The Potential Of Latent Semantic Analysis To Identify Themes In Online Forums

Mark J. Paul

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2007

Program Authorized to Offer Degree:
Institute of Technology - Tacoma

University of Washington
Graduate School

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

Mark J. Paul

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

Committee Members:

_____
Donald Chinn

_____
George Mobus

Date: _____

Signature _____

Date_____

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Table Number Page

GLOSSARY

Corpus – A large collection of text passages.

Forum – A web applications used by groups for holding discussions. It is a message board where members can create threads to discuss a topic, and create posts to comment on what is written.

LSA – Latent Semantic Analysis: A form of analysis done on a large amount of text, which is used to judge how similar two passages of text are. SVD is applied to a corpus to create a semantic space, which is used to find similarity values by calculating cosines.

Post – A single text comment written in a forum. Posts often range in size, from essay length to single sentences. Grammar is often neglected in informal posts.

Semantic Space – A matrix generated from a corpus using SVD, with the rows and columns corresponding to the words and passages found in the corpus. The values found in the matrix are used to find the similarity between two passages or words.

Signature – A common technique of an author writing his or her name at the end of a post. A basic signature contains the author's name. They are often extended to include favorite quotes, anecdotes, or items of interest to the forum's users.

Similarity Value – A value between –1 and 1 that signifies how semantically similar two passages, or words are. A value of 1 represents a perfect match, with 0 representing no shared semantic value, and –1 represents opposite meaning. Finding the cosine between vectors in a semantic space generates the similarity value.

SVD – Singular Value Decomposition: A form of factor analysis used by LSA to reduce the dimensionality of a passage by word matrix, in an attempt to identify underlying semantic meaning.

Thread – A collection of posts related to each other that is found in a forum. A thread is normally started with an initial post offering an opinion or a question. Additional posts are added as people discuss what is being written in the thread.

ACKNOWLEDGEMENTS

A special thanks to Ben Hulscher for his work editing this paper, and every other paper I've written throughout college. Matt Paul gave exceptional advice, and help on the mathematics found in this paper. Thanks to all of my professors who have provided guidance, and tolerated my academic quirks, especially Donald Chinn, George Mobus, Josh Tenenberg, and Ralph Hitz.

DEDICATION

To my parents.

INTRODUCTION

As online forums grow in size they become intimidating to approach and challenging to work with. Threads containing over 100 posts can take more time to read than the information is worth. This paper presents research showing latent semantic analysis has the potential to be used as the base of an application to cluster posts by theme, and present a handful of posts that represent the themes discussed in a thread.

Ideally such a feature on an online forum will act as a summary tool that can be checked before attempting to read a thread. It will allow users to quickly identify threads worth reading, and gain a general feel for the type of posts being written. This will likely increase participation, as the scale of the online forum becomes easier to work with. It may also improve the content of the online forum, if users can easily identify which thread they should be reading, and contributing to.

Chapter 2 of this paper gives an overview of the background information that will be useful in understanding the research. Online forums, latent semantic analysis, and clustering are covered. The problem statement in Chapter 3 covers the original intent and outlook of the research. Chapter 4 covers the methodology of the research, while Chapter 5 presents the results along with analysis on what the results mean. Chapter 6 gives advice on how to implement the ideas presented in this thesis, and which situations produce the best results.

Chapter 1

BACKGROUND INFORMATION

To help in understanding the research presented in this thesis, a brief overview of online forums, latent semantic analysis, and clustering is presented here.

1.1 Online Forums

Online forums (simply forum from here on) are web applications used by groups for holding discussions. Forums are often called message boards, discussion groups, and bulletin boards [17]. The common implementation of a forum is as a message board where members can create topics, and write comments related to the topic. Comments, which average a large paragraph in size, are called posts. A thread is all of the posts listed under a specific topic.

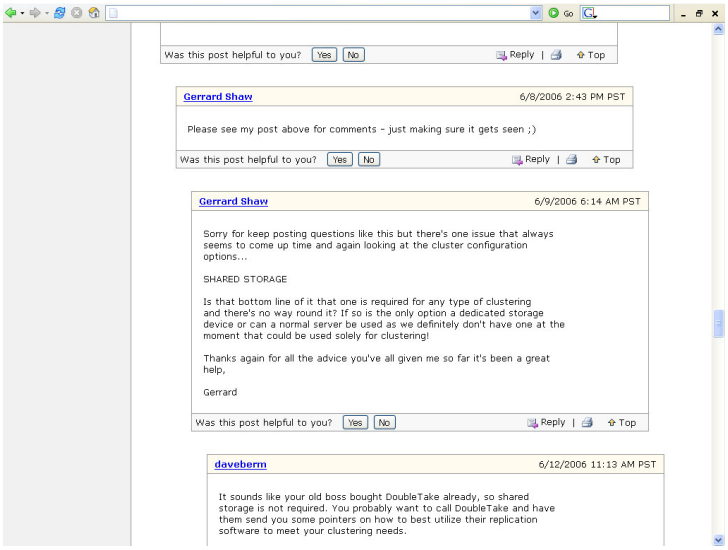

Figure 1.1 Screenshot of a forum.

Posts range in size from essay length to single sentences. Quotes from previous posts can be included in a post in order to address what was written and advance the discussion. Posts can have a signature, which is a method of the author writing his or her name at the end of a post that can be extended to include favorite quotes, anecdotes, or items of interest to the forum's users. Grammar is often neglected in informal posts, and shorthand is commonly used.

Often a forum has moderators that have the ability to delete comments, among other powers. Forums are currently in use by recreational groups using cheap or free systems (EZBoard), by online classes as a method of discussing course work (Blackboard), by businesses (Sony and Microsoft) for customer service, and many others.

An important concept of forums is that comments can be posted at any time, thus allowing members to take part in discussions at their convenience. This is a necessary feature, considering that forums can have thousands of members spread out between multiple time zones. Unfortunately as groups expand, information can quickly be lost in a surplus of posts, especially if members abuse the ability to post comments as often as desired.

1.2 Latent Semantic Analysis

Latent Semantic Analysis (LSA) "is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text [7]." In other words; LSA does an analysis of a large amount of text and uses the results to identify how similar the meaning is between two pieces of text.

The steps involved with LSA are creating a matrix of words by passage, pre-process the matrix, applying singular value decomposition (SVD), and finally using cosine to compare two vectors. The pre-processing applies a word filter, and does term weighting, which adjusts the value of a word based on how often it is used. Singular value decomposition allows for the reduction of the dimensionality, and the creation of a semantic space where every term and passage has its own vector. Comparing the angle of two vectors in semantic space with cosine effectively captures correlations, and is used as a similarity value.

Another way to look at it is that the meaning of a passage is equal to the average meaning of every word found in it, and the meaning of a word is equal to the average meaning of every passage it is found in. It is important to realize that LSA is capable of correctly inferring deep relations, and is not based on co-occurrence counts or keywords. Unlike algorithms implemented using co-occurrence counts, LSA can effectively handle synonyms. A disadvantage of LSA is that current implementations ignore word order, and consequently have no knowledge of syntactic relations, or logic.

Research on LSA has effectively demonstrated its power. LSA has been used to accomplish the following tasks: Grade essays comparable to human graders [6]; identify the various portions of text in a complex maintenance manual [15]; filter e-mail and sort them into an appropriate set of folders [15]; and find correlations while working with complex problem solving simulations [12].

Work with LSA and forums have resulted in the commercial product *Knowledge Post*. The unique features of *Knowledge Post* are using LSA to allow users to search for comments that are semantically similar to others, and have their own replies evaluated prior to posting [11]. The search is used to suggest the next best note to read (one that is not too hard and not too easy), find semantically

similar notes, and perform power searches of an electronic library for semantically similar topics [9]. The evaluation function picks comments in a similar method as LSA essay grading programs [11].

Simon Dennis has led research attempting to solve the inability of LSA to handle word order with the Syntagmatic Paradigmatic model. The Syntagmatic Paradigmatic model is a memory-based mechanism that incorporates word order but preserves the distributional approach of LSA. The idea is that when trying to understand a sentence you get similar sentences from memory and use String Edit Theory to align them and gather meaning. The Syntagmatic Paradigmatic model requires no initial input as far as particular grammars, heuristics, or sets of semantic roles, and it does not require an annotated corpus to train on [3].

Walter Kintsch has conducted research on how to expand LSA's scope to include predicates. His predicate algorithm showed potential when dealing with metaphor interpretation, causal inference, similarity judgments, and homonym disambiguation [5]. Unfortunately the algorithm currently lacks the ability to be implemented without human assistance for distinguishing the predicate and argument of a sentence.

Summarization tools that could be used to identify themes are being worked on, but still have a long way to go before completion. SUMMARIST is an attempt to build an automated text summarization system. Currently extracts can be created, which are specific portions of text or keywords produced verbatim. The goal of the project is to create human level abstracts [4].

1.3 Clustering

Clustering is the partitioning of a set of objects into subsets (clusters) that contain objects that are similar [13]. Each subset should contain objects that are similar to other objects in the cluster, and dissimilar to the objects found in other subsets. Similarity is normally defined by a distance measure. The goal of clustering is to find the natural grouping in a set of unlabeled data.

One method of clustering is hierarchical clustering. Hierarchical clustering starts by considering each object in your set to be its own cluster. Then one at a time it combines clusters until all objects are part of one large cluster (See figure 1.2). There is no point in having all of the objects in one cluster; however, you are able to backtrack and find the last k clusters to be merged, and use those as your clusters, where k is the number of clusters you desire. The algorithm selects the next clusters to be combined by finding the two clusters that are most similar. Using complete-linkage clustering, the similarity between two clusters is equal to the least similar objects belonging to each cluster [13].

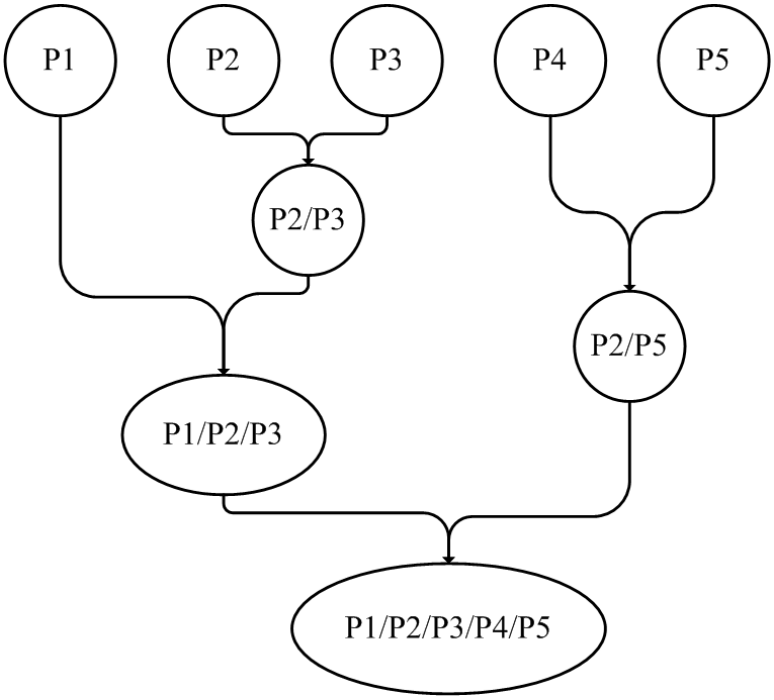

Figure 1.2 Hierarchical Clustering Diagram.

Chapter 2

PROBLEM STATEMENT

Large forums can easily have more comments posted than it is possible to read. This unfortunate aspect of forums leads to poor scalability. Successfully identifying the major themes of a thread, along with extracting a post that summarizes the theme, would allow for forum readers to quickly see what is being discussed under a thread. Consequently it would be possible for a reader to cover more ground, and identify areas of interest in a significantly shorter period of time. Thus LSA could be used to encourage the use of forums at a scale currently viewed as unfeasible.

The idea behind identifying themes is that when dealing with a thread that contains over 100 posts you expect to find many posts that share a common theme. Posts share a common theme if they have the same general meaning. If themes can be identified, then a post that is part of a theme can be chosen to act as a summary for all posts that are contained in the theme. For example, a thread that discusses the validity of a research project may have its posts identified according to which part of the methodology is being commented on.

My hypothesis is that LSA can be used to identify and extract themes in a forum setting with a large scale. The scale of the forum is important for two key reasons. It becomes redundant to identify and extract themes in a forum that receives a small number of posts, or moves at a slow pace, because themes should already be easy to identify by readers. Additionally a large amount of text will likely be needed for LSA to successful identify themes.

Chapter 3

METHODOLOGY

The work done on the LSA research presented in this thesis can be broken up into six groups. Forum selection involves picking text sources to create the corpora, and threads to analyze. Pre-processing is conducted on each corpus in an effort to generate a strong semantic space. Semantic space creation is done using singular value decomposition. The similarity between two passages of text is found by looking at the cosine between the passages' vectors in semantic space. Clustering is applied on the posts of a thread to identify themes found in the thread. Benchmarks for the LSA results are created by having individuals that are familiar with the forum separate the posts by theme.

3.1 Forum Selection

The threads used for this research were selected from the *Everquest 2* (http://eqiiforums.station.sony.com/eq2), and *Rotten Tomatoes* (http://www.rottentomatoes.com/) forums. The *Everquest 2* forum is the official forum for players of the video game *Everquest 2*, and the *Rotten Tomatoes* forum is part of a movie review website. These two forums were selected because of their large size, variety of topics, the ease of creating a related corpus (a series of text passages, often numbering in the thousands), and my familiarity with the subject matter.

The *Everquest 2* forum is filled with topics that attempt to solve a problem, including discussions on how to solve in-game problems, what the best strategies are, and what effect design changes have on the game. Popular topics often receive between 100 and 300 comments, and during prime time over 20

topics a minute can receive comments. A suitable corpus of text to train the LSA application on is available online in the form of a manual, player guides, and posts from the forum itself. The selection of corpus material will be done with a focus on including all of the game specific terms.

