Text Summarization

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment of the

Requirements for the

Degree Master of Computer Science

By

Youn S, Kim

Dec 2011

© 2011

Youn S, Kim

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Text Summarization using SVD (Singular Value Decomposition) and Lanczos Algorithm

by

Youn S, Kim

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science 11/30/2011

Dr. Chris Tseng, Department of Computer Science 11/30/2011

Dr. Jon Pearce, Department of Computer Science 11/30/2011

Acknowledgement

I would like to thank Dr. Chris Pollett for his excellent guidance throughout this project work and my committee members, Dr. Chris Tseng and Dr. Jon Pearce for their time and effort. Also, a special thanks to my family and friends for their support.

Abstract

With the overwhelming amount of textual information available in electronic formats on the web, there is a need for an efficient text summarizer capable of condensing large bodies of text into shorter versions while keeping the relevant information intact. Such a technology would allow users to get their information in a shortened form, saving valuable time. Since 1997, Microsoft Word has included a summarizer for documents, and currently there are companies that summarize breaking news and send SMS for mobile phones. I wish to create a text summarizer to provide condensed versions of original documents. My focus is on blogs, because people are increasingly using this mode of communication to express their opinions on a variety of topics. Consequently, it will be very useful for a reader to be able to employ a concise summary, tailored to his or her own interests to quickly browse through volumes of opinions relevant to any number of topics. Although many summarization methods exist, my approach involves employing the Lanczos algorithm to compute eigenvalues and eigenvectors of a large sparse matrix and SVD (Singular Value Decomposition) as a means of identifying latent topics hidden in contexts; and the next phase of the process involves taking a high-dimensional set of data and reducing it to a lower-dimensional set. This procedure makes it possible to identify the best approximation of the original text. Since SQL makes it possible to allow analyzing data sets and take advantage of the parallel processing available today, in most database management systems, SQL is employed in my project. The utilization of SQL without external math libraries, however, adds to challenge in the computation of the SVD and the Lanczos algorithm.

Table of Contents

1. Introduction 8

2. Theory and Concepts 10

3. Technologies and Tools Used 14

4. Design and Implementation 15

4.1 Overview 15

4.2 Crawl Blogs 17

4.3 Parse Text into Sentences and Words Using Stop Words and Regular Expressions 21

4.4 Calculate the Frequencies of the Base Forms of the Words for Each Sentence and the TF-IDF Weighting Scheme to Scale Frequencies 21

4.5 Calculate SVD Using the Lanczos Algorithm 22

4.6 Extract a Sentence for Each Topic for the Summary 27

4.7 Back-End Functions and Interfaces 30

4.8 User Interface Functions and Front-End Layer 30

4.9 Summary Evaluation 34

4.9.1 Preparation 35

4.9.2 Test Plan and Datasets 35

4.9.3 Results 36

5. Conclusion 38

List of Figures

Figure 1: Tables created after calculating SVD and generating the summary …..……...19

Figure 2: Tables created in the CS298 database ……21

Figure 3: Sample code for demonstrating the use of PHP to control the process of executing

SQL statements ……..26

Figure 4: Sample code for connecting to MySQL server by using PDO library ……..27

Figure 5: Sample code for the usage of PDO to insert records into a table ……....27

Figure 6: Sample code for adding multiple records in one query ……..28

Figure 7: Sample code for updating multiple records in one query ……....28

Figure 8: Sample code for matrix multiplication in SQL ……..29

Figure 9: Sample code for retrieving values from the VAL column in the EIGENVALUE Tables …….....31

Figure 10: Sample code for retrieving an index with the largest value for each row ……..31

Figure 11: Sample code for sorting array ……..31

Figure 12: Sample code for getting summary ……....32

Figure 13: Representation of the SENTENCE table ……32

Figure 14: Representation of the SUMMARY table ……33

Figure 15: Sample code for using JQuery library to make a request to the server ……34

Figure 16: The user interface ……..34

Figure 17: Sample code for retrieving summary by using the slider control ……35

Figure 18: Pop-up window for posting to Twitter ……..36

Figure 19: Twitter ……...36

List of Tables

Table 1: Representation of the transpose of matrix V ……....30

Table 2: ROUGE-1 scores for the most important sentence……..38

Table 3: ROUGE-1 scores for the five most important sentences ……....38

1. Introduction

The increasing abundance of and easy access to data on the Web can cause information overload. To help readers cope with this problem, an automatic summarization program is needed to reduce the length of the original text, while retaining its essential quality.

As ever more people express their opinions on a variety of topics in blogs, this medium has become an important new area for researchers to find variety of unique and valuable information on various events and topics and on many different subjects. Well known in this regard is the use of specially designed software by a group of computer scientists at Johns Hopkins University. These researchers scanned two billion Twitter communications posted between May 2009 and October 2010 and analyzed the messages for information about a variety of health issues including allergies, insomnia, flu cases, depression, pain, and other ailments. They determined that Twitter posts could in fact be used as a useful source of public health information [13]. My project focuses on blogs; and it will be very useful for users to browse quickly through volumes of opinions from accounts of the daily lives of bloggers to current news on various events, and read concise summaries tailored to their interests. By doing so, one can quickly monitor important local and global events or items of interest, and understand how ideas are spread and trends are set in particular fields.

Text summarization is usually performed by extracting the most important sentences in a document. One approach to determining which sentences are important is to count the word frequency, because the most frequent words will collectively represent the text's main topics. However, this approach creates problems, as it does not take into account co-occurrences among words. To overcome this limitation, I have developed a new Web application that summarizes blogs using the linear algebra concepts, SVD (singular value decomposition) and the Lanczos algorithm, which are implemented through SQL and PHP. SVD is used to find the semantically important sentences through a conceptual mapping into

dimensions of all the sentences in the text, and these dimensions can be interpreted as the main “topics” or “concepts” in the document. These topics or concepts are automatically identified by SVD on the basis of co-occurrences represented by a words-by-sentences matrix. SVD can also perform noise reduction, through the production of a condensed approximation of the original text by combining several related words into a topic or concept.

