2005

Intelligent Autonomous Data Categorization

Edward Graham Finegan
Virginia Commonwealth University

Intelligent Autonomous Data Categorization

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

By

Edward Graham Finegan
B.S. Rowan University 2003

Director: David Primeaux, Ph.D.
Interim Chairman, Department of Computer Science

Virginia Commonwealth University
Richmond, Virginia
August 2005

Acknowledgment

I would like to show my appreciation for my wonderful girlfriend Shannon for her help and support.

Table of Contents

List of Tables

List of Figures

Abstract

INTELLIGENT AUTONOMOUS DATA CATEGORIZATION

By Edward Graham Finegan, M.S.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2005.

Major Director: David Primeaux, Ph.D. Interim Chairman

Department of Computer Science

The goal of this research was to determine if the results of a simple comparison algorithm (SCA) could be improved by adding a hyperspace analogue to language model of memory (HAL) layer to form NCA. The HAL layer provides contextual data that otherwise would be unavailable for consideration. It was found that NCA did improve the results when compared to SCA alone. However, NCA added complexity problems that limit its practicality. The complexity of this algorithm is On^3 where n is equal to the number of unique symbols in the

data. While there is a relativity reasonable soft upper bound for the number of unique symbols used in a language, the complexity still limits the uses of the NCA combined algorithm. The conclusion from this research is that NCA can improve results. This research also suggested that the quality of results might increase as more data is processed by NCA.

Review of the Literature

Introduction

This research was conducted to determine if the results of a simple comparison algorithm (SCA) could be improved by adding a hyperspace analogue to language model of memory (HAL) layer. Although this research will focus on grouping similar web pages, it could be applied to any set of text documents. An intelligent algorithm is used because such an algorithm has the ability to adapt to the environment it is used in. This intelligent algorithm will learn from the data that it has already processed and continually improve the results of its output.

The hyperspace analogue to language model of memory (HAL) was chosen for its ability to learn contextual meanings of words. (Burgess 1998) The intent in this research is to combine a learning HAL with SCA to create a new comparison algorithm (NCA) which will effectively form groupings of similar web pages. The HAL layer is intended to give NCA an insight into the context of words and symbols that appear in web pages. HAL can process the contextual meaning of words,

but can also process misspelled words and symbols such as ":)". Since this algorithm processes both words and symbols, the term symbol will be used with the intended meaning that a word can be a symbol. The SCA without HAL would evaluate symbols literally and meanings would not be considered. By combining these two algorithms into NCA an intelligent algorithm should be created that can group web pages based on the context of their content.

The High Dimensional Memory Model

HAL falls into the category of a high-dimensional memory model (HDM). (Osgood, Suci, Tannenbaum 1957) HDM has a goal of extracting and representing the meaning of words or symbols from a streamed input. These models then learn the context of words and symbols similarly to the manner in which humans do. When recalling a sentence most people do not remember what the sentence is word for word, but instead abstract a meaning, which is called a mental model. A HDM works in much the same way. (Burgess 1998)

HDM are not a new concept. In the 1950's Charles Osgood pioneered much of the early work. (Osgood, Suci, Tannenbaum 1957) However, recently, the number of papers investigating HDM has increased. Current papers produced by Burgess, Lund, Landauer, Dumais, Laham and Foltz have advanced the use of

HDM in many directions. Many HDM require some type of user interaction or interpretation. This made using the HDM time consuming and limits its benefits.

HAL is a HDM that requires no user interaction or interpretation to learn the contextual meaning of words or symbols in a language stream. As a result, HAL can address a variety of difficult and problematic situations that caused previous HDM to struggle.

HAL takes a set of text data as input. It then tracks lexical occurrences of symbols within a sliding window. (Burgess 1998) This sliding window can be of any size. In previous research (Burgess 1998) and in this research, the window size has been set to ten symbols before and after the target symbol. HAL then stores the lexical occurrences from this window. Using this stored information HAL can then generate a vector point for each unique symbol in vector space. (Burgess 1998) Since this point represents the context of the symbol, the space it is contained in is referred to as context space. (Burgess 1998) The distance between two points in the context space can be calculated. This distance represents the extent of contextual similarity of one symbol to another.

HAL is not the only style of HDM in use. Latent Semantic Analysis (LSA) is also a very popular model. (Landauer, Foltz, Laham 1998) This model predates HAL. LSA is a method used for extracting and representing the contextual meaning of symbols by statistical computations applied to a large corpus of text. Although the method and application of HAL and LSA differ, the literature indicates that they produce similar results.

The HAL model focuses on finding contextually similar symbols. LSA is a broader model that has more flexibility and can be used in a variety of human memory modeling tasks. Thomas K. Landauer, a major researcher behind LSA claims:

> The principal difference between the HAL and LSA approaches to date, in addition to possibly significant technical differences such as similarity metrics, is our focus on the importance of dimensionality matching as a fundamental inductive mechanism rather than merely a computational convenience. (Landauer, Dumais 1997)

Put simply, the LSA model attends to how humans learn language. A major accomplishment of LSA is its ability to have vocabulary knowledge equal to that of a child. (Landauer, Foltz and Laham 1998) This is achieved by processing the same amount of language data to which a child would have been exposed. LSA uses statistical functions to extract meaning from the data. LSA has also been used to model human brain disorders.

HAL was derived from the LSA model without the constraint of closely modeling human language processing. Instead HAL was designed to produce similar results but without the extra overhead involved in modeling human behaviors.

This research attempts to further the use of the HAL model. HAL was chosen since its goals of producing contextual data of symbols are inline with the goals of this research. LSA's goals were less specific and did not fit as well with this research. HAL can also produce interesting results with a simpler implementation.

Currently, using human agents is potentially the most accurate way to intelligently group collections of data. However, humans have the disadvantages of being slow and prone to fatigue and error. Online search engines such as *Google* have developed algorithms to find similar pages. *Google* finds similarities much faster than humans could. Although these algorithms are proprietary, leading search engine researchers believe they know the general principal behind them.

In general, the search engines create a graph in which web pages (nodes) are connected by the links between them. Related web pages are often highly connected to one another. These densely connected areas are called communities. A web

page that is in another web page's community, with similar keywords, is deemed a similar web page. This method does return acceptable results. However, these results are incomplete. If a web page is weakly linked to its community it may not be deemed relevant. (Churchill 2005) Also, this method only works in situations where the pages are linked as they are on the web. This method would not work on a home user's PC to group plain text files similar in context but containing no links.

The HAL Algorithm

Implementation of HAL is straightforward. All data files are taken as input. A matrix is created of size *N* x *N*, where *N* is equal to the number of unique symbols found throughout the body of data. An example matrix can be found in Table 1. A sliding window is then used to locate lexical similarities of symbols within the text. For example, consider the following sentence, "The motorcycle raced down the street home". The window would start at the beginning of the sentence. The size of the window can be set to any value less then *N*. It has been found that any window value greater then ten adds little to the meaning of the word being analyzed. (Burgess 1998) In this example sentence, however, a window size of five will be used in order to simply demonstrate how this model works.

The word “The” is the first symbol to be analyzed. Since the window size is five, the next five symbols are given a score in the *N* x *N* matrix created earlier (Table 1). As the algorithm proceeds, the values stored in the columns of the matrix will represent values associated with symbols in that half of the window prior to the current symbol. Similarly, values in the matrix’s rows will represent the half of the window that is after the current symbol. By studying the symbol “home”, in Table 1, it is evident that its row is empty. This is because “home” has no symbols following it. However its column does contain data representing the previous appearing symbols in the window.

The values in the matrix are initialized to 0 and the results are accumulated. The maximum score added to any symbol is the size of the window. As seen in Table 1, when the initial “the” is the symbol of interest, “motorcycle” is the closest symbol to “the”, so a score of 5 is added to the initial 0 value associated with “motorcycle”. The next word, “raced” results in a score of 4, “down” 3, “the” 2, and “street” 1. The window then slides to the right, one symbol, and now “motorcycle” is the symbol that scores are being assigned against. From “motorcycle”, “raced” is the closest symbol to the right and results in a score of 5. Next is “down” which results in a score of 4, “the” 3, “street” 2

and finally "home" results in a score of 1. This process is continued sequentially for all of symbols in the text. Once this process is finished the matrix of scores shown in Table 1 would have been generated.

