The Design, Implementation and Use of the Ngram Statistics Package

Satanjeev Banerjee[1] and Ted Pedersen[2]

[1] Carnegie Mellon University, Pittsburgh, PA 15213 USA
[2] University of Minnesota, Duluth, MN 55812 USA
`http://www.d.umn.edu/~tpederse/nsp.html`

Abstract. The Ngram Statistics Package (NSP) is a flexible and easy–to–use software tool that supports the identification and analysis of Ngrams, sequences of N tokens in online text. We have designed and implemented NSP to be easy to customize to particular problems and yet remain general enough to serve a broad range of needs. This paper provides an introduction to NSP while raising some general issues in Ngram analysis, and summarizes several applications where NSP has been successfully employed. NSP is written in Perl and is freely available under the GNU Public License.

1 Introduction

A simple model of written text is as a series of symbols that carry some meaning when considered as a whole. We may wish to treat those symbols as phrases, words, or characters depending on our motivations. Ngrams are a simple representation that suits this view of written language. An Ngram is a sequence of N units, or *tokens*, of text, where those units are typically single characters or strings that are delimited by spaces. However, a token could also be a fixed length character sequence, strings with embedded spaces, etc. depending on the intended application.

The identification of Ngrams that are interesting in some way is a fundamental task in natural language processing. An Ngram might be considered interesting if it occurs more often than would be expected by chance, or has some tendency to predict the occurrence of other phenomena in text. There is a long history of research in this area. Character Ngrams were used by Shannon [10] to estimate the per–letter entropy of the English language. In the last decade there has been a large amount of work in developing corpus–based techniques to identify collocations in text (e.g., [2], [3], [6], [9]).

This paper describes the Ngram Statistics Package (NSP), a general purpose software tool that allows users to define Ngrams as they wish and then utilize standard methods from statistics and information theory to identify interesting or significant instances of Ngrams in large corpora of text.

Earlier versions of this package were known as the Bigram Statistics Package (BSP). This was first released in November 2000 (v0.1) and was limited to dealing with two word sequences (bigrams). In June 2001 BSP became NSP (v0.5) and was extended to handle Ngrams. As of this writing NSP is at v0.51 and remains an active project, with future releases planned.

What follows is a summary of NSP designed to acquaint a potential user with a few of the many features of the package. We also review general issues of Ngram processing, and briefly discuss research that has incorporated NSP.

2 Tokenization of Text

The typical first step of any natural language processing application is *tokenization*. The symbols that make up a text file are divided into *tokens* which represent the smallest indivisible units in that text. Tokens are often defined to be space delimited alphanumeric strings or individual ASCII characters, but could take many other forms depending on the application.

NSP is designed to allow the user to define tokens through the use of Perl regular expressions. In particular we define a token as a contiguous sequence of characters that match one of a set of regular expressions. These may be user-provided (via the `--token` option) and must be Perl regular expressions. If the user does not provide a token definition, the following two regular expressions provide a default, where the backslashes delimit a Perl regular expression:

/\w+/ → a contiguous sequence of alpha–numeric characters
/[\.,;:\?!]/ → a single punctuation mark

This default says that a token is either an alpha–numeric character string or an individual punctuation mark. Thus in *President George W. Bush visits with guests* the tokens are : *President*<>, *George*<>, *W*<>, .<>, *Bush*<>, *visits*<>, *with*<>, and *guests*<>.

In our notation tokens are terminated by the meta–character <>, and Ngrams composed of N tokens are represented by concatenating the <> terminated tokens one after another. Such a representation is required since tokens may include embedded spaces and using white space as a delimiter isn't possible. For example, *George W. Bush*<> represents a single token that starts with *G*, ends with *h* and includes two embedded spaces. This token could then be paired with another to create a bigram, as in *President*<>*George W. Bush*<>.