Since I deal with large data sets of words and sentences from different pages of blogs, I will naturally use a database. Additionally, since most of today's database management systems support parallel processing for executing SQL statements, SQL is employed to take advantage of the parallelism and speed up processing time required for the calculation of the SVD. Since calculating SVD requires many operations including queries, inserts, updates, deletes, aggregations, and copying which of these access and process significant amounts of data in and out of the database. Additionally, if a parallel dataflow language such as Pig developed at Yahoo is used, the computation of SVD for a large data set will be highly efficient, since it allows massive parallel processing of large data sets across clusters by compiling queries written in language called “Pig Latin” into Map Reduce jobs and executing them in Hadoop. In this paper, I will discuss in greater detail how I achieved these goals, including the math concepts and algorithms involved, my technologies and tools, and my AJAX-driven Web user interface. I will describe each step of the process of gathering texts from blogs, parsing them into words and sentences, counting frequencies and calculating TF-IDF, computing SVD, generating the summary, and lastly, presenting the summary to users. I will also use an automatic evaluation tool called ROUGE (Recall-Oriented Understudy for Gisting Evaluation) to determine the quality of the generated summary.

In the section, “Theory and Concepts,” I will define matrices, eigenvalues and eigenvectors, the QL and QR algorithms, the Lanczos algorithm, and the calculation of SVD. “Technology and Tools Used” will briefly discuss the programming languages and development tools utilized in the project. In “Design and Implementation,” I will show how I

created my program which generates and presents summaries of blogs to users. In "Summary Evaluation," I will report and measure the results with statistics and test them for veracity. Finally, I will present my conclusions.

2. Theory and Concepts

Since I deal with matrices throughout this project, I will first define what a matrix is. A matrix is a table of rows and columns that contains data. It can also be viewed as a collection of row vectors or column vectors. A vector is a sequence of numbers corresponding to measurements along a given dimension. Common matrix operations include addition, subtraction, multiplication, and transposition.

SVD (singular value decomposition) is based on the linear algebra theorem that a rectangular matrix A can be decomposed into the product of three matrices (an orthogonal matrix U, a diagonal matrix S, and the transpose of an orthogonal matrix V), and reconstructed by multiplying the three matrices together. The SVD theorem is usually presented as:

$A_{nxp} = U_{nxn} S_{nxp} V^T_{pxp}$ where $UU^T = U^T U = I$ and $V^T V = VV^T = I$.

Calculating SVD consists of finding the eigenvalues and eigenvectors of AA^T and $A^T A$. The eigenvectors of AA^T make up the columns of U, and those of $A^T A$ make up the columns of V. The singular values in S contain the square roots of the eigenvalues from AA^T or $A^T A$ and are placed along the diagonal of S in descending order.

SVD is also a method for reducing a high-dimensional set of data to a lower-dimensional set. It allows us to identify which data exhibit the most variation through an ordering of the dimensions. It gives us the best approximation of the original data by simply ignoring dimensions below certain thresholds, and in doing so this approachreduces the volume of content, while maintaining the main relationships that are present.

To calculate SVD, it is essential to find the eigenvalues and eigenvectors. If a

nonzero vector satisfies the equation below, vector v is called an eigenvector, and scalar λ is called an eigenvalue.

$A\vec{v} = \lambda\vec{v}$, where A is a square matrix

The QL and QR algorithms are used for finding eigenvalues on the tridiagonal matrix by reducing it to a diagonal. In the QR algorithm, the matrix must be factored by the Gram-Schmidt orthogonalization process (a method for converting a set of vectors into orthonormal vectors),into a product of an orthogonal matrix Q_1 and an upper triangular matrix R_1 with positive entries along the diagonal. Next, we multiply the two factors (R_1, Q_1) in reverse order. Therefor, we have $A_2 = R_1 Q_1$. We then repeat the steps above until all entries below the subdiagonal are zero. After a sufficient number of iterations k, the eigenvalues appear along its diagonal on $A_{k.}$.

The QL algorithm is very similar to the QR algorithm, the only difference being that all entries above the subdiagonal are zero in the former, while all those below the subdiagonal are zero in the latter. Since the smallest elements are placed in the upper left corner, and the largest in the lower right corner of the matrix in the QL algorithm, the QL algorithm has less round-off errors than the QR algorithm. Thus, the QL algorithm is preferred. The QL algorithm with implicit shifts is mathematically equivalent to the original QL algorithm. But using the technique of implicit shifting, the convergence of the QL algorithm can be accelerated since it increases the separation between eigenvalues and requires fewer operations. Thus, I use the QL algorithm with implicit shifts to calculate eigenvalues and eigenvectors.

The Lanczos algorithm can determine the eigenvalues for a large sparse matrix efficiently through the employment of the Lanczos recursion, which converts the original matrix A into tridiagonal matrix T through a finite number of orthogonal similarity transformations.

Suppose we have orthonormal vectors $q_1, q_2, q_3, \ldots\ldots q_k$ and let Q = $[q_1, q_2, \ldots q_k]$;

thus, $Q^T Q = I$.

Since we want to change A to a tridiagonal matrix T, we apply a similarity transformation.

Then T can be decomposed into $Q^T AQ$.

We can rewrite the equation as AQ=QT by multiplying both sides by Q.