The *Rotten Tomatoes* forum offers the chance to test how LSA works when dealing with threads where the topic is the poster's opinion of a movie. This acts as an interesting test, because with LSA's inability to identify syntactic relations, it should struggle with opinionated reviews that disagree on the same subject. Posts range from one-sentence thoughts ("worst movie ever") to thousand-word reviews dissecting a movie. Movie reviews written by professional reviewers can be used to generate a corpus. A large enough set of professional reviews should encompass all of the themes found on a review thread.

3.2 Pre-Processing

A corpus is selected to create the LSA word by passage matrix. The average passage size will be a paragraph as recommended by multiple sources [5, 6, 9, 11]. A text parser (see Appendix D for code) reads each passage and identifies a word as being two or more letters without a non-letter character or space between letters. All letters are switched to lower case, to allow for matching the same word written with different capitalization. This method of word identification does not attempt to handle misspelled words, or words containing hyphens, apostrophes, or other non-letter characters. As it handles all passages with the same method, the identification of a word should have little to no effect on the results. While there are other algorithms available for creating a cleaner word list, I was unable to find a good argument for their use in this implementation considering the time needed to implement and run them.

A word filter is used to remove common words with minimum semantic value. The word filter is made up of the most common words found while selecting passages (see Appendix A for word list). The identified words of each passage are used to create a word by passage matrix for the entire corpus, which lists each passage and how many times each word is found in it.

As an example of the pre-processing needed for LSA, the passages found in Figure 3.1 were created to act as a corpus. Actual passages tend to be a paragraph in size. Figure 3.2 shows the passages after they have been parsed. Notice that all letters have been made lowercase, single letter words like "a" have been removed, and "can't" is read as "can" and "t", which resulted in only "can" being kept. The word filter has removed the word "the" from each passage, because it holds little semantic value.

P1: The dog ran like a dog.
P2: The cat chased the dog.
P3: Matt can't outrun the cheetah.
P4: A cat ran past Matt.

Figure 3.1 An example corpus.

P1: dog ran like dog
P2: cat chased dog
P3: matt can outrun cheetah
P4: cat ran past matt

Figure 3.2 Modified corpus: The corpus (Figure 3.1) after text parser, and word filter.

Term weighting is done after the creation of the word by passage matrix, as suggested by the LSA research literature [7]. The tf-idf (term frequency–inverse document frequency) formulation (Figure 3.3) is used to compute

weightings for words, in order to modify the word count by the perceived importance of that word [16]. Words that appear in many passages (for example "the") are considered unimportant and their word count is lowered. Rare words (for example "Seattle") have their word count raised because they are likely to have a high semantic value.

$$w_{ij} = tf_{ij} * \log_2(N/n) \text{ where,}$$

w_{ij} = word count of Word W_j in Passage P_i
tf_{ij} = frequency of Word W_j in Passage P_i
N = number of Passages in Corpus
n = number of Passages where Word W_j occurs at least once

Figure 3.3: Tf-idf equation [14].

Table 3.1 shows a word by passage matrix, as generated from the example corpus. Cell entries are the number of times that a word appeared in a passage. The large number of zeroes demonstrates why using sparse matrices is practically a requirement for implementing LSA. Table 3.2 is the same matrix after tf-idf has been applied. The values of words that appear once are now greater than words that appear twice.

Table 3.1 Word by passage matrix **{X}:** Cell entries are the number of times that a word appeared in a passage.

	P1	P2	P3	P4
dog	2	1	0	0
ran	1	0	0	1
like	1	0	0	0
cat	0	1	0	1
chased	0	1	0	0
matt	0	0	1	1
can	0	0	1	0
outrun	0	0	1	0
cheetah	0	0	1	0
past	0	0	0	1

Table 3.2 Tf-idf modified **{X}:** Word by passage matrix (Table 3.1), after tf-idf is applied.

	P1	P2	P3	P4
dog	2	1	0	0
ran	1	0	0	1
like	2	0	0	0
cat	0	1	0	1
chased	0	2	0	0
matt	0	0	1	1
can	0	0	2	0
outrun	0	0	2	0
cheetah	0	0	2	0
past	0	0	0	2

3.3 Semantic Space Creation

The semantic space is the word by passage matrix created by using SVD to reduce the dimensionality of the original word by passage matrix created from the corpus. SVD decomposes the original matrix X, into the product of three new

matrices, W, S, and P. S is a diagonal matrix containing the singular values. The singular values of X are the square roots of the eigenvalues of X^TX, arranged in decreasing order of magnitude [10].

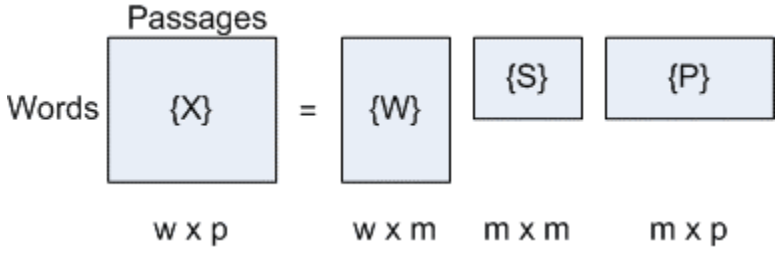

Figure 3.4 SVD matrices: With SVD a word by passage matrix X is decomposed into the product of three other matrices W, S, and P with **{X} = {W}{S}{P}**.

SVDPACKC (http://www.netlib.org/svdpack/) is used to do the SVD computations. SVDPACKC has been used multiple times during LSA research [8], and is the standard for doing SVD research. The single-vector Lanczos algorithm (las2) is used, because of its comparatively low memory and processing requirements [8]. The original matrix X is turned into the sparse matrix *Harwell-Boeing* storage format that las2 takes as input. The settings used when running las2 are based on recommendations from SVDPACKC user's guide [2]. The matrices W, S, and P are created by las2, which has been modified to output in text instead of binary.

To reduce *k* dimensions, the *n* least significant singular values (the smallest values), the last *n* columns of **{W}**, and the last *n* rows of **{P}** are removed (see Appendix D for code). The amount of dimensionality reduction has a major effect on results and is an open issue in LSA literature [1]. Selection of

dimensions is largely a process of trial and error. Once the matrices are reduced, finding the product of the three matrices creates the semantic space.

Tables 3.3 – 3.5 are the results of inputting the modified corpus found in Table 3.2. Table 3.6 is the three matrices after removing a dimension. The singular triplets are output in numerical order by las2. The product of the 3 matrices (Table 3.6) produce the semantic space X' found in Table 3.7. Notice the similarity to the original corpus (Table 3.1). The changes in values ideally have semantic meaning. It can be viewed as the odds of a word appearing in each passage, when thinking only of the semantic intent of the original passage.

Table 3.3 Left singular values matrix **{W}:** Created from Table 3.2.

0.05	-0.65	0.30	-0.07
0.07	-0.36	-0.18	0.29
0.03	-0.51	0.33	0.33
0.06	-0.24	-0.39	-0.29
0.02	-0.27	-0.07	-0.81
0.32	-0.06	-0.30	0.10
0.54	0.08	0.11	-0.03
0.54	0.08	0.11	-0.03
0.54	0.08	0.11	-0.03
0.10	-0.20	-0.70	0.24

Table 3.4 Singular values matrix **{S}:** Created from Table 3.2.

3.63	0	0	0
0	3.24	0	0
0	0	2.54	0
0	0	0	2.21

Table 3.5 Right singular values matrix **{P}:** Created from Table 3.2.

0.06	0.04	0.98	0.18
-0.82	-0.44	0.13	-0.33
0.42	-0.09	0.14	-0.89
0.37	-0.89	-0.03	0.26

Table 3.6 Reduced **{W}**, **{S},** and **{P}**: Tables 3.3 – 3.5 after reducing dimensions.

0.05	-0.65	0.30
0.07	-0.36	-0.18
0.03	-0.51	0.33
0.06	-0.24	-0.39
0.02	-0.27	-0.07
0.32	-0.06	-0.30
0.54	0.08	0.11
0.54	0.08	0.11
0.54	0.08	0.11
0.10	-0.20	-0.70

3.63	0	0
0	3.24	0
0	0	2.54

0.06	0.04	0.98	0.18
-0.82	-0.44	0.13	-0.33
0.42	-0.09	0.14	-0.89

Table 3.7 Matrix **{X}'**: The semantic space is a product of the matrices in Table 3.6.

	P1	P2	P3	P4
dog	2.06	0.86	0.00	0.04
ran	0.77	0.56	0.02	0.84
like	1.73	0.66	0.02	-0.19
cat	0.23	0.44	-0.02	1.16
chased	0.66	0.41	-0.06	0.47
matt	-0.08	0.20	1.01	0.84
can	0.02	-0.06	2.00	0.02
outrun	0.02	-0.06	2.00	0.02
cheetah	0.02	-0.06	2.00	0.02
past	-0.19	0.47	0.02	1.86

3.4 Similarity Values

The semantic space generated can be used to find similarity between two passages. It does not matter if the passages were in the original corpus or not. To compare two passages you create a vector for each passage, and use the cosine between the vectors as a similarity value (see Appendix D for code). A passage's vector is found by adding the vectors for each word found in the passage. This means that if a word is missing from a semantic space, it will have no effect on the passage's semantic value. Consequently, it is important to have a broad corpus that includes your specific subject matter to get best results. Each passage vector will have the same length, which will be equal to the number of passages in the semantic space.

The cosine value between vectors is calculated with the algorithm found in Figure 3.5. It produces a value between –1 and 1. With 1 being a complete match and –1 being the opposite. In practice the values range from –0.3 to 1.0, with a value near 0.90 signifying a strong similarity.

```
Cosine( Vector A, Vector B)
        double ab = 0
        double a = 0
        double b = 0
        int i = 0
        while i < vector length
                ab = ab + (A[i] * B[i])
                a = a + (A[i] * A[i])
                b = b + (B[i] * B[i])
        double cos = ab / (sqrt(a) * sqrt(b))
return cos
```

Figure 3.5 Vector cosine: Algorithm for calculating cosine between two equal length vectors.

Using the semantic space created earlier (Table 3.7), an example can be shown with three new passages (Figure 3.6). Only the underlined words found in Figure 3.6 exist in the semantic space, and have an effect on the passage vectors (Table 3.8). The cosine between vectors is displayed in Table 3.9. As expected when a passage is compared to itself, it results in a 1.0. The high similarity results carry little weight due to the scale of the example.

P5: The dog ran past matt as it barked.
P6: My cat looks like a cheetah.
P7: I have a cat, dog, and frog.

Figure 3.6 Passage example: Three passages to be compared, using the Table 3.7 semantic space. Underlined words appear in the semantic space.

Table 3.8 Passage vectors: Vectors of the three passages (Figure 3.6) using the Table 3.7 semantic space.

P5	2.56	2.09	1.05	3.58
P6	1.98	1.04	2.00	0.99
P7	2.29	1.30	-0.02	1.20

Table 3.9 Similarity values: Similarity between three passages (Figure 3.6) generated by finding the cosine between passage vectors (Table 3.8).

	P5	P6	P7
P5	1.00	0.81	0.89
P6	0.81	1.00	0.77
P7	0.89	0.77	1.00

3.5 Clustering

In order to identify the themes of a forum thread, the posts are clustered. The distance measure used for clustering is the similarity between posts, which is generated by using LSA (see Table 3.9 for example of similarity values). The resulting clusters should signify which posts are semantically similar and dissimilar to each other, which should relate to the themes of the posts.

The clustering algorithm implemented is complete-linkage hierarchical (see Appendix D for code). Other methods implemented with poorer results were single-linkage, and *k*-means. Single-linkage is hierarchical clustering when the similarity between two clusters is equal to the most similar objects belonging to each cluster. Use of single-linkage clustering resulted in one large cluster dominating the mergers. *K*-means clustering involves generating *k* centroids (one for each final cluster) and assigning objects to the most similar centroid. The *k* centroids are then recalculated and objects are again assigned to a centroid, with the process repeating until the recalculated centroids remain the same. *K*-means clustering can produce good results, however, it behaves erratically as I was unable to find an effective method of picking the initial centroids.

The removal of outliers was required to make complete-linkage clustering effective. The nature of forums results in the occasional post that adds little to the general discussions. Selecting a similarity value standard and a value for the smallest cluster you expect starts the removal of outliers. A post is considered an outlier if it does not have a similarity value equal to or greater than the standard to at least as many other posts as the number of posts in the smallest cluster (Table 3.10). This step is done because these outliers are often found to be similar to at least one other post, resulting in them disrupting the clustering.

Table 3.10 Identifying outliers: If the similarity value standard equals 0.70, and minimum cluster size is 3, then Passage *a* should be identified as an outlier.

	P*a*	P*b*	P*c*	P*d*	P*e*	P*f*
P*a*	**1.00**	**0.85**	**0.34**	**0.21**	**0.44**	**-0.12**
P*b*	0.85	1.00	0.92	0.88	0.96	0.76
P*c*	0.34	0.92	1.00	0.99	0.92	0.97
P*d*	0.21	0.88	0.99	1.00	0.96	0.88
P*e*	0.44	0.96	0.92	0.96	1.00	0.95
P*f*	-0.12	0.76	0.97	0.88	0.95	1.00

Picking the correct number of clusters is done automatically, which results in less ideal results, but allows the application to be easily run. The step before you get a cluster containing half or more of the posts is used. Advancing the clusters past this step normally results in one super cluster next to a hand full of miniature clusters. You do not expect to see at least one large theme throughout a thread, so lowering the maximum cluster size below 50 percent of the number of posts results in multiple clusters sharing a theme. Once the clusters are identified, clusters containing one post are removed as outliers.