Table 1: "the motorcycle raced down the street home" scoring matrix

	the	*motorcycle*	*raced*	*down*	*street*	*home*
the	2	5	4	3	6	4
motorcycle	3	0	5	4	2	1
raced	4	0	0	5	3	2
down	5	0	0	0	4	3
street	0	0	0	0	0	5
home	0	0	0	0	0	0

The effect of accumulating scores can be seen by inspection of the row corresponding to the symbols "the". By recording how the symbols are used repeatedly, HAL learns its contextual meaning. As stated before, a window of size five represents five symbols before and after the current symbol. However, in the calculations only that half of the window after the current symbol needs to be taken into account. For example when the window is centered on "motorcycle" the symbol "the" precedes it. The symbol "the" is contained in the window, but since "the" precedes "motorcycle" there is no need to add a score for this relationship to the matrix. This relationship will automatically be represented in the columns of the matrix once all of the rows are filled.

Once all score values for the matrix have been completed, it becomes easy to compare two symbols. To determine a vector

point in the context space, the complete row and column for one symbol is inspected. Since we have six unique symbols in the example above, we have a twelve-dimensional context space. The space is twelve dimensional and not six because there are six values in the columns representing a symbol of interest's contextual relationship to the symbols preceding it in its window, while the six values in the rows represent a symbol of interest's contextual relationship to the symbols following it in its window. The context space location of the symbol "raced" from above would be {4,0,0,5,3,2,4,5,0,0,0,0}. The first six values for this vector are found in the row for "raced" in Table 1. The last six values are then found in the column for "raced" Table 1.

Once it is understood how this model works, it is easy to see why it is so flexible. Previous HDM were unable to handle common spelling errors, or typing errors. In the HAL model, common errors are quickly grouped together by their contextual meanings. If we take the previous example sentence and add an additional similar sentence to our set of data, it becomes apparent why this model is so flexible.

Original: "The motorcycle raced down the street home."

New: "The motorcycle raced down the road home."

Both of these sentences are identical except that one uses “street”, and the other uses “road”. When the two sentences are processed through the HAL, “street” and “road” will have the same vector point in context space. Since the distance between the two will be zero, it is safe to assume that in this set of data these two symbols have the same meaning and are interchangeable. In a larger set of data it would likely be very rare that two symbols will have identical vector points. However, once the distance between such similar symbols is calculated, the distance will be so small in comparison to other distance values that these two symbols will be interpreted as being very near or identical in meaning.

Summary

HAL is a flexible and adaptive algorithm. It can learn the context of words and symbols that normally would not be expected, such as unique symbols resulting from spelling and typing errors. The HAL algorithm is straightforward to implement. HAL creates a matrix of unique symbols and fills in score values based on locations relative to a sliding window. Once the matrix is computed, it can be used to find the vector points of each symbol in the high dimensional context space.

The Problem: An Explosion of Data

Growth of Data and its Organization

Everyday more data is being created. The data ranges from that generated by large corporations with data warehouses, to that generated by end users with a few gigabytes of data on their home PC's hard drive. The Internet is also expanding at a high rate and finding ways to organize this data is becoming challenging. It would seem that keeping the data grouped together based on similarities in its content might prove useful. This research will investigate the possible use of a HAL in conjunction with SCA to discover if HAL's ability to learn the contextual meaning of symbols can benefit the SCA and produce more useful results.

An objective of this research is to find a method that is both fast, because it is processed by computer, as well as adaptive and intelligent enough to produce results that approach those of a tireless and error-free human. An algorithm that can produce both fast and intelligent results will be beneficial to any user wanting to group or categorize data according to content.

Assumptions

Several assumptions underlie this research. The sample data used is assumed to be representative of data that may be discovered in a real world situation. A second assumption is that any findings on this small set of data will hold true for larger sets. Thirdly, while not true in all domains, it is assumed that any trends found in these results will also be present in the larger sample sizes. Finally, even given differences in human opinion regarding which documents or web pages should be grouped together, this data can be intelligently grouped and the similarity of their content agreed upon by reasonable people.

Research Hypothesis

NCA, using the use of the HAL algorithm in conjunction with SCA will produce more intelligent and flexible results when compared to the use of either HAL or SCA alone.

Significance of the Research

This research develops and implements NCA, an algorithm that is intelligent and adaptive. The results of NCA should create contextually similar groups of data. After the

intelligent algorithm processes the data, it groups the data based on content.

Scope

The scope of this research is limited to the design and verification of the NCA algorithm. The sample data in this research, which in theory can be any type of data file, will only consist of web pages from the Internet. Since the sample data set must be selective, it was arbitrary decided that only three categories of data would be used: baseball, football, and the Iraqi war.

Summary

An algorithm able to intelligently group data would be an asset to many users. Hypothetically, such an algorithm can be created by conjoining HAL with SCA. The resulting NCA algorithm would learn contextual meanings from the data it has already processed. This research is limited to a small set of sample data. The scope of this research is to develop this NCA algorithm and consider the likelihood of it working in a less constrained scenario.

Research Procedures

Research Design

This research was performed on web pages collected from the Internet. The data was hand picked in an attempt simulate differences and similarities between the web pages. Baseball, football and the Iraqi war were the three categories that data was chosen from. The NCA algorithm presented in this research was used to process this data. The output was analyzed and compared against the expected results. Anomalies in the results were then further studied in order to determine their cause.

The NCA Algorithm

To conduct this research two tools were created. One tool stripped HTML from web pages. No known public algorithm compares two web pages while utilizing the HAL algorithm. Therefore an implementation of NCA was created to carry out this task.

The first (and simpler) tool was an HTML parser (see Appendix B). The sample data that was acquired contained

HTML tags. Because it is believed that removing the HTML tags the classification results could be based on content alone as opposed to effects of formatting and content. The parser processes the sample data and removes most of the HTML. It can be argued that the formatting can also contain a deeper level of context meaning. While this may be true, consideration of formatting effect on meaning is beyond the scope of this research.

The primary tool used in this research was the design and implementation of NCA that incorporated the HAL algorithm and SCA (see Appendix C). Java was the chosen language for this tool. By using Java the constraints of platform dependency were removed. Java's many libraries allowed more attention to the research as opposed to some implementation details. Once processing of selected data is complete, the user selects a web page to be compared against all other web pages that have been processed. The tool will output a list of like web pages with a score given to each one. A higher score means the two web pages are more contextually similar.

In detail, the NCA tool first traverses all the data and counts the number of unique symbols. A matrix of size N x N (where N is equal to the number of unique symbols) is created. The tool then applies the sliding window to the text and fills in scores for all the symbols in the matrix.

Once the matrix scores are filled in, a second matrix is created of equal size to the first.

The second matrix is the distance matrix. The distance matrix is unique to NCA. It is used to store the distance values for every symbol combination. The distance, Δd is found by using the Euclidean distance equation.

$$\Delta d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_n - b_n)^2}$$

Equation 1: Euclidean distance equation

The vector points ($(a_1,\ldots,a_n)$ from the 1st symbol, $(b_1,\ldots,b_n)$ from the 2nd symbol, where n is equal to the number of unique symbols) are obtained from the first matrix and then plugged into the equation. The distances are calculated so that once they are found the comparison of two selected documents can run swiftly.

Filling this second matrix can be very time consuming. The complexity, which will be discussed in detail later, is $O(n^3)$. The distance matrix stores the distances of every symbol from every other symbol. For example, symbol A to symbol B then symbol B to symbol A. Consequently, this matrix contains a redundant set of data. By only calculating half of the matrix the processing time can be cut in half

with no loss of information. Even with the count of calculations cut in half, however, the processing time can be staggering.

With the distance data complete the program must decide which symbols are similar. This is a difficult problem because the size of context space is dependent on the data set. During this research a novel equation was created to compute a special case of the Euclidean distance and to give intuitive meaning to the HAL algorithm's generated distances. Equation 2 can find a "basic distance" value, Δb.

$$\Delta b = \sqrt{4\sum_{i=1}^{w} i^2}$$

Equation 2: Basic Distance where w = HAL window size

The basic distance equation is derived from the Euclidean distance equation (Equation 1). This equation finds the distance of two symbols that only appear once in the data set and never appear in the same sliding window. An example Δb, with window size w = 3 and n = 10 unique symbols, follows. Since n = 10 a 20 dimensional context space is required. Since the symbols only appear once, possible coordinates for a one symbol will be { 3, 3, 2, 2, 1, 1, 0_7, …… , 0_{20} }. Recall that the two symbols never appear in the

same sliding window. Let vector *A* corresponding to the first symbols be compared to the vector *B* corresponding to the second symbol. Then if $A_i > 0$ then $B_i = 0$. Therefore, when these two vectors are used in the Euclidean distance equation, no set of parentheses will hold two values greater than 0. If the contrary were true contextual information would be present, but this is not the case. As a result when the Euclidean distance equation is used with symbols that never appear in the same sliding window, each difference of coordinates squared will contain at least one zero. As a result a simple summation is used. The 4 in the Δb equation derives from the fact that it represents a distance for two symbols, and each symbol has two values for each position of the window. The other aspects of the computation follow a straight forward extension of the standard Euclidean distance computation.