The j[th] column of the above equation can be viewed as:

$$\begin{bmatrix} & A & \end{bmatrix}\begin{bmatrix} & q_j & \end{bmatrix} = \begin{bmatrix} q_{j-1} & q_j & q_{j+1} \end{bmatrix}\begin{bmatrix} \cdot & \cdot & \cdot & & \\ & \cdot & \cdot & \beta_{j-1} & \\ & & \cdot & \alpha_j & \cdot \\ & & & \beta_j & \cdot \quad \cdot \\ & & & \cdot & \cdot \quad \cdot \end{bmatrix}$$

$A q_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}$ ……..(1).

If we rewrite the above equation by multiplying both sides by q_j^T,

$q_j^T A q_j = \beta_{j-1} q_j^T q_{j-1} + \alpha_j q_j^T q_j + \beta_j q_j^T q_{j+1}$.

$q_j^T q_{j-1}$ and $q_j^T q_{j+1}$ will be zero, and we get $\alpha_j = q_j^T A q_j$.

If we rearrange (1), we can obtain β_j.

$r_j = \beta_j q_{j+1} = A q_j - \alpha_j q_j - \beta_{j-1} q_{j-1}$ ……..(2)

$\|r_j\| = \beta_j (q_{j+1}^T q_{j+1}) = \beta_j$.

We can decide the next orthonormal vector by rearranging (2),

$q_{j+1} = r_j / \beta_j$,

continuing the recursion until j =n, where n is the size of A. T_n is an orthogonal similarity transformation of A and therefore has identical eigenvalues.

By summarizing the above, the Lanczos algorithm can be written as:

r_0 = initial vector

$\beta_0 = \|r_0\|$.

For j =1, 2, 3, ….n

$q_j = r_{j-1} / \beta_{j-1}$

a = A q_j

$\alpha_j = q_j^T$ a

r_j = a - $\beta_{j-1} q_{j-1}$ - $\alpha_j q_j$

$\beta_j = \|r_j\|$.

After n number of Lanczos recursions, an nxn tridiagonal matrix is generated. The eigenvalues for this matrix are approximate to those of the original matrix, A.

Moreover, the eigenvectors of A can be found through the multiplication of the eigenvectors of T by the Lanczos vectors acquired from the recursion. I chose the Lanczos algorithm because the number of arithmetical operations required to generate a Lanczos matrix is proportional to the number of nonzero entries of A (I). This saves running time for a large sparse matrix.

Notwithstanding my calculations, even after a small number of iterations, I quickly lost the orthogonality of the Lanczos vectors because I was unable to completely reduce them to tridiagonal form. During each Lanczos recursion, full reorthogonalization of the current vector was needed in relation to all previous vectors. This was carried out through a Gram-Schmidt process, which may be described as:

$r_j = r_j - \sum_{k=1}^{j-1} (q_k^T r_j) q_k$ j = Lanczos step.

For my project, I worked with a word-by-sentence matrix A, where A_{ij} represents the frequency of a particular word appearing in each sentence. Earlier, I mentioned that A can be decomposed into three matrices (U, S, and V^T). Thus, in our case, the word matrix U consists of one row vector for each word, the sentence matrix V^T consists of one

column vector for each sentence, and the singular matrix S consists of single values along the diagonal of matrix U, reflecting the importance of each dimension. To be specific, each number U_{ij} indicates how strongly related a word i is to the topic or concept represented by semantic dimension j, while each number V_{ij} indicates how strongly related sentence i is to the topic represented by semantic dimension j. Each number S_{ii} on the diagonal of S indicates the importance of the corresponding semantic dimension.

I use the Lanczos algorithm because a word-by-sentence matrix is very sparse, meaning that it is populated primarily by zeros, because the same words seldom reappear in adjoining sentences. The Lanczos algorithm is the fastest method for solving eigenvalues on large sparse matrices.

3. Technologies and Tools Used

This section describes the technologies and tools employed to meet the requirements of the project. The front end was developed using JQuery, JavaScript and HTML. I employed JQuery to make Ajax requests, manipulate the DOM and CSS, and to add effects and animations. The back end was developed using PHP and SQL. I used PHP to respond to JQuery Ajax requests, execute SQL queries, and dynamically create pages. I also used it to control the execution of SQL statements when calculating SVD, and I mainly used SQL to calculate SVD. To configure the Windows operating system for the development environment for the project, XAMPP was installed in place of separate components such as Apache Web Server and MySql.

4. Design and Implementation

My project is implemented in seven major steps. The first is to crawl blogs for text, and the second is to parse text into sentences and words using stop words and regular expressions. The third step involves calculating the frequencies of the base forms of the words in each sentence and the use of the TF-IDF weighting scheme to scale frequencies. The fourth is to compute SVD through the Lanczos algorithm. Fifth, a sentence for each topic is extracted for the summary. The sixth step implements back-end functions and interfaces for posting to Twitter and for responding to AJAX requests, for example, to add new blogs, remove existing blogs, update existing blogs, extract sentences from the database, and shorten the URL using Bit.Ly. The seventh and final step is to create user interface functions and the front-end layer.

4.1 Overview

Below are the steps I took to generate the summary in the back end of the application.

① Crawl posts in a blog.

② Parse texts into words and sentences.

③ Remove stop words and store only the roots of the others.

④ Count frequencies for each word in each sentence, calculate TF-IDF weighting scheme to scale frequencies, and store the result.

⑤ Using the above result as an input, and run the Lanczos algorithm to generate a tridiagonal form.

⑥ Use the eigenvalues and eigenvectors from this tridiagonal matrix to calculate SVD, and store the result.

