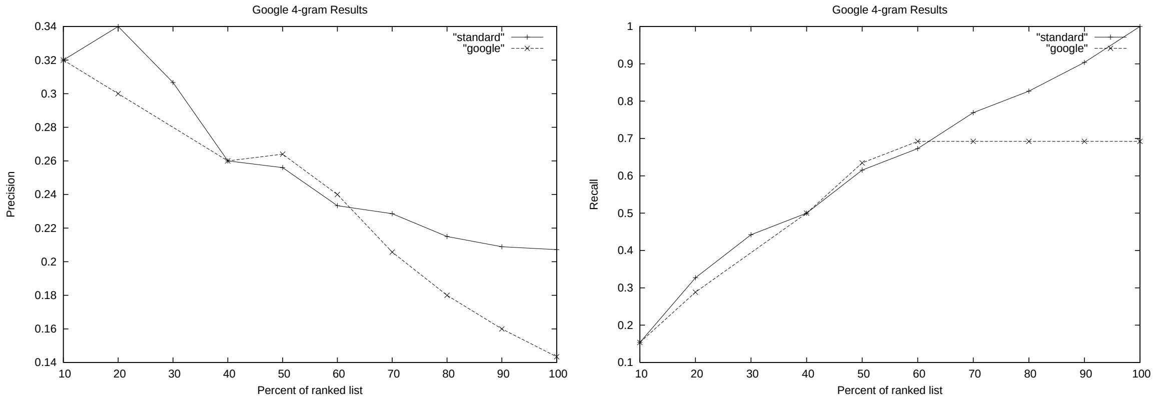


Figure 26: 4-gram Precision and Recall for Google Results

8 Related Work

The research in this paper is focused on obtaining Ngrams and their frequency counts from corpora and using the Log Likelihood Ratio (G^2) to identify collocations from the Ngrams. In this section, we discuss the ideas and algorithms that have been presented that use the World Wide Web to obtain the frequency counts of Ngrams. We will then discuss algorithms that have been proposed to identify collocations from a corpus using statistical methods that rely on these Ngram counts.

8.1 Methods for Obtaining Frequency Counts

Traditionally, the method used for obtaining frequency counts for Ngrams is to count the number of times they appear in a corpus and store them in a data structure. Various types of data structures to hold these Ngrams and their frequency counts have been discussed in Section 2 and 3. All these algorithms work to try to reduce the amount of memory needed to store the Ngrams in order to be able to work with a larger corpus. Obtaining Ngram counts in this fashion becomes very limiting because not only do we have memory constraints that make it infeasible to process large data sets, most data sets only consists of a few million tokens anyway. This problem has lead to increasing discussion on the feasibility of using the World Wide Web as a corpus [17]. The web consists of approximately 3,033 million pages which is an estimate by the Search Engine Showdown web site for Google. This dramatically increases the amount of freely available data to work with and through the use of search engines such as Google and Alta Vista there exists a memory efficient way to access this data.

Zhu and Rosenfeld [27] in **Improving Trigram Language Modeling with the World Wide Web** suggest that using the "hit" counts returned by a search engine could be used to calculate the probability of a trigram occurring in a corpus. Zhu and Rosenfeld [27] determined that the hit count returned by a search engine could be used as the joint frequency count of an Ngram. Log- linear regression was calculated for trigrams, bigrams and unigrams on the hit counts returned by the search engines Alta Vista, Lycos and FAST. The hits counts form the search engines and the frequency counts obtained from the Broadcast News corpus of a given set of Ngrams were compared. It was found that there exists a log-log correlation between the corpus and the web counts.

Keller and Lapata [16] propose using the web to obtain the frequency counts for bigrams that are not observed in their paper **Using the Web to Obtain Frequencies for Unseen Bigrams**. Bigram frequencies are obtained by conducting an exact match search of the bigram and adding up the number of pages that were returned to obtain the joint frequency. The pages themselves are not downloaded to count each bigram that occurs on the page but like Zhu and Rosenfeld, the page count returned by the search engine is used.

To use most measure of association for bigrams the marginal counts for the bigrams and the total number of bigrams that exist need to be determined. Obtaining the total number of bigrams that exist in Google and Alta Vista was estimated based on the counts seen in a corpus. The British National Corpus (BNC) contains 100 million tokens and Alta Vista counts are between 550 and 691 times larger than BNC and Google counts are between 1065 and 1306 times larger than BNC. These counts were used to estimate the number of words on the web resulting in 55.0 to 69.1 billion words for Alta Vista and 106.4 to 139.6 billion words for Google. The authors note that these estimates are in the same order of magnitude as the estimates made by Zhu and Rosenfeld [27].