If a distance is equal to this Δb it is inferred that the two symbols appear one time each and there is no data suggesting any contextual relationship. If the distance value is greater then Δb it implies that one or both symbols were found in the data set more than once; furthermore, contextual information indicates that these two symbols are more contextually different than they are contextually similar. When looking for similar contextual meanings it can

be concluded that the more important distance values are less than Δb. When the distance is less than Δb the data suggests more contextual similarities than differences.

Distances below Δb show evidence of contextual similarities. However a distance only slightly below the basic distance value may not actually demonstrate a similarity. In this research, a ratio of Δb was used to filter out those symbols with less contextual similarity. This ratio value is another aspect that is unique to NCA. The ratio is an experimentally determined number used to fine tune the results produced from the NCA algorithm. This research found that a ratio between .4 and .8 works well on the data used. As the ratio increases, the filter's tolerance for less contextual similarities increases. This ratio can be used to compute a similarity threshold is , T, where $T = (ratio\ value) * (\Delta b)$.

Table 2: Example distance matrix, window size = 3, Δb (basic distance) = 7.48

	Kitten	cat	Dog	car
Kitten	0	3.1	6.9	12.3
Cat	3.1	0	5.8	14.4
Dog	6.9	5.8	0	11.7
Car	12.3	14.4	11.7	0

Table 2 is an example of similar symbols being found once the distance matrix is calculated. This example uses a window size of three. By using Equation 2, Δb can be found.

This value is 7.48. Recall that the lower the distance the more similar two symbols are, symbols with a lower distance than Δb indicate evidence of similarity; distances greater then Δb represent more evidence of dissimilarity than similarity. In this example, then, the symbols "cat" and "kitten" have the lowest distance value. It can therefore be assumed that these are the most similar symbols.

The symbols "dog" and "kitten" have a distance value of 6.9. This is slightly lower then Δb. A dog and a kitten are both similar because they are pets. However, although both may be pets, they remain very different. This is why the similarity threshold is used. This research used the ratio value .8. Given Δb of 7.48, T = 5.98 so only strong similarities are now produced.

It remains necessary to calculate the similarity of web pages. SCA inspects a web page and counts the occurrences of the members of the set of symbols in the complete set of pages. This count is compared to that of other web pages to find similarities. To integrate the HAL algorithm in this researche's NCA, contextually similar symbols were added to the SCA symbol list. These contextually similar symbols were found using the NCA distance matrix. For each set of web pages a similarity score is determined. This score is the

frequency of times the symbols from the SCA symbol list appeared in the web page being compared. The HAL context data is added to the SCA symbol list and then the scores are determined (NCA scores). These scores are then displayed to the user in a sorted list.

Data Collection

It was deicide that for this research it would be useful to use web pages that have obvious differences and similarities. The vast number of topics found on the Internet would likely return web pages that had no similarities if a small number of these pages were randomly selected. For this reason it was decided to collect web pages “by hand”.

Three categories were chosen: baseball news articles, football news articles, and Iraqi war news articles. It was assumed that each article would be similar to the other articles in its category.

To test the sensitivity of NCA some baseball and football news articles were centered on Philadelphia teams. The rationale behind this selection is that two Philadelphia baseball articles should be grouped closer than two baseball articles pertaining to different cities. In addition, a

Philadelphia football article when compared to a Philadelphia baseball article would score better than it would when compared to a football article about a different city. The reference to Philadelphia provides some contextual similarity. The Iraqi war articles were added as contrast data. It was anticipated that the Iraqi war news articles should have little or nothing in common with the football and baseball articles.

The first test started with seven web pages. The second test increased to fourteen web pages. The last test reached twenty-eight web pages. The sample sets contained more baseball articles than football or Iraqi war news, in order to allow NCA to learn more context data about one subject than another. It was not thought to be important that NCA to have in-depth knowledge of all three categories. It was considered more beneficial for this research that NCA have such knowledge in only one category. This should show an expectation of improved results as data, time and computing resources increase.

Obviously this sample data set represents a miniscule portion of possible data available. Yet there is no reason to believe that the NCA model should be less useful when applied to other categories. In fact, if more diverse data were included, it is expected that NCA’s results would

become more accurate due to the added depth of context learned by HAL.

Limitations and Complexity

Certain parts of this research were restricted due to the limits of modern PCs. As was discussed, the largest sample data set was only twenty-eight web pages. Ideally, as this research tries to prove, the NCA algorithm would make its most intelligent Internet data groupings if all the pages on the Internet could be processed. However, the retrieval and processing of this much data is not possible.

Although retrieval was a concern, the required processing power was immense. The cost of processing the data outweighed any other constraint. The small sample set of twenty-eight web pages contained 25,485 total symbols. Of these 25,485 symbols 6,364 were unique. To process this data, two 6,364 x 6,364 sized matrices were created. The complexity of finding the HAL scores was high, but not nearly so high as the complexity of finding the distance scores.

The number of calculations required to find the distance matrix can be found with the following equation: $c = 4n(n^2)$. From this equation the order of complexity for finding the

distance matrix can be derived which is $O(n^3)$ where n is equal to the number of unique symbols.

The distance matrix contained 40,500,496 ($6,364^2$) distances that need to be calculated. For each one of these distances, the distance equation contained 12,728 (6,364 * 2) addition and multiplication problems for a total of 25,456 (12,728 * 2). In total about 1,030,980,626,176 (40,500,496 * 25,456) calculations had to be computed to find the distances for a set of twenty-eight web pages. If just one unique symbol were added to this set, it would increase the total calculations to 1,031,466,708,500. An increase of 486,082,324 calculations for the addition of just one unique symbol! The problem will continue to grow exponentially.

In this research in a very small set of data, computation was performed using the Linux operating system running on a dual processor (hyperthreaded to present itself as a quad processor), shared memory parallel processing 64-bit system with eight gigabytes of memory. The calculations were spread across all four logical processors to speed up the program. Even with this relatively powerful hardware, twenty-eight web pages needed nearly a full day to completely process.

Despite the computational complexity, however, heuristics indicate that in real world situations it might be possible

to process more data. As pages are added it is anticipated that fewer novel symbols will be introduced by the data. It is safe to assume that eventually every possible symbol will likely be represented. At that point there would be virtually no additional cost to add a web page. And, this point might be reached rather quickly. There are about 54,000 words in modern English. (Nation, Waring 1997) However, 16,000 words represent 98% of most written English (Chafe, Danielewicz 1987). Once this written vocabulary is represented, adding new web pages becomes much less resource intense.

Summary

This research was performed with the aid of two custom made tools. The first was a HTML parser that removed HTML tags. The second was NCA incorporating HAL and SCA. Data was collected in a nonrandom fashion; it was limited to three distinct categories due to the purpose of this research. The data was intentionally selected so that certain web pages would be similar or dissimilar. This data was then processed and the results were analyzed. The high costs of processing this data required use of a high-end computing system. Although the computation complexity is $O(n^3)$, heuristics indicate a computable soft upper bound as the size of that data increases.

Results of Research

Results

This research was performed on three sets of data. The first was on a group of data containing seven web pages. Three web pages were articles about baseball, two web pages were articles about football and the final two web pages were articles about the war in Iraq. The next test was conducted on fourteen web pages. The last test was conducted on twenty-eight web pages. Both the fourteen web page and twenty-eight web page tests maintained the ratio of topics established for the first test. The two larger tests were supersets of the smaller test.

Once all the distance calculations are complete, a web page is arbitrarily selected. A list of all other web pages is displayed with a score assigned. The higher the score the more similar the content of that web page in comparison to the original. Representative results are shown here. The results are selected as a fair representation of the strengths and weaknesses of the NCA al6gorithm.

Each set of data contains a target page that other pages were compared to. The top five highest scoring web pages are listed when the NCA algorithm is used. The order of their listing is descending order based on the score assigned using NCA. The table lists the points gained that the HAL algorithm added to the SCA score. Each target article was chosen from the three categories (baseball, football, Iraqi war). All NCA generated scores are equal to or greater than the SCA score. This is because the use of HAL only adds context data, which in turn can only raise the score.