Once the clusters have been created, one post from each cluster is selected to use as a representative of the cluster. The representative cluster should effectively display the theme of the cluster, so that it can be used as a summary for users. Finding the mean and median similarity value for each post relative to the other posts in the cluster is used to select the representative. The post with the highest mean or median is used (Table 3.11 calculates mean). An additional approach is to look at the highest similarity values for a given post, and use the mean of those in an attempt to identify the core of a cluster.

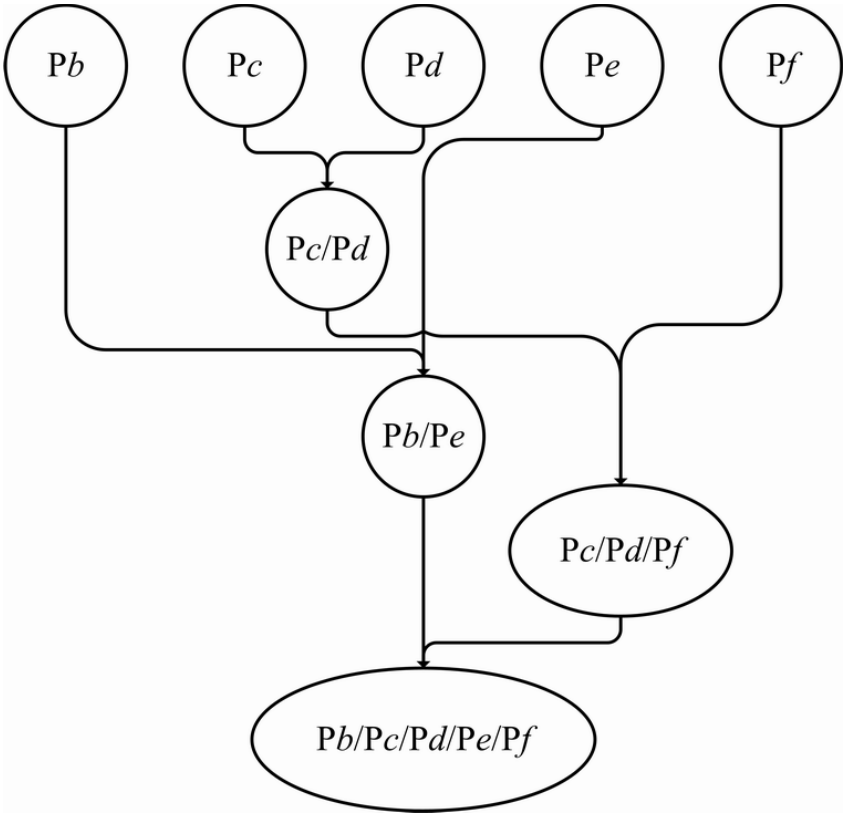

Figure 3.7 Clustering example: Clustering passages from Table 3.10, after outliers are removed.

Table 3.11 Similarity mean: The mean similarity values for the cluster of P*c*, P*d*, and P*f* from Table 3.10 and Figure 3.7. P*c* is the representative of the cluster.

P*c*	(0.99 + 0.97) / 2 = 0.98
P*d*	(0.99 + 0.88) / 2 = 0.94
P*f*	(0.97 + 0.88) / 2 = 0.93

3.6 Benchmark

To act as a benchmark a survey was conducted. Individuals were given a thread taken from a forum they were familiar with. They were asked to cluster the

posts in the thread, and pick posts that are strong representatives for each cluster and the thread.

As suggested by the University of Washington's Human Subject Division, *Catalyst WebQ* (http://catalyst.washington.edu/) was used to conduct the survey. Email invitations were sent out to the forum moderators and people with experience using the specific forums that I know. The specific questions can be found in Appendix B.

The benchmark cluster results are compared with the LSA cluster results by checking each pair of passages that share the same cluster, and seeing if both results agree that the pair belongs to the same cluster. If the pair belongs to the same cluster in both results it is considered a success. If the pair is assigned to the same cluster in one of the results, but the pair is split up to different clusters in the other result, it is considered a failure. If a passage is identified as an outlier by either result it is ignored.

Chapter 4

ANALYSIS

The analysis section looks in detail at the tests used on LSA. To create the semantic spaces, nine corpora were generated, and six threads were selected to cluster. Benchmarks were made for each of the threads. The results suggest that with the correct corpus a thread can effectively be clustered according to theme. Outlier identification gave mixed results, but appears to be worth using. The selection of the best post method performed poorly.

4.1 Corpora generated

The following corpora were generated to act as semantic spaces for the tests:

Reviews: Generated from professional reviews of the movie Jarhead. Reviews were found online via links provided by the website *Rotten Tomatoes*. Each review acts as a passage.

Guides: Generated from game guides taken from *EverQuest 2* fan web sites. Guides were selected to cover a wide range of topics about the game. Each guide acts as a passage.

General Forum: Generated from posts taken from the forum of the game *EverQuest 2*. Threads with a length between 5 and 50 posts were selected from multiple public boards. An emphasis was placed on selecting threads containing a wide variety of topics. Many of the posts use abbreviations, have poor grammar, and contain lengthy signatures.

Random Forum: Generated from posts taken from the forum of *EverQuest 2*. Threads were selected that covered non-gameplay issues, and had no relation to the specific subjects covered by the threads used for the tests. Many of the posts use abbreviations, have poor grammar, and contain lengthy signatures.

Gameplay: Generated from posts taken from the gameplay forum of *EverQuest 2*. Threads were selected that covered the main gameplay issues. Many of the posts use abbreviations, have poor grammar, and contain lengthy signatures.

Subset, and **Edited Subset**: Generated from posts taken from the forum of *EverQuest 2*. The posts were taken from a board specifically focused on an aspect of gameplay discussed in a thread used for theme analysis. Many of the posts use abbreviations, have poor grammar, and contain lengthy signatures. An edited version of this corpus was created, by removing posts with long signatures, and quotes, along with short one-sentence posts.

Sticky Posts: Generated from posts taken from the forum of *EverQuest 2*. The posts selected were those that started a sticky thread. A sticky thread is a thread that stays at the top of the list of threads found on a message board. Normally forum moderators create sticky threads by finding threads that are composed of quality posts and address issues commonly brought up. The initial posts were selected from sticky threads since they normally act as guides to a specific issue.

Gameplay/Guides: Generated by combining the Gameplay, and Guides corpora.

Table 4.1 Corpus data: The number of passages and unique words belonging to each corpus.

Corpus Name	Passages	Unique Words
Reviews	142	9,531
Guides	66	4,179
General Forum	2,968	16,889
Random Forum	330	2,388
Gameplay	245	3,079
Subset	1,433	8,138
Edited Subset	971	6,755
Sticky Posts	97	10,718
Gameplay/Guides	311	5,990

4.2 Threads Selected

The threads selected to test theme identification were picked based on the topic of the threads relative to the different corpora generated. The following threads were selected to test:

Gameplay One: A thread discussing a gameplay issue selected from the *Everquest 2* forum. It covers a topic likely to be missing from the Reviews, and Random Forum corpora. The thread is made up of 33 posts.

Gameplay Two: A thread discussing a gameplay issue selected from the *Everquest 2* forum. The goal of this thread is to reinforce the results found using the Gameplay One thread. It covers a topic likely to be missing from the Reviews, and Random Forum corpora. The thread is made up of 50 posts.

Subset Long, **Subset Edited**, and **Subset Unedited**: A thread selected from the *Everquest 2* forum. It was taken from the same board used to generate

the Subset corpus. The thread was selected mainly due to its size of 155 posts. It contains enough posts that the LSA results can act as a valuable tool for quickly summarizing the thread's content. The posts were edited with signatures being removed. The first 50 posts of the thread were used to generate the Subset Edited thread, in an attempt to see how results would change as a thread grows. The Subset Unedited thread was created using the unedited first 50 posts of the thread, to test the effect signatures have on the results.

Review Posts: A thread generated from a list of reviews posted by *Rotten Tomatoes* users. 50 reviews were selected, with the most recently written reviews being chosen. The reviews averaged a paragraph in length, and often contained grammar errors. The tone of the reviews is informal, and they are brief compared to the professional reviews used to generate the Review corpus.

4.3 Benchmarks

For each thread two to three benchmarks were created by conducting a survey. The survey asked individuals to cluster the posts according to theme, and pick posts that effectively represent the theme. The survey can be found in Appendix B.

The benchmark cluster results are compared with the LSA cluster results by finding the pairs of posts that are assigned to the same cluster in each result. If a post is identified as an outlier by either result, it is ignored. Table 4.2 shows the results when the benchmarks belonging to the same thread are compared. The general decline of averages loosely corresponds with the increase in thread size. This is not a trivial problem to solve. As the amount of posts looked at increases, it is reasonable to expect effort spent on identifying a good cluster will be

inversely related. The challenge and time needed to cluster so many posts is one of the key reasons LSA is being applied in attempt to automate the process.

Table 4.2 Benchmark similarity: Average pairs of passages that benchmarks of a thread agree belong to the same cluster.

Thread	Pairs agree
Gameplay One	34.2%
Gameplay Two	26.4%
Subset Long	20.1%
Subset Edited	16.9%
Subset Unedited	16.9%
Review Posts	29.5%

Two benchmarks were created for the Subset Long thread, with each person clustering the 155 posts. These two benchmarks were truncated (by removing all posts past the first 50, while leaving the remaining posts in the same clusters) to create benchmarks for the threads Subset Edited, and Subset Unedited. Because Subset Edited, and Subset Unedited use the same benchmarks, they have the same average in Table 4.2. The benchmarks have a higher average when looked at in full instead of truncated. The poorer performance of the shorter benchmarks is likely because the clusters were created with a thought towards what is found throughout the entire thread and not just the first 50 posts.

4.4 Cluster Results

The corpora were turned into semantic spaces, and used to cluster each of the threads. The cluster results are judged by finding the percentage of pairs of posts that are assigned to the same cluster in the result and benchmark. This measurement does an effective job of comparing how well each corpus did relative to each other for a specific thread. Unfortunately the number of clusters

can greatly affect results. With these tests, however, the damage is minimal due to the cluster algorithm, which normally produces a similar number of clusters.

The low percentages found in Table 4.3 are a result of the measurement and not the quality of the results. Table 4.2 shows how even when comparing clusters created by people the values are low. If you count all possible pairs (instead of only those found in the same cluster) the results can average over 70 percent. However that computation results in the number of clusters created having a material effect, which distorts the results.

Table 4.3 Cluster results: The percentage of pairs of posts that are assigned to the same cluster in the benchmarks and the given corpora. The best result is in bold, and the worst result is in italics.

Thread → Forum ↓	Gameplay One	Gameplay Two	Subset Long	Subset Edited	Subset Unedited	Review Posts
Reviews	**28.9%**	17.2%	16.3%	12.1%	17.6%	**32.8%**
Guides	19.7%	21.2%	13.2%	*11.8%*	15.9%	14.4%
General Forum	19.2%	**26.9%**	14.6%	14.8%	14.8%	15.2%
Random Forum	25.9%	*17.0%*	17.6%	**16.1%**	15.8%	18.3%
Gameplay	21.1%	23.6%	*13.4%*	13.9%	**18.5%**	21.3%
Subset	19.2%	25.2%	**18.8%**	12.7%	16.9%	20.0%
Edited Subset	*15.1%*	26.0%	17.2%	13.2%	15.5%	18.6%
Sticky Posts	18.9%	21.9%	15.6%	12.5%	15.5%	20.4%
Gameplay/Guides	20.5%	20.3%	18.1%	14.4%	*14.5%*	*12.7%*

The following are observations from looking at the cluster results:

Performance relative to human clustering: On average the clusters created using LSA are significantly worse than those created by individuals for benchmarks. The best result for a given thread, however, does an excellent job of clustering. When dealing with the best LSA generated results, it is difficult to

identify the computer-generated results from those done by a human. Table 4.4 demonstrates that when comparing the best result clusters to the benchmarks it is as similar to them as they are to each other.

Table 4.4 Results compared to benchmarks: The best corpus result from Table 4.3 compared to the difference among benchmarks found in Table 4.2

Gameplay One	Gameplay Two	Subset Long	Subset Edited	Subset Unedited	Review Posts
-5.3%	+0.5%	-1.3%	-0.8%	+1.6%	+7.3%

Corpus and thread subject: The Review Posts thread was the only thread not from the *EverQuest 2* forum. The Review corpus is the only corpus based on the same subject. It forms a stronger topic connection than any other thread corpus pair, because they both cover the same topic (reviews of the movie *Jarhead*). The resulting clustering percentages are exactly what were expected, with the Review corpus producing superior results than the corpora generated from *EverQuest 2* content. The Review corpus produced the best overall results relative to the benchmarks, suggesting that the stronger the relation between corpus and thread, the better the cluster results.

To a lesser extent the Subset corpus also supports the idea that you want the corpus and thread to cover a similar subject. While not dealing with the same specific topic of the three Subset threads, the Subset corpus was generated from posts taken from the same gameplay section of the *EverQuest 2* forum. As expected the best result for the Subset Long thread is the Subset corpus. Less expected is why the Subset corpus has a poor showing with the shorter Subset threads. The low results appear to be due to the poor quality of the benchmarks, which was caused when they were created by truncating the Subset Long benchmark. Another question is why the Subset Edited corpus received a low

result when clustering the Subset Long thread. This could be explained by pointing out that the Subset Edited corpus is a good deal smaller than the Subset corpus, and may have received too much editing.

Along the same lines it is expected that a corpus with even a little exposure to the topic discussed in a thread should do better than a corpus with no passages containing the topic. The Gameplay Two thread results are a picture perfect example of this. The Reviews, and Random Forum corpora do significantly worse than the other corpora, which were generated from posts that should at least in part cover the topics of the thread.