Keller and Lapata [16] note the concerns that exist due to the noisiness of the web data. Punctuation and capitalization are removed even if the search term is in quotes and false positive counts are often generated because there does not exist any parsing, tagging or chunking to ensure that the query match is actually correct. It was also observed that Google returns pages that do not contain the query term but have a link to the page that term exists on. It was concluded though that although the web counts are noisy,the advantage of the large amounts of data available on the web outweighs the disadvantages that are associated with a noisy corpus.

To determine if the web counts and corpus counts were correlated observed and non-observed bigrams were extracted from the North American News Text Corpus (NANTC) and the British National Corpus (BNC). It was found that Google and Alta Vista counts correlate with NANTC counts for both the observed and unobserved bigrams. They ranged for Alta Vista between .667 and .788 and for Google between .662 and .787. The BNC was slightly higher than the correlation between both the web counts and NANTC counts. By using the t-test though, Keller and Lapata found that difference in correlation coefficients was not significant for either search engine.

8.2 Methods for Extracting Collocations

Research in collocation extraction is commonly broken into three areas: computational methods, linguistic methods and hybrid methods. This research is focused primarily on computational methods that have been used to try and solve this problem. Computational methods involve using association measures to try and determine the relatedness of the words in the collocation. There exists three types of computational methods that have been used. The frequency measure, the information- theoretic measures and the statistical measures [24]. The focus of this research is to identify collocations by analyzing trigrams, and 4-grams, hence the discussion in this section focuses on measures that either have been used or can be extended to identify these collocations from Ngrams where $n \geq 2$.

The **Word Association Norm, Mutual Information, and Lexicography** by Church and Hanks [3] propose the statistical measure *Pointwise Mutual Information* (PMI) to determine the association between two words based on how many times they are seen together in a corpus. For example, some common words that co-occur with $bank$ are $money$, $loans$, and $account$. PMI determines their association with each other based on mutual information (MI). Church and Hanks state that the technical definition of mutual information according to Fano [9] is :

$$I(x, y) = log_2 \frac{P(x, y)}{P(x)P(y)} \tag{15}$$

where x and y are two words with the probabilities $P(x)$ and $P(y)$. Church and Hanks [3] use this measure to compare the probability of the words, (x, y), in an n-gram occurring together versus the probability of the words occurring independently. If an association exists between x and y, then the joint probability will be larger than the independent probabilities. This equation is calculated by estimating the probabilities of $P(x)$ and $P(y)$ by counting the number of times x and y occur in a corpus and dividing each of the respective counts by the size of the corpus. The joint probability, $P(x, y)$, is estimated by determining the number of times x occurs in the corpus followed by y and dividing it by the size of the corpus. This statistic can then be re-defined as follows:

$$PMI = log \frac{n_{11}}{m_{11}} \tag{16}$$

where n_{11} is the known joint frequency, and m_{11}, is the expected joint frequency assuming independence.

The differences between PMI and MI is that the joint probabilities between two tokens, x and y, are assumed to be equal, $p(x,y) = p(y,x)$. This is not the case when estimating PMI due to the fact that the $freq(x,y)$ is the number of times that you observe x followed by y therefore the $freq(x,y)! = freq(y,x)$. Another difference is that it will not always be the case that $freq(x,y) < freq(x)$ and $freq(x,y) < freq(y)$ when identifying positional Ngrams. This is because there could be several occurrences of x and y in a given window.

Church and Hanks show that this measure can be used to identify a semantic relations between words for example *bread and butter*, *United States*, as well as semantic relations such as *doctor nurse*.

The PMI measure was extended to identify collocations that consisted of three words as well as two by Church and Yamamoto in **Using Suffix Arrays to Computer Term Frequency and Document Frequency for All Substrings in a Corpus** [26]. They show how the PMI measure can be extended as well as introduce a new measure, Residual Inverse Document Frequency (RIDF).

The PMI measure is an extension of the measure previously introduced by Church and Hanks (discussed above) in order to identify the association between units of words that contain more than two words. PMI is redefined for this purpose as:

$$PMI(x\mathbf{yz}) = \mathbf{log}\frac{\mathbf{p(xyz)}}{\mathbf{p(xy)p(z|y)}} \tag{17}$$

where x**yz** is the Ngram, x and z are the first and last tokens in the Ngram and **y** is the tokens between the first and the last.