Table 3: 1st Seven web page data set,

Target article "MLB recap Reds at Phillies May 15" (baseball)

Points gained from HAL	NCA score	SCA score	Title	Category
21	183	162	MLB recap Reds at Phillies May 13th	baseball
24	145	121	MLB recap Pirates at Cardinals May 23th	baseball
0	92	92	No going across the middle	football
6	78	72	9 US troops killed in Iraq	Iraqi war
0	64	64	US forces encircle Iraqi city in new offensive	Iraqi war

Table 4: 2nd Seven web page data set,

Target article "9 US troops killed in Iraq" (Iraqi war)

Points gained from NCA	NCA score	SCA score	Title	Category

7	93	86	US forces encircle Iraqi city in new offensive	Iraqi war
10	91	81	No going across the middle	football
5	78	73	MLB recap Reds at Phillies May 15th	baseball
5	77	72	MLB recap Reds at Phillies May 13th	baseball
3	70	67	MLB recap Pirates at Cardinals May 23th	baseball

Table 5: 1st Fourteen web page data set,

Target article "MLB recap Reds at Phillies May 15" (baseball)

Points gained from NCA	NCA score	SCA score	Title	Category
19	181	162	MLB recap Reds at Phillies May 13th	baseball
19	139	120	MLB recap Pirates at Cardinals May 23th	baseball
0	92	92	No going across the middle	football
0	90	90	Oil for food program investigation	Iraqi war
0	77	77	Rice to compete for spot as Denver's No. 4	football

Table 6: 2nd Fourteen web page data set,

Target article "9 US troops killed in Iraq" (Iraqi war)

Points gained from NCA	NCA score	SCA score	Title	Category
11	118	107	Oil for food program investigation	Iraqi war
3	89	86	US forces encircle Iraqi city in new offensive	Iraqi war
6	89	83	US unleashes surprise offensive in Iraq	Iraqi war
6	87	81	No going across the middle	football
3	76	73	MLB recap Reds at Phillies May 15th	baseball

Table 7: 1st Twenty-eight web page data set,

Target article "MLB recap Reds at Phillies May 15" (baseball)

Points gained from NCA	NCA score	SCA score	Title	Category
16	178	162	MLB recap Reds at Phillies May 13th	baseball
12	133	121	Phillies Ranger Recap June 7th	baseball
11	132	121	MLB recap Pirates at Cardinals May 23th	Baseball
1	126	125	MLB Pedro dominant	baseball
0	120	120	Angels Brave recap June 7th	baseball

Table 8: 2nd Twenty-eight web page data set,

Target article "9 US troops killed in Iraq" (Iraqi war)

Points gained from NCA	NCA score	SCA score	Title	Category
1	128	127	Iraqi president defends Shiite militia	Iraqi war
7	114	107	Oil for food program investigation	Iraqi war
4	103	99	US and Iraqi troops launch Tal Afar offensive	Iraqi war
0	67	94	MLB Pedro dominant	baseball
3	89	86	US forces encircle Iraqi city in new offensive	Iraqi war

This data demonstrates that the web pages with a clear relationship to the target web page gain the most by using NCA. For example in Table 3, when the target "MLB recap Reds at Phillies May 15th" and is compared to "MLB recap Reds at Phillies May 13th", "MLB recap Reds at Phillies May 13th"

gains the most points from HAL in the seven page test. This is no surprise because both web pages are extremely similar. In Table 3 when "9 US troops killed in Iraq" is compared to the same web page, HAL gives no extra points. This trend will likely continue as the data sets become larger. Unexpectedly, the larger the data set, the less points HAL assigned on average. This was true for both the similar articles and for not so similar articles.

Summary

Data was generated from three tests containing different data set sizes. Each test was performed with two different target web pages. The number of points NCA added was shown for each of the top five scoring web pages. NCA added the most points to similar pages and added little to dissimilar pages. As the data set size increased the number of points NCA contributed slightly decreased.

Conclusions and Implications

Summary of Research

The purpose of this research was to determine if the NCA algorithm using HAL in conjunction with SCA would produce better, more intelligent results than the SCA algorithm alone to group web pages based on similarity of content. NCA implemented using the HAL algorithm's matrix to find each symbol's vector point in context space. NCA implemented a distance matrix to determine relationships among all symbols. The distance matrix is resource intensive to calculate, having $O(n^3)$ complexity.

The data set used was small due to complexity issues and the focus of this research. The Data set sizes were seven, fourteen, and twenty-eight web pages. Web pages were limited to three categories (baseball, football and Iraqi war articles) so that results could be easily studied. The NCA program processed the data using the HAL algorithm and compared its results to SCA. This was done to measure the influence of the HAL algorithm. Once all the computations were complete, the results were compared.

Conclusions

In all tests, HAL generally improved the score of similar pages more than it improved the score for non-similar pages. For example, in Table 5 a baseball content web page is the target. The two top baseball web pages are each given 19 addition points when HAL is used. The football and Iraqi war web pages are given no extra points. This is a good example of how HAL improves the results.

On the other hand, Table 6 has an Iraqi war content web page as the target. In this example HAL adds 11 additional points to another Iraqi war content web page. However, the rest of the results are somewhat problematic. The second and third web pages are both Iraqi war related, but only gain 3 and 6 points respectively from HAL. The fourth web page is about football and HAL increases its score by 6 addition points. The last web page is about baseball and gains 3 points from HAL. In this example HAL is adding the same or more points to the football web page as it does for the two prior Iraqi war web pages.

These Table 6 results are clearly less supportive to the hypothesis than are those in Table 5. This is not unexpected. The HAL algorithm learns contextual meanings of

words by processing data. The more data it can process the more meaning it can learn. Since this research is using relatively small sets of data, HAL has only a very limited understanding of the contextual meanings of words it processes. In order to show some stronger results the categories are not evenly represented. As mentioned earlier, the data sets contained proportionately more baseball web pages then any other web page category. This was done with the expectation that it would provide the HAL algorithm with a relatively stronger knowledge of baseball vocabulary. The results support this expectation. This is evident by comparing the results from Table 3 to Table 4, Table 5 to Table 6 and Table 7 to Table 8.

These results strongly support the hypothesis for this research: The HAL algorithm in conjunction with SCA does provide more accurate results. However, the data also shows a trend that as more data is added the effects of NCA decreases. There seem to be three possible reasons for this. Recall that the algorithm was tested using very small sets of data. Since previous research has shown that the HAL algorithm improves its knowledge as more data is processed, (Burgess 1998) it is very possible that the tests that were scored higher and contained the smaller data sets, were proportionally over scored. As HAL becomes better trained

(upon inclusion of more data), scores become more accurate in their representation of the data's contextual meaning.

A second possible reason for the difficulty in handling the Iraqi war articles could derive from the many uses of metaphor in describing war and sports. In sports the term "rocket" may be used to describe a pitcher's arm, or a good pass from a quarterback. But in an article about the Iraqi war, a rocket is most likely an explosive projectile. This may have caused difficulty for HAL. Since a small set of data was used HAL may have interpreted these metaphors as similarities, and missed other relevant contextual similarities that would have over-shadowed these metaphors.

The final possible reason is the basic distance, Δb, and the threshold T. T was computed using an experimentally determined ratio found to generally give good results. This ratio tunes the results of the algorithm. The context space was very volatile due to the small data sets used. This makes having only one ratio tuned appropriately for all data sets nearly impossible. As the data set size increase, the context space becomes more stable. A simple ratio might be then found to work well across many (or all) the larger sample sizes.

Implications

This research shows that HAL has benefits when used with SCA to find matching web pages. However, the complexity of using the algorithm makes its practical use limited.

Future Research

This research demonstrates that there is a potential benefit from using a HAL model with a SCA algorithm to group Internet web pages based on their content. However this research has also shown that to achieve this, a very high level of computation resources is needed. In its current state, the algorithm is not practical for casual use. Although some optimizations were effective, their impact is insufficient to overcome the polynomial growth of calculations needed to be performed as unique symbols are added. However there is reason to believe that this rate of growth is not sustainable and so may not be as critical as it seems. Once the algorithm has encountered most common symbols, any new web page will only add a few new symbols. It is safe to say at some point most web pages will provide no new symbols, but only context data. Currently every time a new web page is added the whole set of data is processed again. It seems very likely that this would not be needed after some relatively large number of pages has been

processed. As symbols are added, there is no reason that the algorithm could not dynamically update its internal data structures to reflect appropriate changes. Over time this would substantially reduce the computational power required to use the algorithm. Once the algorithm has become more efficient, research should be conducted on a larger set of data to determine if it produces better results as more data is added.

Summary

The results do show that there is a benefit to using NCA to gain contextual meaning. There the results also support the notion that the HAL algorithm will prove less influential as the data size grows. Larger data sets would not only provide a better-trained NCA algorithm, but also provide more stability to the context space used to store the contextual relationships. With a more stable context space, a more appropriate ratio might be found. This would improve the results on larger data sets over a range of sizes.

Bibliography

Bibliography

Broder, Andrei, Ravi Kumar, Farzin Maghoul, Praghakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins and Janet Wiener. Graph structure in the web. Proceedings of the Ninth International World Wide Web Conference, 2000.