While the Gameplay Two thread results matches the theory of how LSA should perform, the Gameplay One thread shows how inconsistent the results can be. The opposite of what is expected happens in the Gameplay One test, as the corpora with no passages focused on the topic do the best job clustering. The Gameplay One test has two benchmarks and a closer look shows that the Reviews corpus received an interesting high rating, as it received 17.4% (4^{th} best result for benchmark) compared to one benchmark, and 40.4% (best result by far to benchmark) compared to the other benchmark. Looking at the cluster results in detail there is no obvious quirk that would cause this difference. The results just happened to match up extremely well to one of the benchmarks.

Corpus size: Due to the differences in content of the passages in each corpus, it is challenging to identify how big of an effect the size of the corpus has on the results. The smallest corpus generated, the Guides corpus, performed the worst when looking at the average cluster percentage for each corpus (Table 4.5). The second and third smallest corpora, Random Forum, and Gameplay, created good results. General Forum, the largest corpus, received middle of the pack scores.

Table 4.5 Cluster average: The average cluster percentage for each corpus. Based on Table 4.3.

Corpus Name	Cluster Rating
Reviews	20.8%
Guides	16.0%
General Forum	17.6%
Random Forum	18.5%
Gameplay	18.6%
Subset	18.8%
Edited Subset	17.6%
Sticky Posts	17.5%
Gameplay/Guides	16.8%

Curiously the combination of the Gameplay, and Guides corpora received results that normally fit between the two clusters. This suggests that simply expanding a corpus will not create a better semantic space, and can even create weaker results. The best explanation for this result may be that the Guides corpus performed poorly because of the scope of its passages, which contain many multi-paged guides, compared to the paragraph size passages that make up the Gameplay corpus, and the tested threads.

The most straightforward example of the effect of corpus size is between the Subset, and Edited Subset corpora. The Edited Subset corpus is an edited version of Subset corpus that was created, by removing posts with long signatures, and quotes, along with short one-sentence posts. This removed close to one third of the posts. Hypothetically this process could improve the semantic space by generating a corpus with more varied passages, however, in practice the Edited Subset corpus performed slightly worse than the Subset corpus. The reduction in size appears to have hurt the results more than the editing helped (assuming that the editing actually did help). The two corpora did create closer

results than any other pair of corpora. When comparing cluster results to each other they averaged 57.3%.

Effect of signatures: A signature is a common technique of an author writing his or her name at the end of a post. A basic signature contains the author's name, however, they are often extended to include favorite quotes, anecdotes, or items of interest to the forum's users. Almost every post of the Subset Unedited thread contains a signature, which were removed in the Subset Edited thread.

The Subset Unedited thread produced better results with 7 of the 9 corpora, and only in one case did worse. These improved results appear to exist because posts with long signatures from the same author are given higher similarity values. For example two posts with the same signature using the Gamplay corpus received similarity values of 0.64 without a signature, and 0.87 with. This creates a situation where LSA behaves as if it is guessing that posts with the same signature are likely to belong to the same cluster. Because posts by the same author normally stay with the same topic or opinion throughout a thread, giving the values a higher similarity value produces positive results.

4.5 Outlier Results

The removal of outliers is used in an attempt to diminish the number of small inconsequential clusters. The goal is to remove posts that add little to the general discussion.

Only four of the benchmarks identified outliers. One of the four is a benchmark for the Subset Long thread, so it can be truncated to work with the

Subset Edited, and Subset Unedited threads. The benchmarks ranged from 15 to 56 percent of the posts in a thread being identified as outliers.

Table 4.6 shows how many outliers each corpus identified along with how many of those agreed with the outliers selected by a given benchmark. The results show nothing conclusive, as many of there are spots where the outliers match up poorly, and others spots where they fit great. Two of the benchmarks were for the same thread (the ones with 22 and 28 outliers in Table 4.6), and they agreed 16 posts were outliers, while disagreeing on 18 posts. If you use that example as an ideal baseline, the outlier results appear to be far from perfect, but consistent enough to be worth using.

Table 4.6 Outlier results: Number of outliers identified by corpus that are also benchmark outliers for a thread.

Number of benchmark outliers	5	28	22	47	11	11
Reviews	2 of 9	5 of 9	6 of 9	13 of 15	8 of 18	3 of 4
Guides	1 of 2	2 of 2	2 of 2	0 of 0	1 of 4	2 of 2
General Forum	2 of 7	5 of 7	5 of 7	4 of 4	5 of 9	4 of 5
Random Forum	1 of 2	7 of 9	5 of 9	13 of 16	8 of 14	5 of 7
Gameplay	1 of 2	3 of 4	4 of 4	16 of 19	4 of 8	3 of 3
Subset	1 of 4	3 of 4	3 of 4	2 of 2	2 of 5	2 of 2
Edited Subset	1 of 4	1 of 1	1 of 1	2 of 7	1 of 4	1 of 1
Sticky Posts	0 of 0	2 of 2	0 of 2	1 of 1	1 of 1	1 of 1
Gameplay/Guides	1 of 1	1 of 1	1 of 1	5 of 6	2 of 5	3 of 3

4.6 Best Posts Results

Once the clusters have been created, one post from each cluster is selected to use as a representative of the cluster. The similarity values are used to calculate a mean, or median for each passage, and the highest result acts as the

representative. Another calculation is to find the mean while looking at only the top 5 similar passages.

Due to the challenge of creating a benchmark to judge how well a post represents a given cluster, the process was instead applied to the entire thread. Each corpus was used to create a similarity vector for each post of a thread, then the mean, median, and top 5 mean (mean calculated from the 5 highest similarity values) were calculated. The five posts with the largest value from each method were compared with the benchmark outliers. If a selected post is an outlier it is considered a poor choice to represent the thread, while non-outliers are acceptable.

The mean and median on average agreed on 3 out of the 5 posts picked. The posts both agreed on were just as likely to be an outlier as those they disagreed on. The top 5 mean normally chose five unique posts. Table 4.7 shows the percentage of outliers found among the representation posts selected from a given method. The last three columns of Table 4.7 demonstrate the ideal results. The other results identify outliers as if picking at random. The reason for the poor performance is due to "flaming," and quotes.

Table 4.7 Outlier percentages: The percentage of posts that are outliers in a thread, and the percentage of outliers found among the representation posts selected from a given method.

Posts that are outliers:	15.2%	56.0%	44.0%	30.3%	22.0%	22.0%
Mean	11.1%	55.6%	28.9%	2.2%	0.0%	2.2%
Median	11.1%	66.7%	33.3%	2.2%	2.2%	4.4%
Top 5 Mean	17.8%	64.4%	33.3%	0.0%	2.2%	17.8%

A post is considered a “flame” if it is written with the intent of agitating those that read it. Often this can lead to a series of posts arguing without contributing anything to the general discussion. A human can easily identify these posts as outliers. LSA on the other hand finds a group of similar posts and identifies them as a cluster. If there is a large enough set of posts bickering then one of the posts is likely to be picked to represent the thread.

It is common for one post to quote another in a thread. Having the same text appear in both posts results in the two posts receiving a high similarity value. This works as an advantage for clustering, because if a post uses a quote it is highly probable that the two share the same theme. One use of quotes is to quote a post followed by an expression of agreement or disagreement. In this situation it is likely that the two posts will receive close to a perfect similarity value. In contrast, a human may identify it as an outlier, because the new post offers little value to the general discussion. A series of quotes from the same post can result in multiple posts having a couple of perfect similarity values. This mechanism especially impacts the top 5 mean, as it almost exclusively identifies posts with quotes to represent a thread.

Chapter 5

DISCUSSION AND CONCLUSIONS

Based on the research it appears that LSA has the potential to be a useful tool when implemented in the correct environment. Used with a good forum it will make an effective summary tool for quickly getting a feel for a thread. This will in turn hopefully encourage more individuals to use the forum. In a forum without favorable properties LSA is unlikely to be consistent enough to act as anything more than a fun feature used mainly for curiosity's sake. An ideal implementation needs large threads, a good source for the corpus, and a strategy for handling signatures and quotes. Additionally, the method for picking the posts to display presented in this paper needs to be improved.

The ideal forum needs to have threads that consistently reach at least 100 posts. The main issue here is that the lower the post count the easier it is to simply look at the thread in whole without summarization. Threads with few posts are also less likely to contain multiple themes, which results in the attempt to cluster them being futile. When dealing with large threads and a big corpus it is important to remember that LSA can require a large amount of computation time. Based on the hardware available and the amount of traffic a forum receives, an implementation may require that updates to a thread's results be made after a certain time period instead of in real time.

A corpus built with passages of a similar subject to a thread will produce the best result. Ideally this means the forum's content is dedicated to a specific subject, so that a strong corpus can be generated for it. Larger forums may need multiple sections (the *Everquest 2* forum has over 50 boards), with a unique corpus for each. A forum filled with threads based on different topics is unlikely

to produce good results. It is possible to build a corpus directly from the threads of a forum. There should be plenty of posts to work with, assuming the forum is large enough to take advantage of LSA. Another advantage of generating the corpus from a forum is it produces new sources of passages that can be used to evolve the corpus.

The method used to deal with signatures and quotes will have a big effect on the results. Attempting to remove them may be infeasible. Even with guidelines, and tools for using signatures and quotes it is highly unlikely that users will comply in a way that makes it possible to automate the removal. Choosing to ignore the issue is the simplest method, however, it may improve results to embrace signatures and quotes. If the same poster normally stays with one theme on all posts in a thread, and a post that quotes another post likely shares the same theme, it may improve results to increase the similarity values of those posts by finding the relations through other means. An easy method would involve raising the similarity value of posts written under the same user name.

The method used for picking the best post to represent a thread needs to be improved. It is probably useful to always display the initial post of a thread, if only to offer insight into how the thread started. Doing a keyword search for inflammatory words might identify posts that should not be displayed. It is difficult figuring out how to deal with the selection of posts with a small comment under a quote from a more valuable post. It is dangerous to label a post with a quote less valuable, because some posts with quotes are superior, as the quotes can involve a set of questions that the new post answers. Perhaps it is best to leave things as they are, because users can read the quote and ignore the comment. It is worth mentioning that graphs and pictures are ignored. There is not much that can be done to identify the value of a picture, but it is important to realize that if

someone takes the time to import a picture some effort has likely been placed into the post.

LSA in a favorable forum setting should act as a powerful summarization tool. The next step in advancing this research involves implementing it in a large scale setting, and seeing what happens. Even if the results end up being poor, it will act as an interesting feature for a curious forum user.

BIBLIOGRAPHY

[1] M. Berry, “Large scale sparse singular value computations”, **The International Journal of Supercomputer Applications**, 6(1), 1992, pp. 13-49.

[2] M. Berry, et al., “SVDPACKC (Version 1.0) User's Guide”, **University of Tennessee Tech. Report CS-93-194**, 1993.

[3] S. Dennis, “An unsupervised method for the extraction of propositional information from text”, **Proceedings of the National Academy of Sciences**, 101, 2004, pp. 5206-5213.

[4] E. Hovy, and C. Lin, “Automated Text Summarization in SUMMARIST”, **Advances in Automated Text Summarization**, MIT Press, 1997.

[5] W. Kintsch, “Predication”, **Cognitive Science**, 25, 2001, pp. 173-202.

[6] T. K. Landauer, D. Laham, B. Rehder, and M. E. Schreiner, “How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans”, **Proceedings of the 19th annual meeting of the Cognitive Science Society**, 1997, pp. 412-417.

[7] T. K. Landauer, P. W. Foltx, and D. Laham, “Introduction to Latent Semantic Analysis”, **Discourse Processes**, 25, 1998, pp. 259-284.

[8] "Latent Semantic Analysis”, *InfoVis Cyberinfrastructure*, SLIS, Indiana University, 2004, retrieved February 14, 2007, from http://iv.slis.indiana.edu/sw/lsa.html.

[9] N. LaVoie, J. Psotka, K. E. Lochbaum, and C. Krupnick, “Automated Tools for Distance Learning”, **New Learning Technologies Conference**, February 18-20, 2004, Orlando, FL.

[10] D. C. Lay, **Linear Algebra And Its Applications: Second Edition**, Addison Wesley Longman, 1996.

[11] K. Lochbaum, J. Psotka, and L. Streeter, “Harnessing the Power of Peers”, **Interservice/Industry, Simulation and Education Conference**, December 2-5, 2002, Orlando, FL.

[12] J. F. Quesada, W. Kintsch, and E. Gomez, "A theory of Complex Problem Solving using Latent Semantic Analysis", **Proceedings of the 24th Annual Conference of the Cognitive Science Society**, 2002, pp. 750-755.

[13] H.C. Romesburg, **Cluster Analysis for Researchers**, Lightning Source Inc, 2004.

[14] G. Salton, **Automatic text processing: The Transformation Analysis and Retrieval of Information by Computer**, Addison-Wesley, 1989.

[15] M. J. Strait, J. A. Haynes, and P. W. Foltz, "Applications of Latent Semantic Analysis to Lessons Learned Systems", **Intelligent Lessons Learned Systems: Papers from the 2000 Workshop (Technical Report WS-00-03)**, 2000.

[16] S. M. Weiss, N. Indurkhya, T. Zhang, and F. J. Damerau, **Text Mining: Predictive Methods for Analyzing Unstructured Information**, Springer, 2005.

[17] Wikipedia contributors, "Internet forum" *Wikipedia, The Free Encyclopedia*, retrieved July 4, 2006, from http://en.wikipedia.org/w/index.php?title=Internet_forum&oldid=61992518.

Appendix A

WORD FILTER

This is the list of words filtered from a corpus when generating a semantic space.

a
all
am
an
and
another
any
are
b
be
both
but
c
could
d
did
do
does
e
each
either
every
f
g
h
had
has
have
i
is
j
k
l
m
many
may
might
more
most
much
n
neither
no
none
nor
not
o
of
or
other
p
q
r
s
several
shall
should
some
t
the
u
v
w
was
were
will
would
x
y
z

Appendix B

QUESTIONNAIRE

The following are the questions asked in the benchmark survey:

Question 1:
Describe each theme found in the thread with a brief sentence. (A theme can be defined as a talking point, or topic.) After each theme, list the post that best represents it. For example: "The movie's acting quality. Post 23."