RIDF is a statistical measure based on the Inverse Document Frequency (IDF) of an Ngram. This measure identifies Ngrams whose observed IDF is larger than the expected IDF score based on the assumption that the tokens in the Ngram are independent. This measure is defined as

$$RIDF = -log(df/D) + log(1 - exp(-tf/D)) \tag{18}$$

where df is the document frequency of the Ngram, tf is the joint frequency and D is the total number of documents. This measure weights term frequency over document. If the same number of Ngrams are seen over a smaller number of documents the RIDF score will be higher. For example, assume that we have an Ngram with the term frequency of ten, a document frequency of three will result in a lower RIDF score than a document frequency of one.

It was observed that although The measures are similar in the sense in how they take the log of the ratio between the empiricle and chance based estimation, but the types of collocations that they identify are different. PMI measure identifies general collocations similar to the type that would be found in a dictionary while the RIDF identified key words that were not typically found in a dictionary, such as *the joint commission* and *the kibbutz*. Even though the measures have little correlation with each other, Ngrams that have both a high PMI and RIDF score tend to be significant collocations while Ngrams with a low PMI and RIDF score tend to be insignificant, meaning their occurrence together is close to random.

The **Study and Implementation of Combined Techniques for Automatic Extraction of Terminology** by Daille [4] conduct a comparative analysis between Pointwise Mutual Information [3], Log Likelihood ratio [7], the Φ^2 Coefficient and frequency value was conducted for identifying collocations.

Daille [4] proposes the extraction of collocations from corpora using a combination of linguistic and statistical approaches. The linguistic approach, which is applied first, consists of a linguistic filter used to determine possible candidates that may be collocations based on their part-of-speech. The linguistic filter selects the candidates from a corpus by extracting Ngrams from a corpus that have one of the following part-of-speech patterns: (noun adjective), (noun determiner noun), (noun preposition noun) and (noun noun). The corpus was tagged using the stochastic tagger and lemmatizer developed by the Scientific Center of IBM-France.

Following the linguistic approach, a statistical approach was used to identify the collocations from the Ngrams that were extracted by the linguistic filter. Three measures of association and the simple frequency counts were compared to determine which measure was the most optimal for identifying collocations as well as how they compared with using just the raw frequency counts of the Ngrams to identify collocations. The measures of association that were used were PMI [3], G^2 [7] and the Φ^2 Coefficient. The PMI measure is defined in Equation . The G^2 measure is defined as

$$LL = 2 * \sum_{i}^{j} n_{ij} * log(n_{ij}/m_{ij}) \tag{19}$$

where n_{ij} are the observed frequencies of the Ngram and m_{ij} is the expected frequencies of the Ngram assuming that the Ngram is independent. The Φ^2 Coefficient is defined as

$$\Phi^2 = \frac{(n11 * n22 - n21 * n12)^2}{n1p * np1 * np2 * n2p} \tag{20}$$

where $n1p$, $np1$, $np2$ and $n2p$ are the marginal totals of the Ngram and $n11$, $n12$, $n21$, and $n22$ are the observed frequencies of the Ngram. This notation is described in more detail in Section 2.2.

The measures of association were evaluated by comparing the collocations extracted by each measure with terminology data bank of the same domain as the source corpus. It was found that the collocations identified by using the frequency of the Ngram were the more significant than the multi- word units identified by the statistical measures of association. This occurrence is because the higher the frequency of an Ngram the more probable it is that the Ngram was a collocation. The problem that arises with using only the frequency count is that a greater number of false positives are also found. The G^2 score was found to achieve the highest results because of its ability to obtain collocations while filtering out false positives. The Φ^2 Coefficient and Pointwise Mutual Information were found not to perform as well as the mere frequency counts.

Moore [20] discusses the use of statistics that come from significance testing for Natural Language Processing (NLP) tasks such as collocation identification in **On Log Likelihood-Ratios and the Significance or Rare Events**. Typically there exists very few words that occur many times in a corpus and many words that occur very few times. Statistical measures that arise from significance testing are noted to be unreliable for expected frequency less than five which is common for most existing corpora due to the sparseness of the data. They are commonly used in conjunction with significance testing to identify a threshold where anything above the threshold is accepted and anything below is rejected.