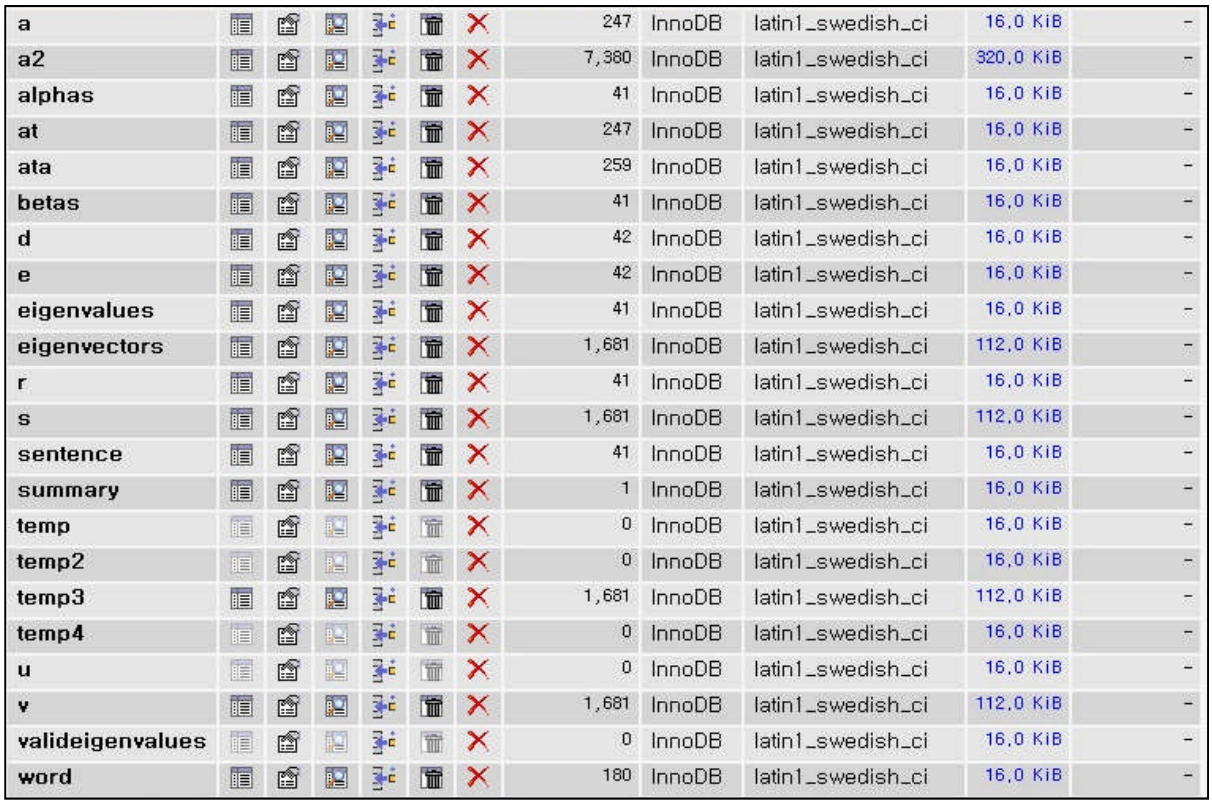
⑦ Arrange the eigenvalues and eigenvectors into U, S, and V^T.

⑧ From the V^T matrix, extract a sentence for each topic.

⑨ Return to step 1 until all the desired blogs have been visited and processed.

After performing the above steps, tables such as A, A2, ALPHAS, AT, ATA, BETAS, EIGENVALUES, EIGENVECTORS, SENTENCE, SUMMARY, WORD, U, and V will be created and stored.

The A table contains the weighted frequencies of words and sentences, while the A2 table contains the frequencies of words and sentences alone. The ALPHAS table initially contains the diagonal elements of a tridiagonal matrix generated through the employment of the Lanczos algorithm, but later contains the eigenvalues of the tridiagonal matrix. The AT table is a transposition of the A table. The ATA table contains the multiplication values of the A and AT tables. The BETAS table shows the super-diagonal elements of the tridiagonal matrix acquired by running the Lanczos algorithm. The EIGENVALUES table contains the eigenvalues of the tridiagonal matrix copied from the ALPHAS table. The EIGENVECTORS table displays the eigenvectors of the tridiagonal matrix. The V table initially contains Lanczos vectors, but later contains the values from the right singular matrix in SVD. The SENTENCE table stores the sentences parsed from the original text, and the WORD table stores the words parsed from these sentences. The SUMMARY table contains sentences to be included in the summary. The U table contains the values from the left singular matrix in SVD. The rest of the tables are not significant and are only temporary.



a							247	InnoDB	latin1_swedish_ci	16,0 KiB	-
a2							7,380	InnoDB	latin1_swedish_ci	320,0 KiB	-
alphas							41	InnoDB	latin1_swedish_ci	16,0 KiB	-
at							247	InnoDB	latin1_swedish_ci	16,0 KiB	-
ata							259	InnoDB	latin1_swedish_ci	16,0 KiB	-
betas							41	InnoDB	latin1_swedish_ci	16,0 KiB	-
d							42	InnoDB	latin1_swedish_ci	16,0 KiB	-
e							42	InnoDB	latin1_swedish_ci	16,0 KiB	-
eigenvalues							41	InnoDB	latin1_swedish_ci	16,0 KiB	-
eigenvectors							1,681	InnoDB	latin1_swedish_ci	112,0 KiB	-
r							41	InnoDB	latin1_swedish_ci	16,0 KiB	-
s							1,681	InnoDB	latin1_swedish_ci	112,0 KiB	-
sentence							41	InnoDB	latin1_swedish_ci	16,0 KiB	-
summary							1	InnoDB	latin1_swedish_ci	16,0 KiB	-
temp							0	InnoDB	latin1_swedish_ci	16,0 KiB	-
temp2							0	InnoDB	latin1_swedish_ci	16,0 KiB	-
temp3							1,681	InnoDB	latin1_swedish_ci	112,0 KiB	-
temp4							0	InnoDB	latin1_swedish_ci	16,0 KiB	-
u							0	InnoDB	latin1_swedish_ci	16,0 KiB	-
v							1,681	InnoDB	latin1_swedish_ci	112,0 KiB	-
valideigenvalues							0	InnoDB	latin1_swedish_ci	16,0 KiB	-
word							180	InnoDB	latin1_swedish_ci	16,0 KiB	-

Figure 1: Tables created after calculating SVD and generating the summary

Below are the processes required for response to user requests in the front end of the application, including button clicks and slider movements .