Question 2:
List the posts according to which theme you think they belong to. For example: Theme 1: 1, 2, 5, 8, 9. Theme 2: 3, 10. Theme 3: 6, 11. Misc.: 4, 12.

Question 3:
List every post that you feel could act as a good summary for this thread.

Question 4:
Which post best represents the thread?

Appendix C

CONSENT FORM

The following is the consent form used for the survey found in Appendix B:

UNIVERSITY OF WASHINGTON
CONSENT FORM
ONLINE MESSAGE BOARD THREAD ANALYSIS

Researcher:
Mark Paul, graduate student, Computing and Software Systems, Institute of Technology, University of Washington, Tacoma, (253) 472-7260, kramluap@u.washington.edu

Researcher's Statement:

I am asking you to be in a research study. The purpose of this consent form is to give you the information you will need to help you decide whether to be in the study or not. Please read the form carefully. You may ask questions about the purpose of the research, what we would ask you to do, the possible risks and benefits, your rights as a volunteer, and anything else about the research or this form that is not clear. When all your questions have been answered, you can decide if you want to be in the study or not. This process is called 'informed consent.' We will give you a copy of this form for your records.

PURPOSE OF THE STUDY

I created a software application to do Latent Semantic Analysis (LSA) to identify themes of posts in an online message board thread. This application will demonstrate the potential of using LSA to help readers navigate large threads and message boards. Ideally LSA can be used to automatically display a small number of quality posts that represent what is being discussed. The results you produce from this survey will be used as a benchmark to compare with the results the LSA application gets when applied to the same threads.

STUDY PROCEDURE

You will be given a set of threads from your associated message board. For each thread you will be asked to do the following: Group the posts according to their themes. From each theme you will be asked to pick the post that best represents it.

List which posts you feel can act as a summary for the thread. Identify the best post for being used as a summary for the thread.

This should take in total between 20 and 30 minutes.

You may skip any section of the study.

RISK, STRESS, OR DISCOMFORT

Some people feel that participating in a study can result in a loss of privacy. Concerns over privacy are addressed in the Other Information section below.

BENEFITS OF THE STUDY

Hopefully the study will contribute to developing additional functionality for online message boards. You might not directly benefit from taking part in this study.

OTHER INFORMATION

Taking part in this study is voluntary. You can decide to not participate any point, and no one will know of your decision to participate or not. If you have any questions about this research study, please contact Mark Paul at the phone number or e-mail address above.

All data collected from the study will be kept confidential by storing them on secure computers, or in locked file cabinets. Any publication that results from the analysis of the data collected will contain no specific names, so as to preserve anonymity and confidentiality.

You will be given a code name, which will be used in all records. I will be the only person with access to the master list that links your real name and code name. The data and the master list will be kept separate. The master list will be stored in a locked cabinet, and will be destroyed after 6 months.

Printed name of Researcher Signature of researcher Date

Subject's statement:

This study has been explained to me. I volunteer to take part in this research. I have had a chance to ask questions. If I have questions later about the research, I can ask the researcher listed above. If I have questions about my rights as a

research subject, I can call the Human Subjects Division at (206) 543-0098. I will receive a copy of this consent form.

Appendix D

SOURCE CODE

This appendix includes the Java source code written for conducting the research. It includes the classes PassageHandler, Passage, LSA, LSAanalysis, and LSAstats. It does not include the code used from SVDPACKC.

PassageHandler.java

```
import java.text.DecimalFormat;
import java.util.*;
import java.io.*;

import jmp.SparseRowMatrix;
//http://www.mi.uib.no/~bjornoh/jmp/index2.html
import Jama.*; //http://math.nist.gov/javanumerics/jama/

/**
 * PassageHandler class takes a list of passages, and creates a
matrix(passage X word),
 * which is used for singular value decomposition, and then for
Latent Semantic Analysis.
 * Designed to work with SVDPACKC. http://www.netlib.org/svdpack/
 *
 * @author Mark Paul
 * kramluap@u.washington.edu
 * Influenced and including code written by Katy Börner
 * Last modified 1/22/07
 */
public class PassageHandler {
      private Vector passages; //list of passages read from input
      private Vector wordList; //list of words read from input
      private Vector filterWords; //list of words to be filtered
      private SparseRowMatrix matrixProb;
      private int nnzero;//number of non-zeros in matrix

      /**
       * Constructor
       */
      public PassageHandler() {}

      /**
       * Goes through passage list and calls print() for each
passage.
       */
      public void printPassages() {
```

```
            System.out.println("Read " + passages.size() + "
documents:");
            for (int i = 0; i < passages.size(); i++){
                  ((Passage)passages.get(i)).print();
            }
    }

      /**
       * Print to "passage.word": passage list, word list, and
word X passage matrix.
       */
      public void outputList(){
            try{
                  PrintWriter out = new PrintWriter( new
FileOutputStream("passage.word"));

                  out.println("Read " + passages.size() + "
passages:");
                  for (int i = 0; i < passages.size(); i++){
                        out.print(i + ": ");

      out.println(((Passage)passages.get(i)).getPassage());
                  }
                  out.println("");
                  out.println("Read " + wordList.size() + "
words:");
                  for (int i = 0; i < wordList.size(); i++){
                        out.print(i + ": ");
                        out.println((String)wordList.get(i));
                  }
                  out.println("");
                  for(int i=0; i < wordList.size(); i++ ){
                      for( int j=0; j < passages.size(); j++ ){
                        out.print( matrixProb.getValue(i,j) + " "
);
                      }
                      out.println("");
                  }
                  out.flush();
                  out.close();
                  System.out.println("Matrix has been written to
passage.word");
            }catch(Exception e){
                  System.out.println("error wrting to
passage.word");
            }
      }

    /**
     * Reads a file containing a list of passages and generates
the passage
```

```
     * and word lists. When reading input file, the algorithm
identifies
     * separate passages by a line break.
     * @param inputFile Name of file containing list of passages.
     */
    public void readPassageInput (String inputFile) {
      // generate vector of all words
        passages = new Vector();
      wordList = new Vector();
      setDefaultFilter();

      try{
            BufferedReader input = new BufferedReader(new
FileReader(new File(inputFile)));
            String s = input.readLine();
            int passageNumber =0;
            Passage pass;
            while (s!=null) {
                  //read new passage
                  passageNumber++;
                  pass = new Passage(passageNumber, s);
                  s = s.toLowerCase(); //do not care about case
                  String word = "";
                  //find each "word" in passage and add it to
word list.
                for (int i = 0; i < s.length(); i++) {
                        //check to see if word is found by
looking at one letter at a time
                  int letter = s.charAt(i);
                        if(letter >= 97 && letter <= 122 ||
letter >= 65 && letter <= 90){
                              //letter suggests a word, concact
it.
                              word =
word.concat(Character.toString((char)letter));
                        }else if (!word.equals("")){
                              //add or filter word
                              if(filterWords.contains(word)){
                                    //apply word filter, do
nothing
                              }else if(wordList.contains(word)){
                                    //repeat word, add it to
passage
                                    pass.addWord(new
Integer(wordList.indexOf(word)));
                              }else{
                                    //new word, add it to passage
and wordList
                                    wordList.add(word);
                                    //System.out.println(word);
                                    pass.addWord(new
Integer(wordList.indexOf(word)));
```

```
                              }
                              word = "";
                        }
                        //check for last word
                              if((i == s.length() - 1) &&
!word.equals("")){

      if(filterWords.contains(word)){
                                    }else
if(wordList.contains(word)){
                                          pass.addWord(new
Integer(wordList.indexOf(word)));
                                    }else{
                                          wordList.add(word);
                                          pass.addWord(new
Integer(wordList.indexOf(word)));
                                    }
                              }
                }
                pass.finalVector();
                  passages.add(pass);
                  s = input.readLine();
            }
            input.close();
        }catch(Exception e){
            System.out.println("IO error");
                  System.exit(1);
            }
        System.out.println("Found " + wordList.size() + " words,
and " + passages.size() + " passages.");
        //save memory
        wordList.trimToSize();
        passages.trimToSize();
      }


      /**
       * Creates passage X word matrix.
       */
      public void createSVDmatrix(){
            // matrix [word rows] [passages columns]
            matrixProb = new SparseRowMatrix(wordList.size(),
passages.size());
            nnzero = 0;
            System.out.println("Creating " + wordList.size() + "
word frequency X " + passages.size() + " passage matrix." );
            Passage currentDoc;
            //look at each word
            for (int i = 0; i < passages.size(); i++) {
                  currentDoc = (Passage)passages.get(i);
                  Vector words = currentDoc.getWords();
                  //Go through word vector of each passage
```

```
                for (int j = 0; j < words.size(); j++){
                      //add to matrix
                      int wordLoc =
((Integer)words.get(j)).intValue();
                      matrixProb.addValue(wordLoc, i, 1);
                      //check to see if new non-zero is found.
                      if(matrixProb.getValue(wordLoc, i) == 1){
                            //new non zero
                            nnzero++;
                      }
                }
                //currentDoc.print();
          }
     }

     /**
      * Implements tf-idf to add weights to word count according
to
      * importance of word. Frequent words have their counts
lowered,
      * while rare words have their counts raised.
      */
     public void tfidf(){
          //calculate for each word how many passages it
appears in.
          int[] passFreq = new
int[matrixProb.numRows()];//passage frequency for each word
          for(int i = 0; i < matrixProb.numRows(); i++){
                for(int j = 0; j < matrixProb.numColumns();
j++){
                      if(matrixProb.getValue(i, j) != 0){
                            //word appears in this passage
                            passFreq[i]++;
                      }
                }
          }

          //tf-idf is: word count * log2(number of passages /
number of passages word appears in)
          for(int i = 0; i < matrixProb.numRows(); i++){
                for(int j = 0; j < matrixProb.numColumns();
j++){
                      double value = matrixProb.getValue(i, j);
                      if(value != 0){
                            //Math.log finds ln.
                            double log2 =
Math.log(matrixProb.numColumns() / passFreq[i]) / Math.log(2);
                            matrixProb.setValue(i, j, (value *
log2));
                      }
                }
          }
```

```
            System.out.println("tf-idf applied");
      }

      public void printlap2(String description){
            try{
                  PrintWriter out = new PrintWriter(new
FileOutputStream("lap2"));
                  int maxprs = Math.round(passages.size() /
3);//the number of singular triplets of A
                  out.print("'" + description + "' " +
passages.size() + " " + maxprs + " -1.0e-30 1.0e-30 TRUE 1.0e-6
0");
                  out.flush();
                  out.close();
            }catch (Exception e ){
                  System.err.println( e.getMessage() + "\nThe
stack trace is:" );
                  e.printStackTrace();
            }
      }

      /**
       * Writes passage X word matrix to output file in Harwell-
Boeing format.
       * Method written by Katy Börner. Modified by Mark Paul to
handle doubles 5/13/06.
       * @param outputFile Name of file to write output.
       */
      public void printHBformat(String outputFile){
            try {
                  //Convert matrix to Harwell-Boeing format.
                  PrintWriter out = new PrintWriter(new
FileOutputStream(outputFile));
                  out.println();

                  out.println("Writing SVD data file");
                  out.println("Science Citation Data
" + outputFile);
                  out.println("Transposed");
                  out.println("rra                    " +
wordList.size() + "            " +
                  passages.size() + "           " + nnzero + "
0");
                  out.println("          (10i8)          (10i8)
(8f10.3)            (8f10.3)");

                  //3 vectors used to keep track of matrix,
value[x], rowind[x], relate to each other
                  Vector value = new Vector(); //numerical
values: list of non-zero values from matrix
                  Vector rowind = new Vector(); //row indices:
list of order that words appear through passage scan
```

```
                Vector pointr = new Vector(); //pointers:
stores the locations in the value array that start a column
                boolean firstValue;

                // transforming data into compressed column
storage format
                for (int j = 0; j < passages.size(); j++) {
                      firstValue = true;
                      for (int i = 0; i < wordList.size(); i++)
{
                            //check each passage, word
                            double matrixValue =
matrixProb.getValue(i, j);
                            //svd needs row >= column, if more
passages switch for svd (not implemented)
                            if (matrixValue !=0) {
                                  //non-zero value
                                  value.addElement(new
Double(matrixValue));
                                  rowind.addElement(new
Integer(i+1));
                                  if (firstValue){
                                        //first non-zero value
for a passage.
                                        pointr.addElement(new
Integer(value.size()));
                                        firstValue=false;
                                  }
                            }
                      }
                }
                out.println();
                out.print("       ");
                //print out all passage pointers
                for (int i = 0; i < pointr.size(); i++) {
                      out.print(pointr.elementAt(i) + "     ");
                      if (i%10==9) {
                            //new line
                            out.println("");
                            out.print("       ");
                      }
                }

                //last pointer is number of non-zero values + 1
                out.print(nnzero+1);
                out.println();

                out.print("       ");
                //print out all row indices
                for (int i = 0; i < rowind.size(); i++) {
                      out.print(rowind.elementAt(i) + "     ");
                      if (i%10==9) {
```

```
                              //new line
                              out.println("");
                              out.print("       ");

                        }
                  }

                  out.println();

                  out.print("       ");
                  //print out all numerical values
                  DecimalFormat outMatrix = new
DecimalFormat("0.000");
                  for (int i = 0; i < value.size(); i++) {

      out.print(outMatrix.format(((Double)value.elementAt(i)).dou
bleValue()) + "     ");
                        if (i%10==9) {
                              //new line
                              out.println("");
                              out.print("       ");
                        }
                  }
                  out.close();

                  System.out.println("SVD data has been written
in \"" + outputFile + "\"");
            }catch (Exception e ){
                  System.err.println( e.getMessage() + "\nThe
stack trace is:" );
                  e.printStackTrace();
            }
      }

      /**
       * Prints list of words to a file.
       * Number of words is printed on first line.
       * @param fileName file to print word list
       */
      public void printWordList(String fileName){
            try {
                  PrintWriter out = new PrintWriter(new
FileOutputStream(fileName));
                  out.println(wordList.size());//number of words
as header

                  for (int i = 0; i < wordList.size(); i++) {
                        out.println((String)wordList.get(i));
                  }
                  out.flush();
                  out.close();
```

```
                System.out.println("Word list has been written
in \"" + fileName + "\"");
            }catch (Exception e ){
                System.err.println("error printing word list");
                e.printStackTrace();
            }
      }

      /**
       * Word filter used to ignore common words that have little
semantic value.
       */
      private void setDefaultFilter(){
            filterWords = new Vector(70);
            filterWords.add("a");
            filterWords.add("all");
            filterWords.add("am");
            filterWords.add("an");
            filterWords.add("and");
            filterWords.add("another");
            filterWords.add("any");
            filterWords.add("are");
            filterWords.add("b");
            filterWords.add("be");
            filterWords.add("both");
            filterWords.add("but");
            filterWords.add("c");
            filterWords.add("could");
            filterWords.add("d");
            filterWords.add("did");
            filterWords.add("do");
            filterWords.add("does");
            filterWords.add("e");
            filterWords.add("each");
            filterWords.add("either");
            filterWords.add("every");
            filterWords.add("f");
            filterWords.add("g");
            filterWords.add("h");
            filterWords.add("had");
            filterWords.add("has");
            filterWords.add("have");
            filterWords.add("i");
            filterWords.add("is");
            filterWords.add("j");
            filterWords.add("k");
            filterWords.add("l");
            filterWords.add("m");
            filterWords.add("many");
            filterWords.add("may");
            filterWords.add("might");
            filterWords.add("more");
```

```
            filterWords.add("most");
            filterWords.add("much");
            filterWords.add("n");
            filterWords.add("neither");
            filterWords.add("no");
            filterWords.add("none");
            filterWords.add("nor");
            filterWords.add("not");
            filterWords.add("o");
            filterWords.add("of");
            filterWords.add("or");
            filterWords.add("other");
            filterWords.add("p");
            filterWords.add("q");
            filterWords.add("r");
            filterWords.add("s");
            filterWords.add("several");
            filterWords.add("shall");
            filterWords.add("should");
            filterWords.add("some");
            filterWords.add("t");
            filterWords.add("the");
            filterWords.add("u");
            filterWords.add("v");
            filterWords.add("w");
            filterWords.add("was");
            filterWords.add("were");
            filterWords.add("will");
            filterWords.add("would");
            filterWords.add("x");
            filterWords.add("y");
            filterWords.add("z");
      }

      /**
       * Main method for quickly generating matrix. First
argument is name of
       * passage list input file, second argument is name of SVD
matrix output.
       * If no arguments are entered default input is
"passages.txt", output
       * is "matrix.new".
       *
       * run: java -mx2000m PassageHandler
       *
       * @param args Expects 2 file names: 1st is input, 2nd is
output.
       */
      public static void main(String args[]){
            String INPUTFILE;
            String OUTPUTFILE;
            if(args.length != 2){
```

```
                INPUTFILE = "passages.txt";      //passages.txt
                OUTPUTFILE = "matrix.new";
          }else{
                INPUTFILE = args[0];
                OUTPUTFILE = args[1];
          }
          PassageHandler ph = new PassageHandler();
          System.out.println("readPassageInput");
          ph.readPassageInput(INPUTFILE);
          System.out.println("createSVDmatrix");
          ph.createSVDmatrix();
          ph.tfidf();
          ph.printHBformat(OUTPUTFILE);
          ph.printlap2(INPUTFILE);
          ph.printWordList("WordList");
          //ph.outputList();
          //ph.test();
    }

    /**
     * Test used while figuring out svdpackc
     */
    public void test(){
          Matrix m = new Matrix(matrixProb.numRows(),
matrixProb.numColumns());
          for (int j = 0; j < passages.size(); j++) {
                for (int i = 0; i < wordList.size(); i++) {
                      m.set(i,j,matrixProb.getValue(i, j));
                }
          }
          SingularValueDecomposition svd = new
SingularValueDecomposition(m);
          Matrix s = svd.getS();
          Matrix u = svd.getU();
          Matrix v = svd.getV();

          try{

                PrintWriter out = new PrintWriter(new
FileOutputStream("svdtest.test"));
                out.println("original");
                m.print(out,5,2);
                out.println("SV");
                s.print(out,5,2);
                out.println("left");
                u.print(out,5,2);
                out.println("right");
                v.print(out,5,2);
                Matrix us = u.times(s);
                Matrix usv = us.times(v);
                out.println("LSA");
                usv.print(out,5,2);
```

```
                out.flush();
                out.close();
            }catch(Exception e){
                System.out.println("error test");
            }
      }
}
```