The G^2 comes from significance testing but Moore shows that this measure can still be applicable for NLP tasks because the G^2 measure is nearly equivalent to mutual information. This near equivalence allows us to use this measure to determine the association between words rather than the significance. Therefore, whether or not significance testing is appropriate for NLP tasks is independent from whether G^2 is an appropriate measure to use.

The **Automatic Recognition of Multi-Word Terms: the C-value / NC-value Method** by Frantzi, Ananiadou and Hideki [11] discuss a combined approach using linguistical and statistical knowledge to identify multi-word terms called the *NC-value* which incorporates the *C-value* approach.

The *C-value* algorithm has a two step approach, first identify possible multi-word candidates using linguistic information and second use statistical information of to identify the multi-word terms from the list of candidates. The linguistic approach consists of three steps: part-of-speech tagging, applying a linguistic filter to extract Ngrams that have a specific part- of-speech pattern and removing Ngrams that contain words in which are not expected to occur in the muti-word terms. The three linguistic filters used extracts Ngrams that have one of the following three part-of-speech patterns:

1. $Noun + Noun$
2. $(Adj|Noun) + Noun$
3. $(Adj|Noun) + |((Adj|Noun) * (Prep)?(Adj|Noun)*)Noun$

The third step involves applying a statistical measure, *C-value* to the extracted Ngrams. The statistic measure is defined as

$$C - value(a) = \begin{cases} log_2|a| * f(a) & \text{if a is not nested} \\ log_2|a| * f(a) - \frac{1}{P(T_a)} \sum_{b \epsilon T_a} f(b) & \text{otherwise} \end{cases}$$

where a is the candidate Ngram,
$f(.)$ is the frequency of . in the corpus,
T_a is the set of candidate Ngrams that contain a,
$P(T_a)$ is the number of candidate Ngrams that contain a.

The filters are analyzed individually and compared to the results obtained using the raw frequency counts of the Ngrams. The analysis of this shows that that the all filters obtain a higher precision than the raw frequency counts but the second filter obtained the best results. Analysis between the *C-value* and frequency show that the *C-value* algorithm identifies a greater number of multi-word terms therefore putting a greater number of multi-word terms towards the top of the list of candidate terms.

The *NC-value* algorithm extends the *C-value* algorithm by incorporating context information to extract multi-word terms. This algorithm is divided into three steps. The first step is to use the *C-value* algorithm to identify candidate multi-word terms. The second stage involves the extraction of the context words and calculating their weights. Context words are words that are within a specified window of a multi-word term. These words are extracted by identifying verbs, nouns and adjectives that are within the vicinity of 60 of the top 200 terms determined by the *C-value* measure. The 60 terms are determined by creating an ordered list the 200 terms by their *C-value* score and selecting 20 terms from the top, middle and bottom sections of the list. The weights that are associated with each context word are calculated by taking the quotient of the number of terms the context word appears with and the total number of terms that exist.

The third step is the actually calculation of the *NC-value* which is defined as:

$$NC - value(a) = 0.8 * C - value(a) + 0.2 * \sum_{b \epsilon C_a} f_a(b) weight(b) \tag{21}$$

where a is the possible multi-word term
C_a is the list of context words for a
b is a context word from C_a
$f_a(b)$ is the frequency of context word b
$weight(b)$ is the weight of context word b

The weights of 0.8 for the *C-value* and *NC-value* are assigned through experimentation by the authors because it gave the best distribution of the precision of the extracted terms. The experimental results showed that the *NC-value* algorithm performed with a higher precision than the *C-value* approach, increasing the number of actual multi-word terms at the top of the list.

The *C/NC-value* approach is designed to identify terms in a corpus but it is important to look at the calculation of the the *C-value* measure because it was noted that it can be used to identify collocation. The **Extracting Nested Collocations** by Frantzi and Ananiadou [10] use a slight variation of this measure to extract collocations from the Wall Street Journal newswire corpus in 1996 therefore we chose to discuss the C-value approach that was discussed in **Automatic Recognition of Multi-Word Terms: the C-value/NC-value Method**.