1. When a user makes an AJAX request, the application will display a message such as “Loading...” Then make a POST or GET a request to the PHP script such as getXXX.php, which will accept the request, process it, and return the result.
2. If the response is successful, display the result; otherwise, show an error message.

4.2 Crawl Blogs

A blog contains many individual posts by the author. When all of these posts are visited, the blog is marked as read and is moved from the UNVISITEDBLOGS to the VISITEDBLOGS table.

To crawl many different blogs, an initial URL must be selected. This URL can lead to any blog and is randomly chosen. In my project, I chose http://bashfulblogger.wordpress.com as the initial URL, which is the same as

http://bashfulblogger.wordpress.com/index.php. If summarization is being resumed, the URL is retrieved from the UNVISITEDBLOGS table. While visiting a blog, the program extracts URLs to other blogs, as well as many individual posts within the same blog that belong to the wordpress.com domain. URLs are extracted using regular expressions.

Posts within a blog are visited using the CURL PHP library. We can easily tell whether the application connected and processed a post by checking its returned HTTP status. A successful connection is usually indicated by a 2XX HTTP code. If a post fails to crawl, the next URL in the queue from the LINKS table is processed.

After a post is crawled successfully, sentences are stored in the SENTENCES table in the CS298 database along with information such as domain and subdomain. For each URL link to other blogs, the domain and subdomain are parsed and stored in the UNVISITEDBLOGS table if they are not already there. Each URL linking to posts within the same blog is stored in the LINKS table of the database along with information including domain, subdomain and URL. In our case, the subdomain is usually the name of a blog.

After crawling a post, sentences are extracted. If this is successful, a new database, A, is created and named with the concatenating domain, subdomain and unique id from the LINKS table with a "_" character. When the URL is inserted into the LINKS table, a unique id is generated from the database and put into the ID column. Inside the newly created database A, sentences having the same domain and subdomain as a given post are retrieved from the SENTENCES table in the CS298 database and stored in the SENTENCE table. The next URL to posts within the same blog will then be extracted from the LINKS table and processed. Sentences are stored in the SENTENCE table, while words are stored in the WORDS table in database A.

The figure below shows the tables created in the CS298 database.

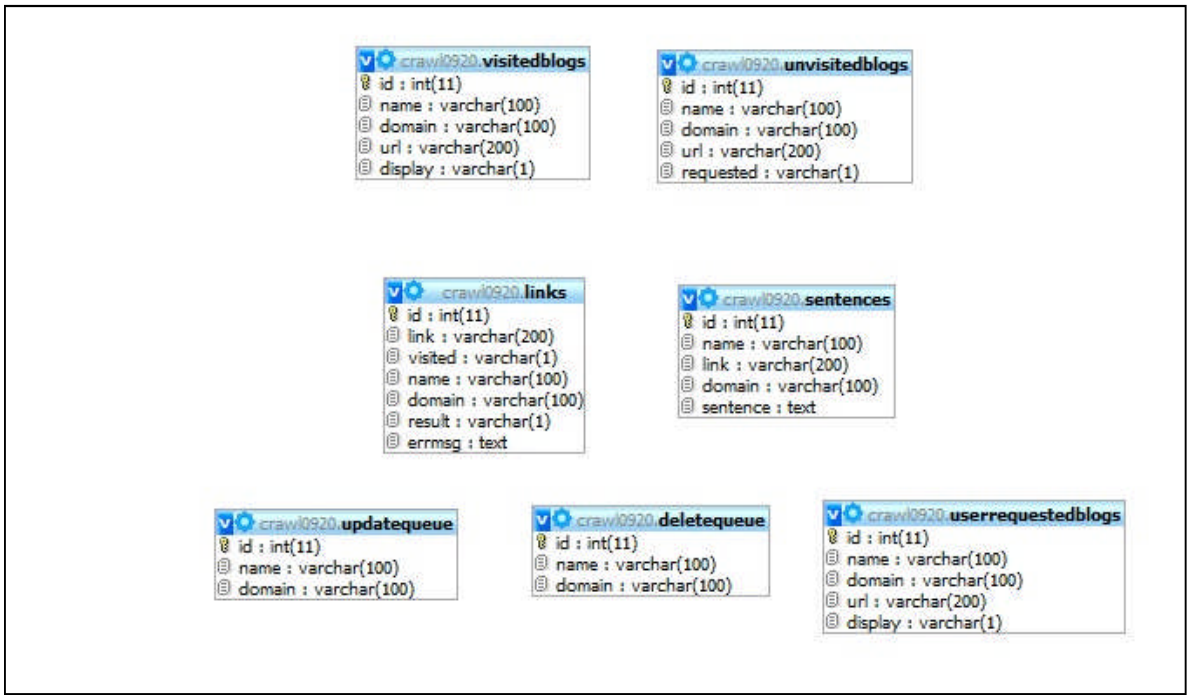


Figure 2: Tables created in the CS298 database

Furthermore, my program keeps track of requests from users such as for adding new blogs and updating or removing existing ones. Such requests are handled when the program starts or resumes, or when it finishes summarizing the currently processing blog if it is already running

The steps below briefly explain how my program crawls blogs.