Passage.java

```
import java.util.*;

/**
 * Passage class holds data of passages used for Latent Semantic
Analysis.
 *
 * @author Mark Paul
 * kramluap@u.washington.edu
 * Influenced by code written by Katy Börner
 * Last modified 5/13/06
 */
public class Passage {
      private String passage; //complete passage
    private int number; //number passage appeared
    private Vector words; //list of words used in LSA matrix

    /**
     * Constructor
     * @param number The number passage appeared in origional
document list.
     * @param passage The complete passage.
     */
    public Passage(int number, String passage) {
      this.passage  = passage;
      this.number = number;
      this.words = new Vector();
    }


    /**
     * Use after list of words is finished to trim vector and
save memory.
     */
    public void finalVector(){
      words.trimToSize();
    }

    /**
     * Add a word to vector containing list of words used in LSA
matrix.
```

```
     * Words to be added once for each time they appeared in
passage.
     * @param wordPosition
     */
    public void addWord(Integer wordPosition){
      words.add(wordPosition);
    }

    /**
     * Returns a vector containing a list of the words passage
has that
     * are used in the LSA matrix.
     * @return Vector containing list of words found in passage.
     */
    public Vector getWords(){
      return words;
    }

    /**
     * Complete passage as read from the input.
     * @return String complete passage.
     */
    public String getPassage(){
      return passage;
    }

    /**
     * Returns the number passage appeared in origional document
list.
     * @return int passage number
     */
    public int getNumber(){
      return number;
    }

    /**
     * Prints the number of passage, followed by the complete
passage.
     * Of the form: "number: passage"
     */
    public void print() {
      System.out.println(number + ": " + passage);
    }
}
```

LSA.java

```
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintWriter;
import java.util.StringTokenizer;
```

```
import java.text.*;


/**
 * LSA class takes results from running a modified las2 from
SVDPACKC and produces a
 * corpus for use with Latent Semantic Analysis.
 * Designed to work with SVDPACKC. http://www.netlib.org/svdpack/
 *
 * @author Mark Paul
 * kramluap@u.washington.edu
 * Influenced by code written by Katy Börner
 * Last modified 7/4/06
 */
public class LSA {

      private double[][] rsv; //right-singular vectors (passages)
      private double[][] lsv; //left-singular vectors (words)
      private double[] sv; //singular values
      private double[][] similarity; //similarity matrix
      private double[][] lsaMatrix; //word-passage matrix with
reduced dimensionality
      private int nsv; //number of single values
      private int nCol; //number of columns
      private int nRow; //number of rows

      /**
       * Returns number of singular values found in las2 output
file.
       * @return number of single values
       */
      public int getNSV(){
            return nsv;
      }

      /**
       * Read result output of SVD from data file.
       */
      private void readResult(){
            // get number of singular values
            nsv = 0;
            String line;
            String token = "";
            try{
                  BufferedReader reader = new BufferedReader(new
FileReader("lao2"));//SVDPACKC las2.c output file
                  //read column and singular value totals
                  while((line = reader.readLine()) != null){
                        StringTokenizer tokens = new
StringTokenizer(line, "= \t");
                        if(tokens.hasMoreTokens()){
                              token = tokens.nextToken();
```

```
                        }
                        //look for number of columns and rows
data
                        if(tokens.hasMoreTokens() &&
token.equals( "..." )){
                        if(tokens.hasMoreTokens() &&
tokens.nextToken().equals("NO.")){
                              tokens.nextToken(); //skip "OF"
                              token = tokens.nextToken();
                              if(token.equals("TERMS")){
                                    tokens.nextToken();//skip
"(ROWS)"
                                    //found number of rows
                                    nRow =
Integer.parseInt(tokens.nextToken());
                              }else
if(token.equals("DOCUMENTS")){
                                    tokens.nextToken();//skip
"(COLS)"
                                    //found number of columns
                                    nCol =
Integer.parseInt(tokens.nextToken());
                              }
                        }
                     //look for number of singular values
                     }else if(tokens.hasMoreTokens() &&
token.equals("......")){
                        if(tokens.hasMoreTokens() &&
tokens.nextToken().equals("NSIG")){
                              //found number of singular vaules
                              nsv =
Integer.parseInt(tokens.nextToken());
                              break;
                        }
                     }
                  }

                  if(nsv <= 0){
                        System.out.println("Invalid number of
singular values: " + nsv);
                        System.exit(-1);
                  }
                  //read singular value vector
                  line = reader.readLine(); // skip blank line
                  line = reader.readLine(); // skip line
"COMPUTED S-VALUES ..."
                  line = reader.readLine(); // skip blank line
                  sv = new double[nsv];
                  for(int i=0; i < nsv; i++){
                        line = reader.readLine();
                        StringTokenizer tokens = new
StringTokenizer( line, " \t" );
```

```
                        token = tokens.nextToken(); // skip line
header
                        token = tokens.nextToken(); // skip line
#
                        token = tokens.nextToken();
                        //singular values are reverse ordered in
lao2
                        //such that the most significant/highest
value occurs last
                        sv[nsv - i - 1] =
Double.parseDouble(token);//I think?
                  }
            }catch(Exception e){
                  e.printStackTrace();
                  System.out.println("error reading lao2");
            }

            try{
                  //read right and left singular vectors
                  BufferedReader read = new BufferedReader(new
FileReader("las2out"));//SVDPACKC las2.c custom output file
                  System.out.println("Columns = " + nCol);
                  System.out.println("Rows = " + nRow);
                  System.out.println("Number of Singular Values =
" + nsv);
                  int filesize = nRow*nsv + nsv*nCol; //values
stored in file

                  lsv = new double[nRow][nsv];//left singular
vectors
                  rsv = new double[nsv][nCol];//right singular
vectors

                  for(int i=0; i < nsv ; i++){
                      for(int j = 0; j < nCol; j++){//read rsv
values for a row
                        rsv[nsv - 1 - i][j] =
Double.parseDouble(read.readLine());
                      }
                      for(int j = 0; j < nRow; j++){//read lsv
values for a row
                        lsv[j][nsv - 1- i] =
Double.parseDouble(read.readLine());
                      }
                  }
            }catch(Exception e){
                  e.printStackTrace();
                  System.out.println("error reading las2out");
            }
      }
```

```
	/**
	 * Used to lower the dimensions of LSA matrix.
	 * @param dimensions number of dimensions, must be < number
of singular values.
	 */
	private void lowerDimensions(int dimensions){
		//might want to edit into readResults for better run
time and less memory.

		//edit singular values
		double[] svTemp = new double[dimensions];
		for(int i = 0; i < dimensions; i++){
			svTemp[i] = sv[i];
		}
		sv = svTemp;
		nsv = dimensions;
		//add clean up for temps?

		// edit left-singular vectors (words)
		double[][] lsvTemp = new double[nRow][dimensions];
		for(int i = 0; i < nRow; i++){
			for(int j = 0; j < dimensions; j++){
				lsvTemp[i][j] = lsv[i][j];
			}
		}
		lsv = lsvTemp;

		//edit right-singular vectors (passages)
		double[][] rsvTemp = new double[dimensions][nCol];
		for(int i = 0; i < dimensions; i++){
			for(int j = 0; j < nCol; j++){
				rsvTemp[i][j] = rsv[i][j];
			}
		}
		rsv = rsvTemp;

		System.out.println("Dimensions reduced to " +
dimensions);
	}

	/**
	 * Creates LSA corpus.
	 */
	private void createLSAMatrix(){
		System.out.println("creating LSA matrix...");
		//create sv matrix from singular value matrix
		double[][] svMatrix = new double[nsv][nsv];
		for(int i = 0; i < nsv; i++){
			svMatrix[i][i] = sv[i];
		}
		//System.out.println("matrix multiplication");
```

```
            double[][] lsvSV = matrixMult(lsv, svMatrix, nRow,
nsv, nsv);
            lsaMatrix = matrixMult(lsvSV, rsv, nRow, nsv, nCol);

            //print results to Corpus file
            try{
                  //create corpus file
                  PrintWriter outCorpus = new PrintWriter( new
FileOutputStream("Corpus"));
                  outCorpus.println(nRow);
                  outCorpus.println(nCol);
                  for(int i=0; i < nRow; i++ ){
                      for( int j=0; j < nCol; j++ ){
                        //outCorpus.println(lsaMatrix[i][j]);
                        outCorpus.print(lsaMatrix[i][j] + "\t "
);
                      }
                      outCorpus.println();
                  }
                  outCorpus.flush();
                  outCorpus.close();
            }catch(Exception e){
                  System.out.println("error printing corpus");
                  System.exit(-1);
            }
            System.out.println("Output printed to Corpus");
      }

      /**
       * Prints results to MatrixLSA
       */
      private void printOutput(){
            DecimalFormat outMatrix = new DecimalFormat("0.00");
            //System.out.println("printing output...");
            try{
                  PrintWriter out = new PrintWriter( new
FileOutputStream("MatrixLSA"));
                  out.println("singular values");
                  for(int i = 0; i < nsv; i++){
                        for(int j = 0; j < nsv; j++){
                              if(i == j){

      out.print(outMatrix.format(sv[i]) + "\t");
                              }else{
                                    out.print(outMatrix.format(0)
+ "\t");
                              }
                        }
                        out.println();
                  }
                  out.println();
                  out.println("left values");
```

```
                    for(int i=0; i < nRow; i++ ){
                        for( int j=0; j < nsv; j++ ){
                          out.print(outMatrix.format(lsv[i][j]) +
"\t" );
                        }
                        out.println();
                    }
                    out.println();
                    out.println("right values");
                    for(int i=0; i < nsv; i++ ){
                        for( int j=0; j < nCol; j++ ){
                          out.print(outMatrix.format(rsv[i][j]) +
"\t" );
                        }
                        out.println();
                    }
                    out.println("");

                    out.println("corpus");
                    for(int i=0; i < nRow; i++ ){
                        for( int j=0; j < nCol; j++ ){

      out.print(outMatrix.format(lsaMatrix[i][j]) + "\t" );
                        }
                        out.println();
                    }
                    out.flush();
                    out.close();
              }catch(Exception e){
                    e.printStackTrace();
                    System.out.println("error printing");
              }
              System.out.println("Output printed to MatrixLSA");
      }

      /**
       * Linear algebraic matrix multiplication, A * B
       * @param a left Matrix
       * @param b right Matrix
       * @param nr number of rows for left Matrix
       * @param inner number of columns for left Matrix (Same as
rows for right Matrix)
       * @param nc number of columns for right Matrix
       * @return New matrix resulting from multiplication.
       */
      public double[][] matrixMult(double[][] a, double[][] b,
int nr, int inner, int nc){
              double[][] newMatrix = new double[nr][nc];
              double[] Bcolj = new double[inner];
              for(int j = 0; j < nc; j++){
                    for(int k = 0; k < inner; k++){
                          Bcolj[k] = b[k][j];
```

```
                }
                for(int i = 0; i < nr; i++){
                      double[] Arowi = a[i];
                      double s = 0;
                      for (int k = 0; k < inner; k++){
                            s += Arowi[k]*Bcolj[k];
                      }
                      newMatrix[i][j] = s;
                }
          }
          return newMatrix;
    }


    /**
     * Main method. Run after using modified las2 from
SVDPACKC.
     * run: java -mx2000m LSA
     * @param args int value of number of dimensions wanted for
corpus generation
     */
    public static void main(String args[]){

          LSA lsa = new LSA();

          lsa.readResult();
          if(args.length == 1){
                int dimensions = Integer.parseInt(args[0]);
                if(dimensions < lsa.getNSV() && dimensions >
0){
                      lsa.lowerDimensions(dimensions);
                }
          }
          lsa.createLSAMatrix();
          lsa.printOutput();
          System.out.print("done");
    }
}
```