The NSP default definition of a token as a string of alphanumerics or a single punctuation mark may not be suitable in all cases. For example, we may not want to treat *George*<>, *W.*<>, and *Bush*<> as three separate tokens but as one, since they represent a single entity known as *George W. Bush*<>. On the other hand, in *Welcome first–time home buyers!* should the string *first–time* be two tokens or one? Further, do we wish to distinguish between *first–time* and *first time*? What about punctuation marks; should they be a part of the previous word, a token by themselves or should they be ignored? Similarly, there are various choices to be made in dealing with numbers, symbols, dates, abbreviations, etc. The lack of a universally appropriate definition for tokens motivates our desire to support a very flexible notion of tokenization in NSP.

Tokenization in NSP is done via the program `count.pl`. It converts the input file into one long string by replacing new–line characters with spaces. This string is then matched against the user–provided regular expressions (or the system defaults). Every regular expression specified is checked (in order) to see if any of them match the string starting with the first character of the input string. If none match, then the first character of the input string is considered a *non–token* and is removed and henceforth ignored. Otherwise the matching process stops at the first regular expression that yields a match. The longest sequence of characters (starting with the first character of the input string) that matches this regular expression is then identified as the next *token* and removed from the string. This process continues until the entire string has been matched and all the text identified as either tokens or not.

For example, assume that the following two regular expressions are being used to define tokens: /George W. Bush/, /\w+/. That is, the string *George W. Bush<>* will be considered a token, and so will every other unbroken sequence of alpha numeric characters. Thus, given the sentence *President George W. Bush visits with guests*, the output tokens are *President<>*, *George W. Bush<>*, *visits<>*, *with<>* and *guests<>*. Note that after *President<>* has been recognized as a token and removed, the resulting string *George W. Bush visits with guests* is matched by both regular expressions. However since regular expression are checked in the order in which they are provided, and the matching process stops at the first successful match, the resulting token is *George W. Bush<>* instead of just *George<>*. Thus the ordering of regular expressions in the token definition imposes a sort of priority, and it should be clear to a user that different orderings of a set of regular expressions can result in different tokenizations.

3 From Tokens to Ngrams

Once tokens are identified, `count.pl` assembles sequences of N tokens into Ngrams. Typically Ngrams are formed of *contiguous tokens*, that is tokens that occur one after another in the input corpus. Given *President George W. Bush visits with guests* and the token definition regular expression /\w+/, the possible bigrams (Ngrams with N = 2) are: *President<>George<>, George<>W<>*, *W<>Bush<>*, *Bush<> visits<>*, *visits<>with<>*, and *with<>guests<>*. Similarly, the possible trigrams (Ngrams with N =3) are: *President<>George<>W<>*, *George<>W<>Bush<>*, *W<>Bush<>visits<>*, *Bush<>visits<>with<>*, and *visits<>with<>guests<>*.

It may also be necessary to identify Ngrams from *non–contiguous* tokens, that is tokens separated by some number of intermediate tokens. For example, given the text *President George W. Bush*, it may be advantageous to identify the bigrams *President<>Bush<>* and *George<>Bush<>* and the trigram *President<>George<>Bush<>* in addition to the sequential bigrams described above. This is useful when one wants to report having observed the bigram

George<>Bush<> even when those two tokens are separated by the intervening token *W<>*.

To allow Ngrams to be formed from non–contiguous tokens, `count.pl` provides a `--window` option. This defines a window of k contiguous tokens, where the value of k is greater than or equal to the value of N. An Ngram can be formed from any N tokens as long as all the tokens belong to a single window of size k. Further the N tokens in the Ngram must occur in exactly the same order as they do in the original window of text.

Thus given a window size of k and an Ngram size of N, we have ${}^{k}C_{N}$ (k choose N) Ngrams per window. For example, consider again the text *President George W. Bush visits with guests.* The following are all the possible bigrams for a window size of 3: *President<>George<>*, *President<>W<>*, *George<>W<>*, *George<>Bush<>*, *W<>Bush<>*, *W<>visits<>*, *Bush<> visits<>*, *Bush<>with<>*, *visits<>with<>*, *visits<>guests<>,* and *with <>guests<>*.