1. Generate a database and create tables, or use an existing database.
2. Execute the startCrawling() function, where the following occurs:
    a. Determine whether or not there are any requests in the UPDATEQUEUE table. If so, update the previous result by following, again, the process for extracting the summary.
    b. Determine whether or not there are any requests in the DELETEQUEUE table. If so, delete these blogs from the VISITEDBLOGS, UNVISITEDBLOGS, or USERREQUESTEDBLOGS table.
    c. Check for requests in the USERREQUESTEDBLOGS table. If they exist, run the process for extracting the summary.

d. Retrieve a blog from the UNVISITEDBLOGS table and repeat the loop until no blogs are left there. Inside the loop, the following occurs:
e. Parse the URL for the retrieved blog into domain and subdomain.
f. Check if the parsed domain is within the pre-defined domains such as wordpress.com.
g. Determine whether or not the parsed domain and subdomain have already been visited by checking them against any matched blogs in the VISITEDBLOGS table. If so, skip them.
h. Execute the crawlBlog() function, where the following occurs:
    i. Visit a page using the URL passed via function argument.
    ii. Extract the URLs to individual posts within the same or other blogs.
    iii. Take sentences from the crawled text and store them in the SENTENCES table along with information such as domain and subdomain.
    iv. Create a new database, retrieve sentences from the SENTENCES table, and store them in the SENTENCE table in the newly created database.
    v. Determine whether or not the extracted URL is valid. Depending on what it is, insert it in the LINKS or UNVISITEDBLOGS table.
    vi. Check for requests in the UPDATEQUEUE table. If any exist, update the previous result by following the process for extracting the summary again.
    vii. Determine whether or not there are any requests in the DELETEQUEUE table. If so, delete the requested blogs from the VISITEDBLOGS, UNVISITEDBLOGS, or USERREQUESTEDBLOGS table.

viii. Check for requests in the USERREQUESTEDBLOGS table. If any are present, run the process for extracting a summary.

i. Check for any records in the UNVISITEDBLOGS table. If there are none, exit.

4.3 Parse Text into Sentences and Words Using Stop Words and Regular Expressions

Initially, sentences are extracted using regular expressions, while JavaScript, pictures, tags and special HTML characters are removed and stored in the SENTENCES table in the CS298 database. After a database for the post is created, sentences are retrieved from the SENTENCES table in the CS298 database and inserted in the new SENTENCE table. Furthermore, words are extracted using regular expressions and placed in the individual post's WORD table.

Stop words are commonly used words such as "a," "the," and "for," and are usually ignored to reduce the size of the input since they do not add much value. The file stopWords.txt contains a list of these words.

4.4 Calculate the Frequencies of the Base Forms of the Words for Each Sentence and the TF-IDF Weighting Scheme to Scale Frequencies

Stemming is the process of reducing words to their roots. I used a free PHP implementation of the Porter stemming algorithm to return the stemmed words. This algorithm was developed at the University of Cambridge in 1980 by Martin Porter. The frequencies per sentence for each word are counted using the database's previously created WORDS and SENTENCE tables.TF-IDF stands for Term Frequency-Inverse Document Frequency. This method evaluates words in each sentence to minimize the influence of those that are very common across the document and do not carry much meaning. The importance of a word is high if it is frequent in a particular sentence, but less frequent in others.

TF-IDF is equal to TF*IDF. Although there are many different weighting formulas, TF and IDF are commonly computed as follows:

TF(i,j) is the number of the occurrences of the word i in sentence j divided by the sum of the occurrences of all the words in sentence j.

IDF (i) = log(|S| / S_i), where |S| is the total number of sentences in the input text, and S_i is the number of sentences in which the word i appears.

4.5 Calculate SVD Using the Lanczos Algorithm

One of the most efficient techniques for finding the eigenvalues and eigenvectors of a symmetric real matrix is the combination of the Lanc". The Lanczos algorithm reduces this matrix to a tridiagonal form from its orthogonal projection onto the corresponding subspace spanned by the Lanczos vectors generated, followed by a QR or QL algorithm. The QL algorithm is known to have smaller round-off errors than the QR algorithm. The QR algorithm can in fact give extreme round-off errors, and thus should be avoided. Round-off errors can come from computation and storage or from each arithmetic operation. The best example is 1/3. In the case of addition, the result of an arithmetic operation cannot also be represented exactly as a valid floating point number. Thus, the computer rounds to a number adjacent to two normalized floating point numbers, introducing a round-off error. Such errors can cause serious problems, since calculating SVD entails a large number of repetitive arithmetic operations involving floating point numbers.

As previously mentioned, the basic idea behind the QL algorithm is that any real matrix can be decomposed into the form:

C = Q · L, where Q is the orthogonal matrix, and L is the lower triangular one.

The QL algorithm works as follows. Once a tridiagonal matrix C is obtained:

① Find its Q · L decomposition.

② Generate C_1= L · Q.

③ Find the $C_1 \cdot Q_1$ decomposition of C_1.

④ Generate $C_2 = L_1 \cdot Q_1$

⑤ Obtain the $Q_2 \cdot L_2$ decomposition of C_2.

⑥ Continue the above steps until the off-diagonal elements are annihilated. The cost of this algorithm is $O(n^3)$ per iteration for a general matrix, but only O(n) per iteration for a tridiagonal matrix, which is why the Lanczos algorithm is used before the QL algorithm.