9 Future Work

It has been shown that the Log Likelihood Ratio (G^2) can be extended to improve the accuracy for identifying 3 and 4-dimensional collocations for general English by incorporating different hypothesized models. Our algorithms best precision is within the range of 0.45 and 0.55. This is better than the precision that was achieved by the frequency and standard G^2 and C-value approach. We have also shown that it is feasible to obtain Ngram counts using the "hit counts" obtained from the search engines Google and Alta Vista. There exists some issues that need to be addressed in order to make this a flawless system but the results shown are promising. Due to this, we believe there is sufficient reason to look more closely at extending some of these ideas.

9.1 The Extended Log Likelihood Ratio

We have hypothesized that incorporating lexical information such as part-of-speech to filter out Ngrams that do not have the correct syntactic pattern to be a collocation would improve the precision results. This was done by Daille [4] who initially extracted Ngrams that conformed to a specific part-of-speech pattern and then applied statistical methods to identify collocations. Frantzi and Ananiadou [10] use a similar technique by extracting noun phrases from a corpus to filter out unwanted Ngrams and then perform their C-value measure to extract the collocations.

We also feel that this approach could be extended to extract 5-dimensional collocations and then possibly n-dimensional collocations. The number of possible hypothesized models for an Ngram increases exponentially as the dimensions of the Ngram increase, for example there exists 56 different models for the 5-dimensional case. Iterating through each of these different models to find the best fit is still possible but as n grows it becomes less feasible to do this. It may be possible to incorporate sequential model searching [23] using either Forward or Backward Sequential Search in order to identify the "best fitting" model with out iterating though every possible choice.

We would like to apply the extended G^2 to identify collocations in clinical notes to see how well the measure would work in the medical domain. Identifying these collocation can be used for applications such as the automatic building of lexicons and knowledge bases which are being built on a tri-yearly bases due to the constant increasing vocabulary in this domain.

9.1.1 Applying the Extended Log Likelihood Ratio

We feel that this algorithm could be applied to help identify structural ambiguities in noun phrases. For example, the noun phrases *congestive heart failure* can have at least two interpretations:

1. [[congestive heart] failure]
2. [congestive [heart failure]]

The first interpretation shows *congestive heart* modifying *failure* while the second interpretation shows *congestive* modifying *heart failure*. G^2 could calculated on the Ngram for each of the hypothesized models and then using model fitting techniques, we could determine which model best represents the Ngram.

We also feel that this approach could be used to identify the appropriate parse structure of a sentence by performing the extended G^2 on the head of the constituents returned by a shallow parser. The shallow parser returns base level constituents of a sentence, for example, if we had the sentence: *the cart the horse pulled broke*. A shallow parser would identify the underlying constituents in the sentence as such: *[NP the cart] [NP the horse] [VP pulled broke].*

By obtaining the frequency counts of the heads in each of the phrases, we could obtain the G^2 score for *cart horse pulled* based on each of the hypothesized models and then using model fitting to determine which model best represents the parse structure.

9.2 Obtaining Counts from the Web

We have also shown that obtaining using the "hit counts" from search engines such as Google and Alta Vista in order to use Ngram statistics is a feasible option. The results obtained from the Google search engine have a higher precision that the results obtained from the corpus counts. A problem arises for some of the Ngrams due to rounding errors when calculating G^2 using the Google counts. We would like to identify an approach that would obtain consistent marginal values in order to calculate reasonable expected values.

10 Conclusion

The overall goal of this thesis was to automatically identify collocations from a text using an extension of the Log Likelihood Ratio (G^2).

To reach this goal, we needed to step back and look at how to efficiently determine Ngrams from a corpus, and obtain the frequency counts needed to perform statistical measures to identify these collocations. Therefore, our first objective was to identify an efficient way to obtain Ngrams from a corpora and determine their frequency counts. We have discussed different data structures that could be used to obtain these results from a corpus. We have also discussed using the World Wide Web as another means to obtain these frequency counts.

Using the "hit counts" returned by the search engines Alta Vista and Google; interesting results were attained. These counts were evaluated to determine how well they identified collocations by using them to calculate the G^2 for a set of Ngrams. We compared the results using counts obtained from the search engines with counts obtained from a corpus. The results showed that using the Google counts resulted in a higher precision than both the corpus counts and the Alta Vista counts. The Alta Vista results did not show any improvement in using these counts over the corpus counts. Due to these results, we believe that there is merit in using the hit counts returned by Google in replace of corpus counts. We need to investigate though how to resolve the rounding errors that occurred using the web counts.