LSAanalysis.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.util.*;

/**
 * Class loads semantic space created with LSA.java,
```

```
 * and creates similarity matrix from list of passages.
 *
 * @author Mark Paul
 * kramluap@u.washington.edu
 * Last modified 2/6/07
 */
public class LSAanalysis {
      private Hashtable wordList; //hash string word and get int
equal to row number of word in corpus
      private String corpus;
      private Vector passages; //list of passages that will be
compared
      private Vector passageVectors; //passage vectors double[]
      private double[][] simMatrix;

      //constructor
      public LSAanalysis(){}

      /**
       * Loads semantic space corpus created from LSA.
       * @param corpusFile semantic space matrix created by LSA
       * @param wordFile list of words relating to rows of
corpusFile
       */
      private void loadCorpus(String corpusFile, String
wordFile){
            System.out.println("Loading Corpus");
            corpus = corpusFile;

            //create word list hash table
            try{
                  BufferedReader reader = new BufferedReader(new
FileReader(wordFile));
                  int nwords =
Integer.parseInt(reader.readLine());//number of words, taken from
header
                  int hashSize = Math.round(nwords *
1.5f);//based on suggested size ratio of .75
                  wordList = new Hashtable(hashSize);
                  for(int i = 0; i < nwords; i++){
                        wordList.put(reader.readLine(), new
Integer(i));
                  }
            }catch(Exception e){
                  System.out.println("error reading wordlist");
                  e.printStackTrace();
            }
      }

      /**
       * Creates similarity matrix.
```

```
       * Finds semantic space vector for each passage by adding
word vectors
       * found in corpus. Compares passages by getting the cosine
between
       * their vectors.
       */
      private void makeSimilarityMatrix(){
            System.out.println("Making Similarity Matrix");
            passageVectors = new Vector(passages.size());
            //create semantic space vector for each passage;
            for(int i = 0; i < passages.size(); i++){
                  Passage pass = (Passage)passages.get(i);
                  //get word list
                  Vector words = pass.getWords();//vector of
Integers
                  BufferedReader readPassage;
                  try{
                        readPassage = new BufferedReader(new
FileReader(corpus));
                        //header info
                        int nRow =
Integer.parseInt(readPassage.readLine());
                        int nCol =
Integer.parseInt(readPassage.readLine());
                        if(words.size() == 0){
                              //passage has no words that are in
corpus.
                              System.out.println("Passage "+ i+ "
has no words found in corpus.");
                              passageVectors.add(new double[0]);
                        }else{
                              int[] wordOrder = new
int[words.size()];
                              for(int j = 0; j < words.size();
j++){
                                    wordOrder[j] =
((Integer)words.get(j)).intValue();
                              }
                              Arrays.sort(wordOrder);

                              double[] passVector = new
double[nCol];
                              int wordPosition = 0;//position
worked through word list
                              for(int j = 0; j < nRow; j++){
                                    String line =
readPassage.readLine();//read row i from corpus
                                    //check to see if word is in
passage
                                    if(wordOrder[wordPosition] ==
j){
```

```
                                          StringTokenizer tokens
= new StringTokenizer(line, " \t");
                                          //StringTokenizer
tokens = new StringTokenizer(line);
                                          double[] rowVector =
new double[nCol];
                                          for(int k = 0; k <
nCol; k++){
                                                rowVector[k] =
Double.parseDouble(tokens.nextToken());
                                          }
                                          passVector =
addVectors(passVector, rowVector);
                                          wordPosition++;
                                          if(wordPosition ==
wordOrder.length) break;//prevent array out of bounds

      while(wordOrder[wordPosition] == wordOrder[wordPosition -
1]){
                                                //word appears
more than once in passage
                                                passVector =
addVectors(passVector, rowVector);
                                                wordPosition++;
                                                if(wordPosition
== wordOrder.length) break;
                                          }
                                          if(wordPosition ==
wordOrder.length) break;
                                    }

                              }
                              passageVectors.add(passVector);
                              readPassage.close();
                        }
                  }catch(Exception e){
                        System.out.println("Error creating
passage vectors.");
                        e.printStackTrace();
                        System.exit(-1);
                  }
            }

            //create similarity matrix from passage vectors
            simMatrix = new
double[passages.size()][passages.size()];
            for(int i = 0; i < passages.size(); i++){
                  for(int j = 0; j < passages.size(); j++){
                        if(i >= j){//go to next row, prevents
repeat calculations
```

```
                                    double cos =
vectorCOS((double[])passageVectors.get(i),(double[])passageVector
s.get(j));
                                    //same value appears twice in
matrix
                                    simMatrix[i][j] = cos;
                                    simMatrix[j][i] = cos;
                              }
                        }
                  }
            }

            /**
             * Add two vectors.
             * @param a First vector.
             * @param b Second vector.
             * @return New vector resulting from addition.
             */
            private double[] addVectors(double[] a, double[] b){
                  if(a.length != b.length) return null;
                  double[] ab = new double[a.length];
                  for(int i = 0; i < a.length; i++){
                        ab[i] = a[i] + b[i];
                  }
                  return ab;
            }

            /**
             * Read file with list of passages to be compared using
loaded corpus.
             * Passages are separated by line breaks.
             * @param inputFile File with passages.
             */
            private void readCompare(String inputFile){
                  System.out.println("Read passages.");
                  passages = new Vector();

            try{
                  BufferedReader input = new BufferedReader(new
FileReader(new File(inputFile)));
                  String s = input.readLine();
                  int passageNumber = 0;
                  Passage pass;
                  while (s!=null) {
                        //read new passage
                        pass = new Passage(passageNumber++, s);
                        s = s.toLowerCase(); //do not care about case
                        String word = "";
                        //find each "word" in passage and add it to
word list.
                      for (int i = 0; i < s.length(); i++) {
```

```
                        //check to see if word is found by
looking at one letter at a time
                  int letter = s.charAt(i);
                        if(letter >= 97 && letter <= 122 ||
letter >= 65 && letter <= 90){
                              //letter suggests a word, concact
it.
                              word =
word.concat(Character.toString((char)letter));
                        }else if (!word.equals("")){
                              //add word
                              /*Integer loc =
(Integer)wordList.get(word);
                             if(loc != null) {

                                    pass.addWord(loc);
                              } */
                             if(wordList.get(word) != null) {


      pass.addWord((Integer)wordList.get(word));
                              }
                              word = "";
                        }
                        //check for last word
                              if((i == s.length() - 1) &&
!word.equals("")){
                                    //add word
                                    if(wordList.get(word) !=
null) {

      pass.addWord((Integer)wordList.get(word));
                              }
                              }
                }
                //add finished passage to vector
                pass.finalVector();
                  passages.add(pass);
                  s = input.readLine();
            }
            input.close();
        }catch(Exception e){
            System.out.println("Error reading passages.");
            e.printStackTrace();
                  System.exit(-1);
            }
      }

      /**
       * Find cosine of two equal length vectors
       * @param A First vector.
       * @param B Second vector.
```

```
	 * @return cosine value
	 */
	private double vectorCOS(double[] A, double[] B){
		if(A.length == 0 || B.length == 0){
			//one of the vectors is from a passage with no
words in corpus.
			return 0;
		}
		double ab = 0;
		double a = 0;
		double b = 0;
		for(int i = 0; i < A.length; i++){
			ab = ab + (A[i] * B[i]);
			a = a + (A[i] * A[i]);
			b = b + (B[i] * B[i]);
		}
		if((Math.sqrt(a) * Math.sqrt(b)) == 0) return
0.0;//prevent divide by zero
		double cos = ab / (Math.sqrt(a) * Math.sqrt(b));
		return cos;
	}

	/**
	 * Prints similarity matrix and passage list to file.
	 * @param outFile name of file to print to
	 */
	private void printSimilarity(String outFile){
		System.out.println("Printing Similarity Matrix");
		try{
			PrintWriter out = new PrintWriter( new
FileOutputStream(outFile));
			DecimalFormat outMatrix = new
DecimalFormat("0.00");

			out.println("Similarity Matrix...");
			out.print("\t");
			for(int i = 0; i < passages.size(); i++){
				//print column numbers
				out.print(i + "   \t");
			}
			out.println();
			for(int i = 0; i < passages.size(); i++){
				out.print(i + "\t");//print row numbers
				for(int j = 0; j < passages.size(); j++){
					//print similarity matrix value

	out.print(outMatrix.format(simMatrix[i][j]) + "\t");
				}
				out.println();
			}
			out.println();
			//print list of passages
```

```
                  out.println("Passages...");
                  for(int i = 0; i < passages.size(); i++){
                        Passage pOut = (Passage)passages.get(i);
                        //passage number corresponds to row and
column numbers
                        out.println(pOut.getNumber() + " : " +
pOut.getPassage());
                  }

                  out.flush();
                  out.close();
                  System.out.println("Similarity matrix printed
to " + outFile);
            }catch(Exception e){
            System.out.println("Error writing similarity
matrix.");
            e.printStackTrace();
                  System.exit(-1);
            }
      }

      /**
       * Do analysis of similarity matrix using LSAstats class.
       */
      private void similarityAnalysis(){
            System.out.println("Similarity Anaysis");
            //LSAstats stats = new LSAstats(passages.size(),
simMatrix);
            LSAstats stats = new LSAstats(simMatrix);
            stats.findOutliers(0.7, 3);
            int[] clusters = stats.completeLinkage();
            printClusterArray(clusters);
            double[] print;
            print = stats.getAverages();
            System.out.println("averages");
            printPartArray(print, topX(print, 5));
            print = stats.getMedian();
            System.out.println("median");
            printPartArray(print, topX(print, 5));
            print = stats.getTopAverage(5);
            System.out.println("top 5 avg");
            printPartArray(print, topX(print, 5));
            //for(int i = 3; i <= 10; i++){

      //stats.clusterHierarchical(5, false);

            //printArray(clusters);
                  //printClusterArray(stats.completeLinkage());
                  //i = i + 10;
            //}
      }
```

```
	/**
	 * Prints out clusters in order.
	 * @param ca Cluster array created by cluster methods in
LSAstats
	 */
	private void printClusterArray (int[] ca){
		//array of passages, each value symbolizes what
cluster it belongs to. First cluster is 0.
		//find number of clusters;
		int numClusters = -1;
		System.out.println("Outliers:");
		for(int i = 0; i < ca.length; i++){
			if(ca[i] > numClusters){
				numClusters = ca[i];
			}else if(ca[i] == -1){
				//found an outlier
				System.out.print("  " + i);
			}
		}
		if(numClusters == -1){
			System.out.println("No clusters created.");
			return;
		}
		numClusters++; //clusters start at 0, so one more
exists than highest value.
		int currentCluster = 0;//cluster pointer
		//loop for each cluster
		for(int i = 0; i < numClusters; i++){

			System.out.println("\nCluster " + i + ":");
			//check each passage
			for(int j = 0; j < ca.length; j++){
				if(ca[j] == i){
					System.out.print("  " + j);
				}
			}
		}
		System.out.println("");

	}

	/**
	 * Print double array to screen. Used for testing.
	 * @param da double array to be printed.
	 */
	private void printArray(double[] da){
		for(int i = 0; i < da.length; i++){
			System.out.println(i + " " + da[i]);
		}
	}
	/**
	 * Print int array to screen. Used for testing.
```