Since the QL algorithm with implicit shifts is more or less equivalent to the QL algorithm, but is known to work better in practice, I employed it to determine the eigenvalues and eigenvectors of the real, symmetric, tridiagonal matrix obtained from running the Lanczos algorithm in my project. More information on the QL algorithm with implicit shifts can be found in the *Handbook for Auto. Comp., Vol. ii: Linear Algebra*, by Bowdler, Martin, Reinsch, and Wilkinson (1971), pp. 227-240.

In short, after iterations of a theoretically infinite sequence of orthogonal similarity transformations by the QL algorithm with implicit shifts, the symmetric matrix is converted to a lower diagonal matrix whose diagonal elements are both the eigenvalues and a matrix that contains the eigenvectors. Thus, if A is symmetric, then A = V·D·V', where the eigenvalue matrix D is diagonal and the eigenvector matrix V is orthogonal. In other words, the diagonal values of D are the eigenvalues, and V·V' = I, where I is the identity matrix. The columns of V represent the eigenvectors in the sense that A·V = VD.

PHP is mainly used to direct the execution of the series of SQL statements needed for calculating SVD, such as loops and decision control structures includingif else, for, and while. The snippet of code below demonstrates the use of PHP to control the process of executing SQL statements.

```
$while($j < $n)
{
        if($j == 0)
        {
                $beta = $beta0;
        }
        else
        {
                $statement = "SELECT val FROM BETAS WHERE i = :i);
                $query->$pdo->prepare($statement);
                $query->bindParam(':i',$j-1,PDO::PARAM::INT);
                $query->execute();

        }

        $statement = "UPDATE U SET val = (CASE ";
        for($i=0; $i<$n; $i++)
        {
                $statement .= "WHEN (i=$i) THEN (SELECT val FROM r WHERE i = $i) ";
        }
        $statement .= "END) WHERE j=$j";
        $query = $pdo->prepare($statement);
        $query->execute();
}
```

Figure 3: Sample code for demonstrating the use of PHP to control the process of executing SQL statements

To provide a uniform means of accessing multiple databases, PDO (PHP Data Objects) is used to perform INSERT, DELETE, UPDATE, and SELECT operations in place of either the mysql or mysqli extensions.

I employed the following steps when using PDO.

① Connect to the MySQL server by calling “new PDO()” to obtain a database handle object, as shown in the code below.

```
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$pdo = new PDO("mysql:host=localhost;dbname=$dbname",$dbuser, $dbpass);
```

Figure 4: Sample code for connecting to MySQL server by using PDO library

② Use the database handle to issue SQL statements or retrieve records returned by the statements. As an example, the following piece of code shows the use of PDO to insert records in a table.

```
$query = $pdo->prepare("INSERT INTO temp SELECT i,0,val FROM U WHERE j=:j");
$query->bindParam(':j',$j,PDO::PARAM::INT);
$query->execute();
```

Figure 5: Sample code for the usage of PDO to insert records into a table

③ Disconnect by setting the database handle object to null when the handle is no longer needed.

To calculate SVD in SQL more quickly, I employed several optimization techniques.

① When tables are no longer needed, they are truncated rather than deleted. Calling the TRUNCATE statement removes all the records with no transaction log from which to restore them, and thus it is much faster than calling the DELETE statement.

② Rather than calling several INSERT statements, one INSERT statement is used to add multiple values in one query with the help of PHP. The code snippet below demonstrates how this is done.

```
for($i=0; $i < $n; $i++)
{
        $statement = "INSERT INTO V VALUES";
        for($ii=0; $ii < $n; $ii++)
                $statement .= " ($ii,$i,0.0),"
        $statement = substr($statement,0,strrpos($statement, ","));
        $query = $pdo->prepare($statement);
        $query->execute();
}
```

Figure 6: Sample code for adding multiple records in one query

③ Rather than calling several UPDATE statements, with the help of PHP sonly one statement is used to update multiple values in a single query. The code snippet below is an example.

```
$statement = "UPDATE V SET val = ( CASE ";
for($z=0; $z < $n; $z++)
{
        $statement .= "WHEN (i=$z) THEN ($resultj[$z]) ";
}
$statement .= "END ) WHERE j=$jj";
$query = $pdo->prepare($statement);
$query->execute();
```

Figure 7: Sample code for updating multiple records in one query

The steps below illustrate how SVD is calculated with the Lanczos algorithm using SQL.

① Perform $A^{T}A$ using SQL and store the result in the ATA table, as in the snippet below.

```
$statement = "DELETE FROM A WHERE val = 0";
$query = $pdo->prepare($statement);
$query->execute();
$statement = "CREATE TABLE AT (i INT, j INT, val DOUBLE)";
$query = $pdo->prepare($statement);
$query->execute();
$statement = "INSERT INTO AT SELECT A.j AS i, A.i AS j, A.val FROM A";
$query = $pdo->prepare($statement);
$query->execute();
$statement = "CREATE TABLE ATA (i INT, j INT, val DOUBLE)";
$query = $pdo->prepare($statement);
$query->execute();
$statement = "INSERT INTO ATA SELECT AT.i AS i, A.j AS j, SUM(AT.val * A.val) FROM AT, A WHERE AT.j = A.i GROUP BY AT.i, A.j";
$query = $pdo->prepare($statement);
$query->execute();
```

Figure 8: Sample code for matrix multiplication in SQL

② Run a Lanczos algorithm on A^TA and a Gram Schmidt method on each iteration of the Lanczos algorithm to get a tridiagonal form. After running the Lanczos algorithm, the ALPHAS, BETAS, and V tables are created. The ALPHAS table contains the diagonal elements and the BETAS table, the subdiagonal elements of the tridiagonal matrix resulting from the Lanczos algorithm. Table V contains the Lanczos vectors.