4 Counting Ngram Frequencies

Having tokenized a given corpus of text and from that constructed Ngrams, the program `count.pl` counts the number of times each Ngram occurs in the corpus. It outputs the frequency of each unique Ngram, as well as the frequencies of the various combinations of tokens that make up the Ngram.

4.1 Counting Bigrams

Suppose NSP is counting two token sequences of alphanumeric strings. The output of `count.pl` consists of a count of the total number of bigrams in the corpus, followed by a list of all the unique bigrams and their associated frequency counts. Here we show a small example, which just shows a single bigram and its counts:

```
1,319,237
George<>Bush<>27 134 463
```

The value 1,319,237 is the number of bigrams found in the corpus, and can be thought of as the sample size. Note that this is not a count of the unique bigrams but rather the total number of bigrams without regard to repetition. The next line represents the bigram *George<>Bush<>* and shows that the bigram itself has occurred 27 times in the corpus. Further, the token *George<>* has occurred as the "left hand" token in 134 bigrams in the corpus, which includes the 27 instances of the bigram *George<>Bush<>* itself. Similarly the token *Bush<>* has occurred as the "right hand" token in 463 bigrams, 27 of which are *George<>Bush<>*.

The format of the `count.pl` output is a compact representation of a typical two–by–two contingency table. For example in Table 1, the four internal cells categorize the 1,319,237 bigrams in the corpus into four disjoint sets: 27 instances of the bigram *George<>Bush<>*, 436 bigrams that have *Bush<>* as the second

Table 1. Contingency table for *George<>Bush<>*

	Bush	**!Bush**	
George	27	107	134
!George	436	1,318,667	1,319,103
	463	1,318,774	1,319,237

token and do not have *George<>* as the first token, 107 bigrams that have *George<>* as the first token and do not have *Bush<>* as the second token, and the remaining 1,318,667 bigrams that have neither *George<>* as the first token nor *Bush<>* as the second token.

Observe that the rest of the contingency table can be reconstructed from the internal cell count 27, the marginal frequencies 134 and 463, and the sample size of 1,319,237. Note that the sample size will be the same regardless of which Ngram from the corpus is under consideration. Thus, this value need only be represented once in the `count.pl` output.

4.2 Counting Ngrams

Although counting bigrams is the default behavior of program `count.pl`, the user can set the value of N through the option `--ngram`. For trigrams and longer Ngrams, frequency values of various combinations of tokens are also computed. For example consider the following output after creating and counting trigrams:

```
1,316,737
President<>George<>Bush<>2 338 134 463 3 2 27
```

The sample size is 1,316,737 and indicates the total number of trigrams in the corpus. The next line gives counts for the trigram *President<>George<>Bush<>*, which occurs in the corpus exactly twice. Further, the token *President<>* occurs as the first token in 338 trigrams, the token *George<>* occurs as the second token in 134 trigrams and the token *Bush<>* occurs as the third token in 463 trigrams in the corpus. Finally the tokens *President<>* and *George<>* occur simultaneously as the first and second tokens in 3 trigrams, the tokens *President<>* and *Bush<>* occur as the first and third tokens in 2 trigrams and the tokens *George<>* and *Bush<>* occur as the second and third tokens in 27 trigrams.

This data is represented in Table 2. Here, the 1,316,737 trigrams are broken up into eight categories depending upon whether they contain or do not contain the three particular tokens in the three specific positions. Observe that `count.pl` only produces the minimum number of frequencies required to reconstruct the table. This is particularly important as the value of N grows larger.