```
	 * @param ia int array to be printed.
	 */
	private void printArray(int[] ia){
		for(int i = 0; i < ia.length; i++){
			System.out.println(i + " " + ia[i]);
		}
	}

	/**
	 * Print part of a double array to screen. Used for
printing avg/median statistics.
	 * @param da double array to be printed.
	 * @param printList Part of array to be printed
	 */
	private void printPartArray(double[] da, int[] printList){
		//print list in reverse order.
		for (int i = printList.length - 1; i >= 0 ; i--){
			System.out.println(printList[i] + " " +
da[printList[i]]);
		}
		/*Arrays.sort(printList);
		int position = 0;
		for(int i = 0; i < da.length; i++){
			if(i == printList[position]){
				System.out.println(i + " " + da[i]);
				position++;
				if(position >= printList.length) return;
			}
		}*/
	}

	/**
	 * Print which passes had the highest values.
	 * @param da Array has values ranking passages.
	 * @param top How many passages to keep.
	 * @return Array listing the ranking of top passages.
	 */
	private int[] topX(double[] da, int top){
		int[] topPassages = new int[top];
		double[] values = new double[top];
		//for(int i = top; i < da.length; i++){

		//for each passage
		for(int i = 0; i < da.length; i++){
			int spot = -1;//default not in top x
			//check to see if it is top x, start at lowest
position
			for(int j = 0; j < top; j++){
				if(da[i] > values[j]){
					spot = j;
				}else{
					break;
```

```
                        }
                  }
                  if(spot != -1){
                        //move to top x, move others down;
                        for(int j = 0; j < spot; j++){
                              values[j] = values[j + 1];
                              topPassages[j] = topPassages[j +
1];
                        }
                        values[spot] = da[i];
                        topPassages[spot] = i;
                  }
            }

            return topPassages;
      }

      /**
       * Gets similarity matrix saved under "Corpus" and
"wordlist"
       * @param args First arg is file with passages to be
compared. Second arg is corpus name to save similarity matrix
under.
       */
      public static void main(String[] args) {
            LSAanalysis lsa = new LSAanalysis();
            lsa.loadCorpus("Corpus", "WordList");
            //lsa.readCompare("Compare");
            if(args.length == 2){
                  lsa.readCompare(args[0]);
            }else{
                  System.out.println("Failed due to bad
argument.");
                  System.exit(0);
            }
            lsa.makeSimilarityMatrix();
            lsa.similarityAnalysis();
            lsa.printSimilarity("Similarity." + args[1]+ "."
+args[0]);
            System.out.println("done");

      }
}
```

LSAstats.java

```
import java.util.*;

/**
 * Class used to find statistics from a similarity matrix.
 *
```

```
 * @author Mark Paul
 * kramluap@u.washington.edu
 * Last modified 1/24/07
 */
public class LSAstats {
      private int nPassages;
      private double[][] simMatrix;//passages X passage matrix,
[i][j] = [j][i]
      private boolean useOutliers;
      private int[] outliers;//list of passages in order, with
lowest passage first.

      /**
       * Constructor
       * @param simMatrix passage X passage matrix with values
representing similarity
       */
      public LSAstats(double[][] simMatrix){
            nPassages = simMatrix.length;
            this.simMatrix = simMatrix;
            useOutliers = false;
      }


      /**
       * Performs hierarchical clustering using complete-linkage
algorithm.
       * Returns int array with location of cluster for each
passage. -1 if outlier, first cluster is 0.
       * @return Array identifies which cluster each passage is
assigned to. -1 for outliers.
       */
      public int[] completeLinkage(){
            int[] clusterLoc = new int[nPassages];//keeps track
of cluster location of each passage
            Vector clusters = new Vector(); //vector of ints[],
each int[] represents a cluster
            double[][] clusterMatrix = simMatrix;//similarity
values between clusters
            //remove outliers
            if(useOutliers){
                  if(outliers.length > 0){//make sure there are
outliers
                        int outlierPosition = outliers.length -
1; //remove outliers in reverse order
                        for(int i = nPassages - 1; i >= 0; i--){
                              if(outliers[outlierPosition] == i){
                                    //ignore outlier
                                    clusterMatrix =
removePassage(i,clusterMatrix);
                                    clusterLoc[i] = -1;//passage
no longer assigned to a cluster
```

```
                                    outlierPosition--;//move on
to next outlier
                                    if(outlierPosition == -1){
                                          //done removing
outliers
                                          break;
                                    }
                              }
                        }
                  }
            }
            //set up initial clusters
            for(int i = 0; i < simMatrix.length; i++){
                  //check for outlier
                  if(clusterLoc[i] != -1){
                        int[] clusterItems = new
int[1];//represents cluster containing one passage
                        clusterItems[0] = i;//cluster points to
position in clusterMatrix
                        clusters.add(clusterItems);//add cluster
to list of clusters
                  }
            }

            //set max cluster size to be equal to half of the
passages to be clustered.
            int maxCluster = clusters.size() / 2;

            //merge clusters until break
            while(true){
                  //find nearest neighbor
                  int nearRow = -1;
                  int nearCol = -1;
                  double nearValue = -1;
                  //look at each pair
                  for(int i = 0; i < clusterMatrix.length; i++){
                        for(int j = 0; j < i; j++){
                              if(nearValue <=
clusterMatrix[i][j]){
                                    //new nearest neighbor
                                    nearValue =
clusterMatrix[i][j];
                                    nearRow = i;
                                    nearCol = j;
                              }
                        }
                  }
                  //check to see if new cluster would contain
over half of the passages.
                  //if it does exit.
```

```
                  //merge two clusters to create new cluster
                  double[] newValues = new
double[clusterMatrix.length - 1]; //similarity values for new
cluster
                  int newValuesPos = 0;//pointer to newValues
location
                  //find new similarity values by picking the
lowest similarity value
                  //of each merged cluster to the other clusters.
                  for(int i = 0; i < clusterMatrix.length; i++){
                        if(i == nearRow || i == nearCol){
                              //remove similarity values between
the two merged clusters
                        }else{
                              //choose lowest value due to
complete-linked algorithm
                              if(clusterMatrix[i][nearRow] >
clusterMatrix[i][nearCol]){
                                    newValues[newValuesPos] =
clusterMatrix[i][nearCol];
                              }else{
                                    newValues[newValuesPos] =
clusterMatrix[i][nearRow];
                              }
                              newValuesPos++;
                        }
                  }
                  //compare to self, resulting in similarity
value of 1 (perfect match)
                  newValues[newValuesPos] = 1;

                  //merge cluster values
                  int[] clusterPass1 =
(int[])clusters.get(nearRow);//passages in first cluster
                  int[] clusterPass2 =
(int[])clusters.get(nearCol);//passages in second cluster
                  int[] newClusterPass = new
int[clusterPass1.length + clusterPass2.length];//new cluster

                  //if new cluster will contain half of the
passages or more, then finished.
                  if(newClusterPass.length >= maxCluster){
                        break;
                  }

                  //add passages from first cluster to new
cluster
                  for(int i = 0; i < clusterPass1.length; i++){
                        newClusterPass[i] = clusterPass1[i];
                  }
                  //add passages from second cluster to new
cluster
```

```
                  for(int i = 0; i < clusterPass2.length; i++){
                        newClusterPass[clusterPass1.length + i] =
clusterPass2[i];
                  }
                  //remove both passages, with highest passage
first
                  if(nearRow > nearCol){
                        clusters.remove(nearRow);
                        clusters.remove(nearCol);
                  }else{
                        clusters.remove(nearCol);
                        clusters.remove(nearRow);
                  }

                  clusters.add(newClusterPass);//add new cluster
to list

                  //remove vectors;
                  clusterMatrix = removePassage(nearRow,
clusterMatrix);
                  clusterMatrix = removePassage(nearCol,
clusterMatrix);
                  //add vector
                  clusterMatrix = addPassage(newValues,
clusterMatrix);
            }

            //create return array
            //for each cluster
            int clusterPointer = 0;//point to cluster
            int minClusterSize = 2;//smallest cluster size to not
be considered an outlier
            for(int i = 0; i < clusters.size(); i++){
                  int[] cluster = (int[])clusters.get(i);
                  if(cluster.length < minClusterSize){
                        //assign cluster to outlier
                        //for each passage in cluster
                        for(int j = 0; j < cluster.length; j++){
                              clusterLoc[cluster[j]] = -1;
                        }
                  }else{
                        //new cluster
                        //for each passage in cluster
                        for(int j = 0; j < cluster.length; j++){
                              //if(cluster.length > 1)
System.out.print("  " + cluster[j]);
                              clusterLoc[cluster[j]] =
clusterPointer;
                        }
                        clusterPointer++;
                  }
            }
```

```
            return clusterLoc;
      }

//    clusterHierarchical uses
      /**
       * Adds a cluster to a matrix of containing similarity
values between clusters.
       * @param newPassage Cluster to be added.
       * @param passageMatrix Similarity values between clusters.
       * @return New similarity values between clusters.
       */
      private double[][] addPassage(double[] newPassage,
double[][] passageMatrix){
            int newClusters = passageMatrix.length + 1;
            double[][] newMatrix = new
double[newClusters][newClusters];
            for(int i = 0; i < passageMatrix.length; i++){
                  //copy matrix
                  for(int j = 0; j < passageMatrix.length; j++){
                        newMatrix[i][j] = passageMatrix[i][j];
                  }
                  //add column for new passage
                  newMatrix[i][passageMatrix.length] =
newPassage[i];
            }
            //add new row
            for(int i = 0; i < newMatrix.length; i++){
                  newMatrix[passageMatrix.length][i] =
newPassage[i];
            }

            return newMatrix;
      }

      //clusterHierarchical uses
      /**
       * Removes a cluster to a matrix of containing similarity
values between clusters.
       * @param removePass Cluster to be removed.
       * @param passageMatrix Similarity values between clusters.
       * @return New similarity values between clusters.
       */
      private double[][] removePassage(int removePass, double[][]
passageMatrix){
            int newClusters = passageMatrix.length - 1;//number
of similarity values for clusters
            double[][] newMatrix = new
double[newClusters][newClusters];
            int rowLoc = 0;//row position of newMatrix
            int colLoc = 0;//column position of newMatrix
            //Traverses the matrix, removing any similarity value
related to the removed cluster.
```

```
            for(int i = 0; i < passageMatrix.length; i ++){
                  if(i == removePass){
                        //do nothing, which removes the line
                  }else{
                        for(int j = 0; j < passageMatrix.length;
j ++){
                              if(j == removePass){
                                    //do nothing, which removes
the cell
                              }else{
                                    newMatrix[rowLoc][colLoc] =
passageMatrix[i][j];
                                    newMatrix[rowLoc][colLoc] =
passageMatrix[i][j];
                                    colLoc++;
                              }
                        }
                        colLoc = 0;//new row
                        rowLoc++;
                  }
            }
            return newMatrix;
      }

      /**
       * Find outliers of similarity matrix by looking at how
many
       * similar passages each passage has.
       * Suggested value of default threshold is 0.7. The value
of passCount
       * should be equal to the smallest cluster you are willing
to get.
       * @param threshold lowest value to be used to identify
similarity
       * @param passCount number of similar passages needed to
not be considered an outlier.
       */
      public void findOutliers(double threshold, int passCount){
            passCount++;//ignore similarity value where passage
is compared to itself
            Vector outlierVector = new Vector();
            for(int i =0; i < nPassages; i++){
                  int count = 0;
                  for(int j =0; j < nPassages; j++){
                        if(simMatrix[i][j] >= threshold){
                              //found new similar passage
                              count++;
                              if(count >= passCount) break;
                        }
                  }
                  if(count < passCount){
                        //passage is an outlier
```

```
                        outlierVector.add(new Integer(i));
                        System.out.println("Outlier: passage " +
i);
                  }
            }

            //switch from vector to int[]
            outliers = new int[outlierVector.size()];
            for(int i = 0; i < outlierVector.size(); i++){
                  int passOutlier =
((Integer)outlierVector.get(i)).intValue();
                  outliers[i] = passOutlier;
            }
            useOutliers = true;
      }

      /**
       * Find the median value for each passage.
       * @return median value for each passage
       */
      public double[] getMedian(){
            double[] medians = new double[nPassages];
            //for each passage
            for(int i =0; i < nPassages; i++){
                  //calculate average for passage
                  double[] medianSort = new double[nPassages];
                  for(int j = 0; j < nPassages; j++){
                        medianSort[j] = simMatrix[i][j];
                  }
                  Arrays.sort(medianSort);
                  int middle = Math.round(nPassages/2);
                  //System.out.println(medianSort[middle]);
                  medians[i] = medianSort[middle];
            }
            return medians;
      }

      /**
       * Find the average for the highest x values of each
passage.
       * @param top number of passages to look at
       * @return average of high vaules for each passage
       */
      public double[] getTopAverage(int top){
            if(top > nPassages) return null;
            double[] topAvg = new double[nPassages];
            //for each passage
            for(int i =0; i < nPassages; i++){
                  //calculate average for passage
                  double[] arraySort = new double[nPassages];
                  for(int j = 0; j < nPassages; j++){
                        arraySort[j] = simMatrix[i][j];
```

```
                }
                Arrays.sort(arraySort);
                double avg = 0;
                for(int j = 0; j < top; j++){
                      avg += arraySort[nPassages - j - 1];
                }
                avg = avg / top;
                //System.out.println(avg);
                topAvg[i] = avg;
          }
          return topAvg;
    }

    /**
     * Find the average values for each passage.
     * @return average vaules for each passage
     */
    public double[] getAverages(){
          double[] averages = new double[nPassages];
          //for each passage
          for(int i =0; i < nPassages; i++){
                //calculate average for passage
                double avg = 0;
                for(int j = 0; j < nPassages; j++){
                      avg += simMatrix[i][j];
                }
                avg = avg / nPassages;
                //System.out.println(avg);
                averages[i] = avg;
          }
          return averages;
    }
}
```