Brooks, Terrence. The Semantic Distance Model of Relevance Assessment. Information Access in the Global Information Economy, Vol 35, Proceedings of the 61st Annual Meeting of ASIS, 33-44, 1998.

Burgess, Curt. From simple associations to the building blocks of language: Modeling meaning in memory with the HAL model. Behavior Research Methods, Instruments, & Company, 189-198, 1998.

Burgess, Curt and Kevin Lund. Modelling Parsing Constraints with High-dimensional Context Space. Psychology Press, part of the Taylor & Francis Group, Vol 12, 2-3, 1997.

Chafe, Wallace and Jane Danielewicz. Properties of Spoken and Written Language. Comprehending Oral and Written Language, Academic Press, 1987.

Churchill, Christine. Search Engine Algorithms & Research. SearchEngineWatch.com, 2005.

Haveliwala, Taher and Sepandar Kamvar. The Second Eigenvalue of the Google Matrix. preprint, 2003.

Kennedy, James, Russell Eberhart and Yuhui Shi. Swarm intelligence. Morgan Kaufman Publishers, 2001.

Hintzman, DL. Human Learning and Memory: Connections and Dissociations. Annual Review of Psychology, Vol 41, 109-139, 1990.

Landauer, Thomas and Susan Dumais. A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. Psychological Review, Vol 104, 211-240, 1997.

Landauer, Thomas and Susan Dumais. Memory model reads encyclopedia, passes vocabulary test. Presented at the Psychonomics Society, 1994.

Landauer, Thomas, Peter Foltz and Darrel Laham. An Introduction to Latent Semantic Analysis. Discourse Processes, Vol 25, 259-284, 1998.

Nation, Paul and Robert Waring. Vocabulary size, text coverage and word lists. Vocabulary: Description, Acquisition and Pedagogy: Cambridge, Cambridge University Press, 6-19, 1997.

Osgood, Charles, George Suci and Percy Tannenbaum. The measurement of meaning. University of Illinois Press, 1957.

Ramscar, Michael and Daneil Yarlett. The use of a high-dimensional, "environmental" context space to model retrieval in analogy and similarity-based transfer. Presented at the Twenty Second Annual Meeting of the Cognitive Science Society, 2000.

Schutze, Hinrich. Dimensions of meaning. Proceedings of Supercomputing, New York: Association for Computing Machinery, 787-796, 1992.

Stwyvers, Mark, Richard Shiffrin and Douglas Nelson. Word Association Spaces for Predicting Semantic Similarity Effects in Episodic Memory. In A. Healy (Ed.), Experimental Cognitive Psychology and its Applications: Festschrift in Honor of Lyle Bourne, Walter Kintsch, and Thomas Landauer, American Psychological Association.

Appendix A

Definitions and Operational Terms

High-dimensional memory model: A class of algorithms that contain a high dimensional space (context space) used to represent relationships among symbols.

Hyperspace analogue to language model of memory (HAL): A machine-learning algorithm that is part of the high dimensional memory model class. HAL has the ability to learn the contextual meanings of symbols.

Simple comparison algorithm: An algorithm that processes sets of data and determines the extent of their similarity. This algorithm simply counts similar occurrences of symbols between two files and generates a score.

Basic distance: A term used for the distance of two symbols that each only appear once in the entire body of data and never appear in the same sliding window. These two symbols therefore, have no evidence of contextual similarities. If the distance of two symbols is less than the basic distance value, it represents evidence of contextual similarity. If it is greater than the basic distance value, it represents evidence of contextual dissimilarity.

Context space: The high dimensional space (hyperspace) that all symbols are located in. Within this hyperspace the contextual distance of two words can be found.

NCA: The algorithm that was created for this research. It is a combination of the HAL algorithm and SCA.

Appendix B

```
#!/bin/sh
#set -x


####################
#Process the files
ProcessFile()
{
echo "$1"
#all variables must be local since recurision is used
local currentFile=$1
local newName=`cat $currentFile |grep title\>|sed s/^......//g|sed
s/........$//g|tr A-Z a-z`
local newNameClean=`echo $newName | sed -e 's/[ ]/_/g'`
local toRemove=`echo $currentFile |sed -e s/"^.*\/"//`
local workingPath=`echo $currentFile |sed -e s/$toRemove$//`
#sed expression used to get rid of most html
local output=`sed -e 's:<[bB][rR]/*>:\
:g' \
    -e '/</{
       :loop
       s/<[^>]*>//g
        /</{
            N
            b loop
        }
     }' \
    -e 's:&[nN][bB][sS][pP];: :g' \
    -e 's:&..;::g; s:&...;::g; s:&....;::g'    \
    $currentFile|tr A-Z a-z`
local safeFileName=`echo "$newNameClean"|sed -e 's/[/]//g'`
if [ "$safeFileName" != "" ]
then
#output clean file
echo $output > output/$safeFileName
fi
}


####################
#Go though the folder of data and process it

GoThroughFolders(){
local currentDir=$1/
#types of files that are not html
local delTypes="pdf swf aif wav mov wma mp3 mpg aiff gif jpg wmv avi exe"
for type in $delTypes
do
```

```
rm  -f $currentDir*.$type
done
for files in `ls $currentDir`
#set up the recursion to go though this folder and all sub folders
do
       if [ -d $currentDir$files ]
       then
              GoThroughFolders $currentDir$files
       else
              ProcessFile $currentDir$files
       fi
done

}


####################
#Start script

if [ -d $1 ]
then
       mkdir output
       GoThroughFolders $1
else
       echo "Usage:"
       echo "       $0 <directory to start in>"
fi
```

Appendix C

HAL.java

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

/*
 * HAL.java created on Apr 18, 2005 5:42:46 PM
 */
/**
 * @author Edward G. Finegan <br>
 *         ed@dryrain.net <br>
 *         <br>
 *         Class HAL <br>
 */
public class HAL implements
    ActionListener {

  private MemoryMatrix     matrix;
  private final int        MAX_MATCHES      = 5;
  private final String     OBJECT_FILE_NAME = "##MEMORY_MATRIX.OBJ";
  private JFrame           frame;
  private JButton          compareButton;
  private JButton          compareButtonNormal;
  private JButton          startButton;
  private JButton          browseButton;
  private JList            nlist;
  private JList            outnList;
  private File             allFiles[];
  private String           dataLocation;
  private JTextField       fileField;
  private JScrollPane      list;
  private JScrollPane      outList;
  private DefaultListModel model;
```

```
  private DefaultListModel outModel;
  private File             dataSource;
  private File             objectFile;

  public static void main(String[] args) {
    if (args.length == 1) {
      HAL hal= new HAL(args[0]);
    }
    else {
      HAL hal= new HAL();
    }
  }

  public HAL() {
    makeGUI();
  }

  public HAL(String dataLocation) {
    this.dataLocation= dataLocation;
    this.objectFile= new File(
        dataLocation + "/"
            + OBJECT_FILE_NAME);
    this.dataSource= new File(
        dataLocation);
    if (!dataSource.isDirectory()) {
      System.out
          .println("The directoy containing the data must be given as a
argument!!!");
      System.exit(-1);
    }
    makeGUI();
    fileField.setText(dataLocation);
  }

  private void startWork() {
    /*
     * Check to see if we have this oject saved. If it is open it and use it, if
     * not make it and save it.
     */
    if (!objectFile.exists()) {
      matrix= new MemoryMatrix(
          dataSource);
      matrix.start();
      ProgressBar progressBar= new ProgressBar(
          frame, matrix);
      while (!progressBar.complete) {
        progressBar.display();
      }
      if (!matrix.cancled) {
        try {
          // Write to disk with FileOutputStream
          FileOutputStream f_out= new FileOutputStream(
              objectFile);
          // Write object with ObjectOutputStream
          ObjectOutputStream obj_out= new ObjectOutputStream(
              f_out);
          // Write object out to disk
          obj_out.writeObject(matrix);
        }
        catch (IOException e) {
          System.out
              .println("Unable to save matrix object!!");
          e.printStackTrace();
        }
      }
    }
    else {
      try {
```