When given an Ngram, `count.pl` represents its leftmost token as w_0, the next token as w_1, and so on until w_{n-1}. Further let $f(a, b, ..., c)$ be the number of Ngrams that have token w_a in position a, token w_b in position b, ... and token

Table 2. Contingency tables for *President<>George<>Bush<>*

		Bush	**!Bush**	
President	**George**	2	1	3
President	**!George**	0	335	335
!President	**George**	25	106	131
!President	**!George**	436	1,315,832	1,316,268
		463	1,316,274	1,316,737

w_c in position c, where $0 <= a < b < ... < c < n$. Then, given an Ngram, the first frequency value reported is $f(0, 1, ..., n-1)$; this is the frequency of the Ngram itself. This is followed by n frequency values, $f(0)$, $f(1)$, ..., $f(n-1)$; these are the frequencies of the individual tokens in their specific positions in the given Ngram. This is followed by (n choose 2) values, $f(0, 1)$, $f(0, 2)$, ..., $f(0, n-1)$, $f(1, 2)$, ..., $f(1, n-1)$, ... $f(n-2, n-1)$. This is followed by (n choose 3) values, $f(0, 1, 2)$, $f(0, 1, 3)$, ..., $f(0, 1, n-1)$, $f(0, 2, 3)$, ..., $f(0, 2, n-1)$, ..., $f(0, n-2, n-1)$, $f(1, 2, 3)$, ..., $f(n-3, n-2, n-1)$. And so on, until (n choose n-1), that is n, frequency values $f(0, 1, ..., n-2)$, $f(0, 1, ..., n-3, n-1)$, $f(0, 1, ..., n-4, n-2, n-1)$, ..., $f(1, 2, ..., n-1)$.

This gives us a total of 2^{n-1} possible frequency values for Ngrams of size n. We call each such frequency value a *frequency combination*, since it expresses the number of Ngrams that have a given combination of one or more tokens in one or more specific positions. By default all such combinations are output, exactly in the order shown above. However the total number of frequency values grows exponentially with the value of n, that is the Ngram size under consideration. Since computing, storing and later displaying such a large number of frequency values can be both very resource intensive as well as unnecessary, the package gives the user the capability to specify which frequency combinations he wishes to have computed and displayed. Specifically the user can use the option `--set_freq_combo` to provide program `count.pl` with a file containing the inputs to the hypothetical $f()$ function above to specify which frequency combinations she desires to have counted. For example, to compute only the frequencies of the trigrams and those of the three individual tokens in the trigrams (and not of the pairs of tokens), the user can tell the package just to count the following $f()$ functions: $f(0, 1, 2)$, $f(0)$, $f(1)$, and $f(2)$. This will result in the following counts:

```
President<>George<>Bush<>2 338 134 463
```

The only difference from the previous example is the fact that the frequency values $f(0, 1)$, $f(0, 2)$ and $f(1, 2)$ are not output. However, there are considerable internal differences as any frequency combinations that are not requested are not counted, thus realizing a considerable savings in computation time and memory utilization for large corpora and larger values of N.

4.3 Ngram Filters

Often it is necessary to filter the entire set of Ngrams and observe only a small subset of all the possible Ngrams in a given input text. For example sometimes Ngrams made up entirely of function words are not interesting and one may wish to *stop* or ignore them. This package provides two different mechanisms through which to create smaller subsets of Ngrams.

In the first mechanism, the user may use the option `--stop` to pass to program `count.pl` a file containing a list of *stop words*, and Ngrams that are made up entirely of these words will not be created. For example if the user provides the words *the* and *of* as stop words, then given the sentence *He is one of the worst kinds*, the bigram *of*<>*the*<> will not be created. However bigrams *one*<>*of*<> and *the*<>*worst*<> will continue to be created since they are not made up *entirely* of stop–words and have at least one word not in the stop list. This stopping technique is particularly useful during the creation of non–contiguous Ngrams when Ngrams composed entirely of function words become more likely.