The second objective was to identify collocations using an extension of the G^2. This approach showed an overall improvement over using the frequency, standard G^2 and C-value approach [11]. The standard G^2 performed approximately 10% worse when evaluating the trigrams and 4-grams. The frequency performed slightly better than the standard G^2 approach but still 10% worse than the extended G^2. The extended G^2 approach performed slightly better overall for 4-grams and either better of equivalent for trigrams. We believe that the extended G^2 approach offers a significant improvement in identifying collocations from a corpus over the other approaches.

The accomplishment of our objectives resulted in our achieving our overall goal of automatically identifying collocations from a text using an extension of G^2. Through this process, we were able to extend the standard G^2 to evaluate trigrams and 4-grams. We have defined an schema to evaluate how well the approach works. We have also extended and evaluated various data structures that can be used to store and analyze trigrams and 4-grams. Lastly, have proposed an approach to collect frequency counts for various size Ngrams using the World Wide Web.

References

[1] S. Banerjee and T. Pedersen. The design, implementation, and use of the Ngrams Statistic Package. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pages 370–381, February 2003.

[2] Eric Brill. Some advances in transformation-based part of speech tagging. In *National Conference on Artificial Intelligence*, pages 722–727, 1994.

[3] K. Church and P. Hanks. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16:1:22–29, 1991.

[4] Béatrice Daille. Study and implementation of combined techniques for automatic extraction of terminology. In Judith Klavans and Philip Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 49–66. The MIT Press, Cambridge, Massachusetts, 1996.

[5] Lopes J. Dias G. and Guillore S. Mutual expectation a measure for multiword lexial unit extraction. In *Proceedings of VEXTAL'99*, Venezia, San Servolo, 1999.

[6] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26:22–29, 1945.

[7] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

[8] B. Everitt. *The Analysis of Contingency Tables*. Chapman and Hall Ltd, London, England, 1977.

[9] R. Fano. *Transmission of Information*. The MIT Press, Cambridge, MA, 1961.

[10] K. Frantzi and S. Ananiadou. Extracting nested collocations. In *International Conference On Computational Linguisitics Proceedings of the 16th conference on Computational linguistics*, Copenhagen, Denmark, 1996.

[11] K. Frantzi, S. Ananiadou, and M. Hideki. Automatic recognition of multi-word terms: the c-value/nc-value method. *International Journal on Digital Libraries*, 3(2):115–130, 2000.

[12] A. Gil and G. Dias. Using masks, suffix array-based data structures and multidimensional arrays to compute positional ngram statistics from corpora. In *Workshop of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, 2003.

[13] G. Gonnet. Unstructured data bases or very efficient text searching. In *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 117–124. ACM Press, 1983.

[14] Marcelline R. Harris, Guergana K. Savova, Thomas M. Johnson, and Christopher G. Chute. A term extraction tool for expanding content in the domain of functioning, disability, and health: proof of concept. *J. of Biomedical Informatics*, 36(4/5):250–259, 2003.

[15] John S. Justeson and Slava M. Katz. Technical terminology: Some linguistic properties and an algorithm for identification in text. *Natural Language Engineering*, 1:9–27, 1995.

[16] Frank Keller and Mirella Lapata. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29:459–484, 2003.

[17] A. Kilgarriff and G. Grefenstette. Introduction to the special issue on web as corpus. Technical Report ITRI-03-20, Information Technology Research Institute, University of Brighton, 2003. Also published in Computational Linguistics 29(3), pp.1-15.

[18] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. In *In Proceedings of The First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327, 1990.

[19] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

[20] Robert C. Moore. On log-likelihood-ratios and the significance of rare events. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 333–340, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[21] F. J. Och and H. Ney. Improved statistical alignment models. In *ACL00*, pages 440–447, Hongkong, China, October 2000.

[22] T. Pedersen, M. Kayaalp, and R. Bruce. Significant lexical relationships. Technical Report 96-CSE-03, Southern Methodist University, February 1996.

[23] T. Bruce R. Pedersen and Wiebe J. Sequential model selection for word sense disambiguation. In *the Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, DC, April 1997.

[24] J. Wermter and U. Hahn. Collocation extraction based on modifiability statistics. In *Proceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland, 2004.

[25] S. S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9:60–62, March 1994.

[26] M. Yamamoto and Church K. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27:1–30, 2001.

[27] X. Zhu and R. Rosenfeld. Improving trigram language modeling with the world wide web. In *In proceedings of International Conference on Acoustics, Speech, and Signal Processing, 2001*, Salt Lake City, Utah, 2001.