```
      // Read from disk using FileInputStream
      FileInputStream f_in= new FileInputStream(
          objectFile);
      // Read object using ObjectInputStream
      ObjectInputStream obj_in= new ObjectInputStream(
          f_in);
      // Read an object
      matrix= (MemoryMatrix)obj_in
          .readObject();
    }
    catch (Exception e) {
      System.out
          .println("Cannot open object!!");
      e.printStackTrace();
    }
    matrix.reopened();
  }
}

private void makeGUI() {
  /*
   * Make the GUI
   */
  frame= new JFrame("Memory Matrix");
  frame
      .setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  JPanel window= new JPanel();
  window
      .setLayout(new BorderLayout());
  JPanel topWindow= new JPanel();
  model= new DefaultListModel();
  nlist= new JList(model);
  list= new JScrollPane(nlist);
  topWindow.add(list);
  JPanel bottomWindow= new JPanel();
  startButton= new JButton("Start");
  startButton.addActionListener(this);
  bottomWindow.add(startButton);
  compareButton= new JButton(
      "Find Matches");
  compareButton
      .addActionListener(this);
  compareButton.setEnabled(false);
  bottomWindow.add(compareButton);
  compareButtonNormal= new JButton(
      "Find Matches WO Alog.");
  compareButtonNormal
      .addActionListener(this);
  compareButtonNormal
      .setEnabled(false);
  bottomWindow
      .add(compareButtonNormal);
  /*
   * create a file selection box
   */
  JPanel fileWindow= new JPanel();
  JLabel selectFile= new JLabel(
      "Select Directory");
  fileField= new JTextField(35);
  browseButton= new JButton("Browse");
  browseButton
      .addActionListener(this);
  bottomWindow.add(browseButton);
  /*
   * create a place of output
   */
  JPanel subWindow= new JPanel();
  subWindow
```

```
        .setLayout(new BorderLayout());
    outModel= new DefaultListModel();
    outnList= new JList(outModel);
    outList= new JScrollPane(outnList);
    JPanel outWindow= new JPanel();
    outWindow.add(outList);
    fileWindow.add(selectFile);
    fileWindow.add(fileField);
    fileWindow.add(browseButton);
    window.add(fileWindow,
        BorderLayout.NORTH);
    window.add(list,
        BorderLayout.CENTER);
    subWindow.add(bottomWindow,
        BorderLayout.NORTH);
    subWindow.add(outList,
        BorderLayout.SOUTH);
    window.add(subWindow,
        BorderLayout.SOUTH);
    frame.getContentPane().add(window);
    frame.pack();
    frame.show();
  }

  public void actionPerformed(
      ActionEvent evt) {
    if (evt.getSource() == browseButton) {
      JFileChooser chooser= new JFileChooser();
      chooser
          .setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
      int returnVal= chooser
          .showOpenDialog(frame);
      /*
       * If ok is selected we go in the if and retrive the file that is selected
       */
      if (returnVal == JFileChooser.APPROVE_OPTION) {
        fileField.setText(chooser
            .getSelectedFile()
            .toString());
      }
    }
    /*
     * Start Button
     */
    if (evt.getSource() == startButton) {
      this.dataLocation= fileField
          .getText();
      this.objectFile= new File(
          dataLocation + "/"
              + OBJECT_FILE_NAME);
      this.dataSource= new File(
          dataLocation);
      if (!dataSource.isDirectory()) {
        JOptionPane
            .showMessageDialog(
                frame,
                "The directory that was enter is invaild. Please enter a\nvaild
directory, or select one using the browse button.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
      }
      else {
        /*
         * display a message before starting work
         */
        if (!objectFile.exists()) {
          int answer= JOptionPane
              .showConfirmDialog(
```

```
                    frame,
                    "There was no precalculated language file found. Would you\n"
                        + "like to calculate one now? (This may take a long
time!!!)",
                    "Object File found",
                    JOptionPane.OK_CANCEL_OPTION,
                    JOptionPane.QUESTION_MESSAGE);
          if (answer == JOptionPane.OK_OPTION) {
            /*
             * Make dialog about processor optimatation
             */
            startButtonGO();
          }
        }
        else {
          Object[] options= {"OK",
              "Recalculate", "Cancle"};
          int answer= JOptionPane
              .showOptionDialog(
                  frame,
                  "A precalculated language file was found. Whould you like to
use this file,\n"
                      + "or recalcuate a new file? (Recalcuating may take a long
time!!!)",
                  "Lanuage File Found",
                  JOptionPane.DEFAULT_OPTION,
                  JOptionPane.QUESTION_MESSAGE,
                  null, options,
                  options[0]);
          /*
           * answer =0 ==OK =1 ==RECALCULATE =2 ==CALCLE
           */
          if (answer == 0) {
            startButtonGO();
          }
          else if (answer == 1) {
            objectFile.delete();
            startButtonGO();
          }
        }
      }
    }
    if (evt.getSource() == compareButtonNormal) {
      int selection= nlist
          .getSelectedIndex();
      try {
        String nearSymbols[]= matrix
            .getAllNearSymbols(allFiles[selection]);
        String mainFileSymbols[]= matrix
            .getParsedData(allFiles[selection]);
        String symbol[]= new String[mainFileSymbols.length];
        int symbolCount[]= new int[mainFileSymbols.length];
        /*
         * null out array
         */
        for (int i= 0; i < symbol.length; i++) {
          symbol[i]= null;
          symbolCount[i]= 0;
        }
        /*
         * fills the (real)symbol array and the array that tracks the number
         * each one is found
         */
        for (int i= 0; i < mainFileSymbols.length; i++) {
          String currentSymbol= mainFileSymbols[i];
          for (int j= 0; j <= i; j++) {
            if (symbol[j] == null) {
              symbol[j]= currentSymbol;
```

```
        symbolCount[j]= 1;
        j= i;
      }
      else if (symbol[j]
          .equalsIgnoreCase(currentSymbol)) {
        symbolCount[j]+= 1;
        j= i;
      }
    }
  }
  /*
   * Find the highest reoccuring sysmbol
   */
  int highestCount= 0;
  int highestSymbol= 0;
  for (int i= 0; i < symbolCount.length; i++) {
    if (highestCount < symbolCount[i]) {
      highestCount= symbolCount[i];
      highestSymbol= i;
    }
  }
  /*
   * count the symbols
   */
  int fileScores[]= new int[allFiles.length];
  for (int i= 0; i < allFiles.length; i++) {
    if (i != selection) {
      String symbolsToCompare[]= matrix
          .getParsedData(allFiles[i]);
      for (int j= 0; symbol[j] != null; j++) {
        for (int k= 0; k < symbolsToCompare.length; k++) {
          int symbolMatchCount= 0;
          if (symbol[j]
              .equalsIgnoreCase(symbolsToCompare[k])) {
            fileScores[i]++;
            symbolMatchCount++;
          }
        }
      }
    }
  }
  /*
   * print out the top results
   */
  String topScoresName[]= new String[MAX_MATCHES];
  int topScores[]= new int[MAX_MATCHES];
  for (int i= 0; i < MAX_MATCHES; i++) {
    topScores[i]= 0;
    topScoresName[i]= "EMPTY";
  }
  /*
   * rank top x number scores
   */
  outModel.clear();
  RankedResults results= new RankedResults(
      fileScores, allFiles);
  for (int i= 0; i < results
      .sizeOf(); i++) {
    outModel.add(outModel
        .getSize(), results
        .getScore(i)
        + " -- "
        + results.getName(i));
  }
}
catch (ArrayIndexOutOfBoundsException e) {
  JOptionPane
      .showMessageDialog(
```

```
            frame,
            "Please select a file to find matches agaist.",
            "",
            JOptionPane.WARNING_MESSAGE);
  }
}
if (evt.getSource() == compareButton) {
  int selection= nlist
      .getSelectedIndex();
  try {
    String nearSymbols[]= matrix
        .getAllNearSymbols(allFiles[selection]);
    String mainFileSymbols[]= matrix
        .getParsedData(allFiles[selection]);
    String symbol[]= new String[nearSymbols.length];
    int symbolCount[]= new int[nearSymbols.length];
    /*
     * null out array
     */
    for (int i= 0; i < symbol.length; i++) {
      symbol[i]= null;
      symbolCount[i]= 0;
    }
    /*
     * fills the (real)symbol array and the array that tracks the number
     * each one is found
     */
    System.out
        .println("nearSymbols = "
            + nearSymbols.length
            + " mainFileSymbols = "
            + mainFileSymbols.length);
    for (int i= 0; i < nearSymbols.length; i++) {
      String currentSymbol;
      currentSymbol= nearSymbols[i];
      for (int j= 0; j <= i; j++) {
        if (symbol[j] == null) {
          symbol[j]= currentSymbol;
          symbolCount[j]= 1;
          j= i;
        }
        else if (symbol[j]
            .equalsIgnoreCase(currentSymbol)) {
          symbolCount[j]+= 1;
          j= i;
        }
      }
    }
    /*
     * Find the highest reoccuring sysmbol
     */
    int highestCount= 0;
    int highestSymbol= 0;
    for (int i= 0; i < symbolCount.length; i++) {
      if (highestCount < symbolCount[i]
          && !(symbol[i]
              .equals("the"))
          && !(symbol[i]
              .equals(" "))
          && !(symbol[i]
              .equals("a"))
          && !(symbol[i]
              .equals("and"))
          && !(symbol[i]
              .equals("in"))
          && !(symbol[i]
              .equals("is"))
          && !(symbol[i]
```