In the second mechanism, the user may specify a *frequency cut–off*. Every Ngram that occurs less than some specified number of times can be ignored (option `--remove`), in which case they are excluded from the sample size and do not affect any frequency counts, or they can be counted but simply not displayed (option `--frequency`), in which case they are included in the sample size and affect the various frequencies. The first case assumes that Ngrams that occur less than the cut–off number of times are not significant enough to include in overall counts, while in the second case these low frequency Ngrams affect the overall counts but are not displayed in the `count.pl` output. These are radically different approaches to counting, and both are appropriate under certain circumstances. The user must choose between these cut–off mechanisms with some care so as to avoid unexpected results.

5 Measures of Association for Ngrams

Once a user has identified and counted Ngrams and their components via the `count.pl` program, NSP allows a user to go on and apply various measures of association to that data with the program `statistic.pl`. Such measures judge whether the tokens that make up the Ngram occur together more often than would be expected by chance. If so, then the Ngram may represent a collocation or some other interesting phenomena.

A measure that returns a score that can be assigned statistical significance is referred to more precisely as a *test of association*. Examples supported in NSP include the log–likelihood ratio, Fisher's exact test, and Pearson's chi–squared test. Measures that do not allow for significance to be assigned to their value include the Dice Coefficient and pointwise Mutual Information. When discussing both kinds of techniques we refer to them generically as *measures of association* and use the more specific term *test of association* when appropriate.

5.1 Background

To support measures of association on Ngrams, NSP implicitly defines N random binary variables $W_i, 0 \leq i < N$, where W_i represents the i^{th} token in the Ngram. Each of these variables indicate whether or not a particular token occurs at the given position. For example, the variable W_0 could represent whether or not *George*<> occurs in the first position of the Ngram.

In Table 1 the first row of the contingency table represents all Ngrams such that $W_0 =$ *George*<> (it occurs), while the second row represents all Ngrams such that $W_0 \neq$ *George*<> (it does not occur). Similarly, the first column represents all Ngrams such that $W_1 =$ *Bush*<> while the second column represents all Ngrams such that $W_1 \neq$ *Bush*<>.

Tests of association between two random variables typically set up a null hypothesis that holds if the two random variables are independent of each other. A pair of words that fail this test might then be considered to be related or dependent in some way, since they have failed to exhibit statistical independence. Formally speaking, for two words that make up a bigram to be considered independent, we would expect the probability of the two words occurring together to be equal to the product of the probabilities of the two words occurring separately.

For example, if the bigram under consideration is *George*<>*Bush*<>, the probability of its occurrence could be represented by $P(W_0, W_1)$. For these two words to be considered independent this joint probability would have to be equal (or nearly so) to the product of the probabilities of the individual words *George*<> and *Bush*<>, represented by $P(W_0)$ and $P(W_1)$. Thus, these tests of association are based on the formal definition of statistical independence, i.e., $P(W_0, W_1) = P(W_0)P(W_1)$. To reject the hypothesis of independence, one must find that the value of $P(W_0, W_1)$ that is based on *observed* frequency counts diverges from the *expected* values that are based on the hypothesis of independence.

For Ngrams where $N \geq 3$, there are numerous ways to formulate a null hypothesis. With more than 2 random variables, the null hypothesis can capture a wider range of possible models than simple independence. Here we are moving from tests of association into the more general realm of statistical model evaluation. For example, when $N = 3$, one can formulate the null hypothesis that the observed probability of a trigram reflects that the three words are completely independent of one another, or that two of the words are dependent on each other but independent of the third. In these cases the null hypotheses could be formulated as: $P(W_0)P(W_1)P(W_2)$ or $P(W_0, W_1)P(W_2)$ or $P(W_0)P(W_1, W_2)$ or $P(W_1)P(W_0, W_2)$.

Each of these null hypotheses represents a different hypotheses, and the expected values for each could be compared to the observed value of $P(W_0, W_1, W_2)$ to determine how closely the observed values correspond with the expected values. Recall from the previous section that program `count.pl` allows the user to compute all such frequencies through the option `--set_freq_combo`. Thus the package allows the user to create a wide range of null hypotheses particularly as N grows larger.