```
          .equals("of"))
      && !(symbol[i]
          .equals("to"))
      && !(symbol[i].equals(""))) {
    highestCount= symbolCount[i];
    highestSymbol= i;
  }
}
System.out
    .println("Highest Count = "
        + highestCount
        + " for "
        + symbol[highestSymbol]
            .charAt(0) + "<<"
        + symbol[highestSymbol]
        + "<<");
/*
 * count the symbols
 */
int fileScores[]= new int[allFiles.length];
for (int i= 0; i < allFiles.length; i++) {
  if (i != selection) {
    String symbolsToCompare[]= matrix
        .getParsedData(allFiles[i]);
    for (int j= 0; symbol[j] != null; j++) {
      for (int k= 0; k < symbolsToCompare.length; k++) {
        int symbolMatchCount= 0;
        if (symbol[j]
            .equalsIgnoreCase(symbolsToCompare[k])) {
          fileScores[i]++;
          symbolMatchCount++;
          System.out
              .println("highestCount "
                  + highestCount
                  + "symbolMatchCount "
                  + symbolMatchCount);
          if (symbolMatchCount > Math
              .ceil(highestCount / 4)) {
            k= symbolsToCompare.length;
            System.out
                .println("CUTOFF");
          }
        }
      }
    }
  }
}
/*
 * print out the top results
 */
String topScoresName[]= new String[MAX_MATCHES];
int topScores[]= new int[MAX_MATCHES];
for (int i= 0; i < MAX_MATCHES; i++) {
  topScores[i]= 0;
  topScoresName[i]= "EMPTY";
}
/*
 * rank top x number scores
 */
outModel.clear();
RankedResults results= new RankedResults(
    fileScores, allFiles);
for (int i= 0; i < results
    .sizeOf(); i++) {
  outModel.add(outModel
      .getSize(), results
      .getScore(i)
      + " -- "
```

```
              + results.getName(i));
        }
      }
      catch (ArrayIndexOutOfBoundsException e) {
        JOptionPane
            .showMessageDialog(
                frame,
                "Please select a file to find matches agaist.",
                "",
                JOptionPane.WARNING_MESSAGE);
      }
    }
  }

  private void startButtonGO() {
    startWork();
    if (!matrix.cancled) {
      allFiles= matrix.getAllFiles();
      model.clear();
      for (int i= 0; i < allFiles.length; i++) {
        model.add(model.getSize(),
            allFiles[i].getName());
      }
      compareButton.setEnabled(true);
      compareButtonNormal
          .setEnabled(true);
      startButton.setLabel("Restart");
      frame.pack();
    }
  }
}
```

MemoryMatrix.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Vector;

/*
 * MemoryMatrix.java created on Apr 25, 2005 4:36:35 PM
 */
/**
 * @author Edward G Finegan ed@dryrain.net
 */
public class MemoryMatrix extends
    Thread implements
    java.io.Serializable {

  /*
   * Constants
   */
  private int          distanceCounter;
  private final int    WINDOW_SIZE              = 10;
  private final double BD_RATIO                 = 0.8;
  private int          numThreads               = 1;
  private final String MEMORY_MATRIX_FILE_NAME   = "##MEMORY_MATRIX.TXT";
  private final String DISTANCE_MATRIX_FILE_NAME = "##DISTANCE_MATRIX.TXT";
  /*
   * class varables
   */
  public int           matrixSize;
```

```
private File[]       allFiles;
private Vector       allSymbolsUnique;
private int          totalSymbols;
public short[][]     matrix;
public float[][]     distanceMatrix;
public boolean[]     completedRows;
private File         memoryMatrixFile;
private File         distanceMatrixFile;
private double       basicDistance;
private File         dataSource;
public boolean       cancled                    = false;

public void stopCalc() {
  distanceCounter= matrixSize - 1;
}

public void run() {
  distanceCounter= 0;
  findDistances();
}

public MemoryMatrix(File dataSource) {
  this.dataSource= dataSource;
  Runtime r= Runtime.getRuntime();
  numThreads= r.availableProcessors();
  System.out.println("Optimized for "
      + numThreads + " CPUs");
  memoryMatrixFile= new File(
      dataSource.toString() + "/"
          + MEMORY_MATRIX_FILE_NAME);
  distanceMatrixFile= new File(
      dataSource.toString() + "/"
          + DISTANCE_MATRIX_FILE_NAME);
  cleanDir(dataSource);
  allFiles= dataSource.listFiles();
  allSymbolsUnique= new Vector();
  totalSymbols= 0;
  findAllSymbols();
  fillMatrix();
  /*
   * Set all distances to -1 Set all rows to false, not completed
   */
  distanceMatrix= new float[matrixSize][matrixSize];
  completedRows= new boolean[matrixSize];
  for (int i= 0; i < matrixSize; i++) {
    completedRows[i]= false;
    for (int j= 0; j < matrixSize; j++) {
      distanceMatrix[i][j]= -1;
    }
  }
}

public void reopened() {
  Runtime r= Runtime.getRuntime();
  numThreads= r.availableProcessors();
  System.out.println("Optimized for "
      + numThreads + " CPUs");
  System.out.println(totalSymbols
      + " total symbols processed");
  System.out
      .println(allSymbolsUnique
          .size()
          + " total unique symbols found");
  System.out.println(basicDistance
      + " Basic Distance");
}

public File[] getAllFiles() {
```

```
  return allFiles;
}

private void cleanDir(
    File dataDirectory) {
  File DSstore= new File(
      dataDirectory.toString()
          + "/.DS_Store");
  DSstore.delete();
}

/**
 * Finds all the symbols in the set of files, and then find the unique symbols
 */
private void findAllSymbols() {
  for (int i= 0; i < allFiles.length; i++) {
    String[] parsedData= getParsedData(i);
    /*
     * Check to see if the word is already in the vector
     */
    if (allSymbolsUnique.size() == 0) {
      allSymbolsUnique
          .add(parsedData[0]);
    }
    for (int j= 0; j < parsedData.length; j++) {
      totalSymbols++;
      boolean notRepeat= true;
      for (int k= 0; k < allSymbolsUnique
          .size(); k++) {
        if (parsedData[j]
            .equalsIgnoreCase((String)allSymbolsUnique
                .get(k))) {
          notRepeat= false;
          k= allSymbolsUnique.size();
        }
      }
      if (notRepeat) {
        allSymbolsUnique
            .add(parsedData[j]);
      }
    }
  }
  matrixSize= allSymbolsUnique.size();
  System.out.println(totalSymbols
      + " total symbols processed");
  System.out
      .println(allSymbolsUnique
          .size()
          + " total unique symbols found");
}

/**
 * Filles the matrix with the symbol data
 */
private void fillMatrix() {
  /*
   * Set it all to 0's
   */
  matrix= new short[matrixSize][matrixSize];
  for (int i= 0; i < matrixSize; i++) {
    for (int j= 0; j < matrixSize; j++) {
      matrix[i][j]= 0;
    }
  }
  /*
   * Go though th files and process the symbols in the matrix
   */
  for (int i= 0; i < allFiles.length; i++) {
```

```
      String[] parsedData= getParsedData(i);
      /*
       * Start to build up the matrix Loop though all the symbols in the file,
       * and then loop though the window area. Record the data to the matrix.
       */
      for (int j= 0; j < parsedData.length; j++) {
        int currentLocationIndex= allSymbolsUnique
            .indexOf(parsedData[j]);
        int distance= WINDOW_SIZE;
        for (int k= j + 1; k <= j
            + WINDOW_SIZE
            && k < parsedData.length; k++) {
          int currentWindowIndex= allSymbolsUnique
              .indexOf(parsedData[k]);
          matrix[currentLocationIndex][currentWindowIndex]+= distance;
          distance--;
        }
      }
    }
  }

  /**
   * Use this to print out the matrix in human readable form
   */
  private void printMatrix() {
    for (int i= 0; i < matrix.length; i++) {
      for (int j= 0; j < matrix.length; j++) {
        System.out.print(matrix[i][j]
            + " ");
      }
      System.out.println();
    }
  }

  /**
   * Use this to print out the distance matrix in human readable form
   */
  private void printDistanceMatrix() {
    for (int i= 0; i < matrixSize; i++) {
      for (int j= 0; j < matrixSize; j++) {
        System.out
            .print(distanceMatrix[i][j]
                + " ");
      }
      System.out.println();
    }
  }

  private void findDistances() {
    /*
     * Find the distances for each symbol related to every other symbol If we
     * are singile threaded don't bother adding the over head of more threads,
     * if we are more then on thread then create them and do it
     */
    if (numThreads == 1) {
      int r= 2;
      for (distanceCounter= 0; distanceCounter < matrixSize; distanceCounter++)
{
        for (int j= distanceCounter; j < matrixSize; j++) {
          long currentDistance= 0;
          for (int k= 0; k < matrixSize; k++) {
            currentDistance+= Math
                .pow(
                    matrix[distanceCounter][k]
                        - matrix[j][k],
                    r);
          }
          for (int k= 0; k < matrixSize; k++) {
```

```
            currentDistance+= Math
                .pow(
                    matrix[k][distanceCounter]
                        - matrix[k][j],
                    r);
          }
          distanceMatrix[distanceCounter][j]= (float)Math
              .pow(currentDistance,
                  1.0 / r);
        }
        completedRows[distanceCounter]= true;
        System.out.println("row "
            + distanceCounter
            + " complete");
      }
    }
    else {
      FindDistances[] findDistancesThreads= new FindDistances[numThreads];
      for (int i= 0; i < numThreads; i++) {
        System.out
            .println("creating thread "
                + i);
        findDistancesThreads[i]= new FindDistances(
            this);
        findDistancesThreads[i].start();
      }
      /*
       * Busy wait here untill the last row is completed
       */
      while (!completedRows[completedRows.length - 1]) {
        if (cancled) {
          findDistancesThreads= null;
        }
        try {
          Thread.sleep(5000);
        }
        catch (Exception e) {
          System.out
              .println("Sleep failed");
          e.printStackTrace();
        }
      }
    }
    /*
     * Find Basic distance
     */
    int total= 0;
    for (int i= 1; i <= WINDOW_SIZE; i++) {
      total+= Math.pow(i, 2);
    }
    basicDistance= Math.sqrt(total * 4);
    /*
     * Find the standared devation
     */
  }

  /**
   * This takes a file, reads it and retunds it in an array.
   *
   * @param fileIndex - int, the number of the file you want to get retruned in
   *          the array
   * @return String[]
   */
  private String[] getParsedData(
      int fileIndex) {
    return getParsedData(allFiles[fileIndex]);
  }
```

```
  public String[] getParsedData(
      File file) {
    String fileData= "";
    String line;
    try {
      BufferedReader fIn= new BufferedReader(
          new FileReader(file));
      /*
       * Go though each line of the file
       */
      while ((line= fIn.readLine()) != null) {
        fileData= fileData + line + " ";
      }
    }
    catch (FileNotFoundException e) {
      System.out
          .println("Was unable to find file: "
              + file);
      e.printStackTrace();
    }
    catch (IOException e) {
      System.out
          .println("IO error with file: "
              + file);
      e.printStackTrace();
    }
    return fileData.replace('\t', ' ')
        .toLowerCase().split(" ");
  }

  public String[] getAllNearSymbols(
      File file) {
    double cutOff= basicDistance
        * BD_RATIO;
    String[] realSymbols= getParsedData(file);
    Vector nearSymbols= new Vector();
    for (int i= 0; i < realSymbols.length; i++) {
      nearSymbols.add(realSymbols[i]);
      int symbolIndex= allSymbolsUnique
          .indexOf(realSymbols[i]);
      /*
       * go though matrix and find the distances that fall below the cutoff
       */
      for (int j= 0; j < matrixSize; j++) {
        if (distanceMatrix[symbolIndex][j] > 0.0
            && distanceMatrix[symbolIndex][j] <= cutOff) {
          nearSymbols
              .add((String)allSymbolsUnique
                  .get(j));
        }
        if (distanceMatrix[j][symbolIndex] > 0.0
            && distanceMatrix[j][symbolIndex] <= cutOff) {
          nearSymbols
              .add((String)allSymbolsUnique
                  .get(j));
        }
      }
    }
    String[] nearSymbolsA= new String[nearSymbols
        .size()];
    for (int i= 0; i < nearSymbols
        .size(); i++) {
      nearSymbolsA[i]= (String)nearSymbols
          .get(i);
    }
    return nearSymbolsA;
  }
}
```