5.2 Implementation

Given the observed frequencies from `count.pl`, a user can apply a measure of association to determine if the words in an Ngram are somehow related. Although this package implements several measures of association, the primary design goal was to facilitate the quick and easy implementation by the user of their own favorite measures. This is achieved via the program `statistic.pl`, which is the tool designed to process the list of Ngram counts produced by `count.pl` and apply measures of association to that data.

The program `statistic.pl` remains unchanged regardless of the measure of association it is performing. This is achieved by requiring that a measure be implemented as a Perl *module* that exists as a file separate from the rest of the program and is plugged into `statistic.pl` at run–time. Such a module must follow a set of rules that specify the interface between it and `statistic.pl`. For each Ngram in the corpus, `statistic.pl` passes to the module the size of the corpus and the various frequency values associated with the Ngram. The module is then expected to return a floating point number that expresses the degree to which the tokens that make up the given Ngram are associated with each other. Mechanisms exist for the module to throw exceptions so that `statistic.pl` can exit gracefully.

A module that implements a measure of association must *export* two functions to `statistic.pl`: `initializeStatistic()` and `calculateStatistic()`. The former is the first function called by `statistic.pl` and is used to pass to the module such information as the Ngram size, the total number of Ngrams in the corpus and a data structure containing the list of frequency combinations associated with each Ngram in this dataset. For every Ngram, `statistic.pl` calls function `calculateStatistic()` and passes to it all the frequency values associated with that Ngram. This function is expected to return a floating point value proportional to the degree of association for the Ngram in question.

Besides these two functions, the user may also implement and export three more functions: `errorCode()`, `errorString()` and `getStatisticName()`. The first two can be used to throw exceptions while the last can be used to return a string containing the name of the measure; if returned, this string is used in the formatted output of the program.

The advantage of this design is that it allows the user to concentrate entirely on the mechanism of the statistical measure without concern to the rest of the infrastructure. For example the processing of the list of Ngrams in the corpus of text, the counting and storage of their frequency values, etc is already taken care of. The author of a new measure need only focus on the measure's inputs, outputs and internal computation.

6 Comparing Ranked Lists of Ngrams

NSP is designed not only to create and analyze Ngrams in a corpus of text, but also to allow the user to study the effect of new measures of association. Section

5 describes how the user can implement a new measure and integrate it into the package. NSP also provides a program `rank.pl` that allows a user to compare two ranked lists of Ngrams and determine how much they differ with respect to each other. Thus if a user introduces a new measure it is possible to determine how much it resembles or differs from some existing ones.

`rank.pl` implements the Spearman's Rank Correlation Coefficient to compare two measures of association. This coefficient measures the correlation between two different rankings of a list of items. Specifically, given a set of Ngrams and their frequencies as observed in a corpus of text, we rank them according to each of the two measures of association, and then compute the correlation between these two different rankings using equation 1.

$$r = 1 - \frac{6 \sum_{i=1}^{i=n} D_i^2}{n(n^2 - 1)} \tag{1}$$

In this equation, n is the total number of unique Ngrams in the corpus, D_i is the difference between the rankings of Ngram i in the two lists and r is the value of the correlation. The value of r ranges from -1 to +1. A value of 0 implies no correlation between the two lists, while values that are further away from 0 imply greater correlation where the sign of the value indicates positive or negative correlation.

7 Applications of the Ngram Statistics Package

The range of applications in which NSP has been utilized reflects the generality with which Ngrams can be employed.

An original motivation for developing NSP was to support the second author's work in word sense disambiguation. He has developed a supervised approach to word sense disambiguation that learns decision trees of bigrams from sense–tagged corpora (e.g., [7], [8]). In this approach a word is disambiguated based on the word bigrams that occur nearby. This approach has proven to be relatively successful and is quite easy to implement, at least in part due to NSP.

The language independence of Ngrams and NSP is demonstrated by several applications with Dutch that identify collocations that involve non–content words. Bouman and Villada [1] use NSP to identify collocational prepositional phrases, while van der Wouden [11] uses it to determine a variety of non–content collocations in Dutch text.

The range of possible applications for Ngrams and NSP is illustrated by the following projects. Zaiu-Inkpen and Hirst [13] extend a database of near–synonyms with information about their collocational behavior. Lopez et. al. [5] use information about word bigrams to take the place of parses when no parse was available in performing word alignment of parallel text. Gill and Oberlander [4] compare the writing styles of introverts and extroverts by identifying word bigrams used by one group but not the other in written text.

8 Future Work

There are a number of possible enhancements to NSP that will be carried out in the next few years.

Ngrams are counted by storing them in a hash table. This poses no problems for relatively large corpora of a few million tokens, but to process 100 million token corpora a more efficient mechanism must be developed. One possibility would be the use of suffix trees as described by Yanamoto and Church [12].

NSP version 0.51 provides a small number of measures of association that are only implemented for bigrams. We are beginning to implement measures for trigrams and will include those in future releases. In addition, NSP is now geared towards ASCII text. We attempted to incorporate Unicode support as provided in Perl 5.6 but found that it was not yet stable. We are hopeful that this situation will improve with Perl 5.8 and allow NSP to support Unicode.

At present NSP is a stand–alone package that runs from the command line. We plan to implement it as a set of library modules that will allow it to be included in programs and also to take advantage of some of the object oriented features that Perl supports. We also plan to provide a graphical interface with Perl/Tk in addition to the command line support. In conjunction with this we would increase the graphs and charts available to the user for exploring their data.

9 Acknowledgments

The Ngram Statistics Package was implemented by the first author while he was at the University of Minnesota, Duluth. During 2000-2001 he was supported by a Grant-in-Aid of Research, Artistry and Scholarship from the Office of the Vice President for Research and the Dean of the Graduate school of the University of Minnesota. In 2001-2002 both authors were supported by a National Science Foundation Faculty Early CAREER Development award (#0092784). The first author is currently supported by the National Science Foundation under Grant No. REC–9979894.

Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government.

References

1. G. Bouman and B. Villada. Corpus–based acquisition of collocational prepositional phrases. *Computational Linguistics in the Netherlands (CLIN)*, 2002.
2. K. Church and P. Hanks. Word association norms, mutual information and lexicography. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, 1990.

3. T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.
4. A. Gill and J. Oberlander. Taking care of the linguistic features of extraversion. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society*, pages 363–368, Washington, D.C., 2002.
5. A. Lopez, M. Nossal, R. Hwa, and P. Resnik. Word–level alignment for multilingual resource acquisition. In *Proceedings of the 2002 LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*, 2002.
6. T. Pedersen. Fishing for exactness. In *Proceedings of the South Central SAS User's Group (SCSUG-96) Conference*, pages 188–200, Austin, TX, October 1996.
7. T. Pedersen. A decision tree of bigrams is an accurate predictor of word sense. In *Proceedings of the Second Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 79–86, Pittsburgh, July 2001.
8. T. Pedersen. Machine learning with lexical features: The Duluth approach to Senseval-2. In *Proceedings of the Senseval-2 Workshop*, pages 139–142, Toulouse, July 2001.
9. T. Pedersen, M. Kayaalp, and R. Bruce. Significant lexical relationships. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 455–460, Portland, OR, August 1996.
10. C. Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, 30(50–64), 1951.
11. T. van der Wouden. Collocational behavior in non content words. In *ACL/EACL Workshop on Collocations*, Toulouse, France, 2001.
12. M. Yanamoto and K. Church. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27(1):1–30, 2001.
13. D. Zaiu Inkpen and G. Hirst. Acquiring collocations for lexical choice between near synonyms. In *SIGLEX Workshop on Unsupervised Lexical Acquisition, 40th meeting of the Association for Computational Linguistics*, Philadelphia, 2002.