FindDistances.java

```
/*
 * FindDistances.java created on May 26, 2005 12:52:33 PM
 */
/**
 * @author Edward G. Finegan <br>
 *         ed@dryrain.net <br>
 *         <br>
 *         Class FindDistances <br>
 */
public class FindDistances extends
    Thread {

  private MemoryMatrix matrixes;

  FindDistances(MemoryMatrix matrixesIn) {
    matrixes= matrixesIn;
  }

  /*
   * Thread that will run and proccess a row of the matrix
   */
  public void run() {
    int r= 2;
    for (int i= 0; i < matrixes.matrixSize; i++) {
      if (!matrixes.completedRows[i]) {
        matrixes.completedRows[i]= true;
        System.out.println("Row " + i
            + " complete");
        for (int j= i; j < matrixes.matrixSize; j++) {
          long currentDistance= 0;
          for (int k= 0; k < matrixes.matrixSize; k++) {
            currentDistance+= Math
                .pow(
                    matrixes.matrix[i][k]
                        - matrixes.matrix[j][k],
                    r);
          }
          for (int k= 0; k < matrixes.matrixSize; k++) {
            currentDistance+= Math
                .pow(
                    matrixes.matrix[k][i]
                        - matrixes.matrix[k][j],
                    r);
          }
          matrixes.distanceMatrix[i][j]= (float)Math
              .pow(currentDistance,
                  1.0 / r);
        }
      }
    }
  }
}
```

RankedResults.java

```
import java.io.File;
import java.util.Vector;

/*
 * RankedResults.java created on May 24, 2005 7:26:21 PM
 */
```

```
public class RankedResults {

  private Vector rankedNames;
  private Vector rankedScores;

  RankedResults(int[] scores,
      File[] names) {
    rankedScores= new Vector();
    rankedNames= new Vector();
    for (int i= 0; i < scores.length; i++) {
      int highestScore= 0;
      int highestPosition= 0;
      for (int j= 0; j < scores.length; j++) {
        if (scores[j] >= highestScore) {
          highestScore= scores[j];
          highestPosition= j;
        }
      }
      rankedScores.add(new Integer(
          scores[highestPosition]));
      rankedNames
          .add(names[highestPosition]
              .getName());
      scores[highestPosition]= -1;
    }
  }

  public String getName(int index) {
    return ((String)rankedNames
        .get(index));
  }

  public int getScore(int index) {
    return ((Integer)(rankedScores
        .get(index))).intValue();
  }

  public int sizeOf() {
    return rankedScores.size();
  }
}
```

ProgressBar.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JProgressBar;
import javax.swing.Timer;

/*
 * ProgressBar.java created on Jun 2, 2005 10:29:01 PM
 */
/**
 * @author Edward G. Finegan <br>
 *         ed@dryrain.net <br>
 *         <br>
 *         Class ProgressBar <br>
 */
public class ProgressBar extends
    JOptionPane implements
    ActionListener {

  private JFrame       frame;
  private MemoryMatrix matrix;
```

```
  private JProgressBar progressBar;
  private int          amountDone = 0;
  private Timer        timer;
  public boolean       complete   = false;

  public ProgressBar(JFrame frame,
      MemoryMatrix matrix) {
    this.frame= frame;
    this.matrix= matrix;
    display();
  }

  public void display() {
    progressBar= new JProgressBar(0,
        matrix.completedRows.length - 1);
    progressBar.setValue(0);
    progressBar.setStringPainted(true);
    Object[] items= {progressBar};
    Object[] options= {"Continue",
        "Exit"};
    // Object[] options= {};
    timer= new Timer(1000, this);
    timer.start();
    int pick= showOptionDialog(
        frame,
        items,
        "Creating Lanuage Matrix",
        JOptionPane.YES_NO_CANCEL_OPTION,
        JOptionPane.PLAIN_MESSAGE,
        null, options, options[0]);
    /*
     * Exit
     */
    if (pick == 1) {
      timer.stop();
      matrix.cancled= true;
      matrix.stopCalc();
      System.exit(0);
    }
    /*
     * Continue
     */
    else if (pick == 0) {
      timer.stop();
    }
  }

  public void actionPerformed(
      ActionEvent evt) {
    for (int i= amountDone; i < matrix.completedRows.length; i++) {
      if (!matrix.completedRows[i]) {
        amountDone= i;
        i= matrix.completedRows.length;
      }
      if (matrix.completedRows[matrix.completedRows.length - 1]) {
        amountDone= matrix.completedRows.length;
        progressBar
            .setString("Complete!!! Please press continue.");
        complete= true;
        timer.stop();
        Object[] item= {"Done"};
        setOptions(item);
        repaint();
      }
    }
    progressBar.setValue(amountDone);
  }
}
```