③ The QL algorithm with implicit shifts uses values from the ALPHAS and BETAS tables to generate eigenvalues and eigenvectors for the tridiagonal matrix. When the QL algorithm with implicit shifts is initiated, the D table contains values copied from the ALPHAS table, and the E table contains values copied from the BETAS table. After the algorithm finishes, the values from the D table are eigenvalues, which are later copied to the ALPHAS table. The values from the ALPHAS table are copied to the EIGENVALUES table although the values for both tables are identical. Since the values from the E table become 0, they are not copied back to the BETAS table. The S table contains eigenvectors; and since the eigenvectors for the ATA matrix are needed, the values from the S and V tables are multiplied and copied to the EIGENVECTORS table. Thus, eigenvectors are stored in the EIGENVECTORS table and, eigenvalues, in the EIGENVALUES table. To calculate SVD, the values from the EIGENVECTORS table are rearranged to get V^T. Table V contains the arranged values. Singular values in S can be produced by rearranging the eigenvalues from the EIGENVALUES table in descending

order. If desired, U can be calculated using the values from the ATA, V, and EIGENVALUES tables.

4.6 Extract a Sentence for Each Topic for the Summary

The columns of the V^T matrix represent the sentences of the input matrix and its rows represent the concepts that are obtained through the calculation of the SVD.

Each entry in the V^T matrix gives information about how closely the sentence is related to the given concept. A higher value means that the sentence is more closely related this concept. Thus, the sentence that is most related to each concept is chosen for the summary until a predefined number of sentences is extracted.

	Sentence 1	Sentence 2	Sentence 3	Sentence 4
Topic 1	0.22	1.51	2.11	7.67
Topic 2	5.33	3.22	1.10	2.43
Topic 3	3.43	5.34	0.74	0.71
Topic 4	2.11	1.31	9.54	2.33

Table 1: Representation of the transpose of matrix V

As shown in the figure above, the first topic is chosen, and after that its most closely related sentence. Next, the second topic is chosen, and the process is repeated until a predefined number of sentences is extracted. Thus, sentence 4 for topic 1, sentence 1 for topic 2, sentence 2 for topic 3, and sentence 3 for topic 4 are chosen.

The steps below illustrate how sentences are selected for inclusion in the summary.

① Retrieve records from the VAL column in the EIGENVALUE tables in descending order and store the result in array A, as shown below.

```
$statement = "SELECT * FROM EIGENVALUES ORDER BY ABS(val) DESC LIMIT :max";
$query = $pdo->prepare($statement);
$query->bindParam(':max',$max,PDO::PARAM_INT);
$query->execute();
while($row = $query->fetch (PDO::FETCH_BOTH))
{
        $iarr[] = $row[0];
}
```

Figure 9: Sample code for retrieving values from the VAL column in the EIGENVALUE tables

② Using the data obtained in step 1, iterate through the array. Retrieve an index with the largest value for each row, and store the result in array B until the desired number of sentences is reached. The figure below illustrates step 2.

```
foreach($iarr as $ia)
{
        $statement = "SELECT ABS(val) FROM V WHERE i = :ia ORDER BY val DESC limit :max";
        $query = $pdo->prepare($statement);
        $query->bindParam(':ia',$ia,PDO::PARAM_INT);
        $query->bindParam(':max',$max,PDO::PARAM_INT);
        $query->execute();
        $val = $query->fetch(PDO::FETCH_NUM);

        for($i=0; $i<$max; $i++)
        {
                $statement = "SELECT j FROM V WHERE ABS(val) = :val and i = :ia";
                $query = $pdo->prepare($statement);
                $query->bindParam(':val',$val[0],PDO::PARAM_INT);
                $query->bindParam(':ia',$ia,PDO::PARAM_INT);
                $query->execute();
                $row = $query->fetch(PDO::FETCH_NUM);
                $cand[] = $row[0];
        }

}
```

Figure 10: Sample code for retrieving an index with the largest value for each row

③ Sort array B so the order of the sentences can be preserved, as seen in the figure below.

```
asort($cand);
```

Figure 11: Sample code for sorting array

Depending on the number K that the user desires, the most important K topics are chosen, as well as a sentence for each. For example, in the figure above, sentence 4 for topic 1, sentence 1 for topic 2, sentence 2 for topic 3, and sentence 3 for topic 4 are selected.

However, the use of this method results in the loss of the order in which the sentences appear in the original text. Since we want to preserve this order, array B must be sorted.

④ Get sentences that have the same indexes in array B and store them in the SUMMARY table. The figure and the code below illustrate how this is done.

```
foreach($cand as $c)
{
        $val = $sentences[$c];
        if(!in_array($val,$summary))
        {
                $summary[] = $val;
        }
}

$content= '';
for ($i=0; $i< count($summary); $i++)
{
        $content .= $summary[$i]." ";
}

$statement = "INSERT INTO summary(line,content) VALUES(:num,:content)";
$query->bindParam(':num',$num,PDO::PARAM_INT);
$query->bindParam(':content',$content,PDO::PARAM_STR);
$query->execute();
```

Figure 12: Sample code for getting summary

The SENTENCE table contains sentences that are arranged in an orderly manner and that are easily recognized by the column value I, as seen in the following figure.

i	j	sentence
0	0	Hooray! Ive been given a LiebsterAward.
1	0	Thank you Darian Wilks, the Crazy Lady with a Pen,...
2	0	Darian is also the author of Love Unfinished.So he...
3	0	Drum roll please.1.
4	0	Adoptsomom, Necesstiy is the Mother of Invention.
5	0	Adoptsomomhosts a blog about family life, life as ...
6	0	The two of us have known each other since we were ...
7	0	She inspired me to start my own blogand has offere...
8	0	Her blog post, Blueberry trifecta: labor day weeke...
9	0	Diane Doherty, Reading is Sexy.
10	0	Diane is a new blogger friend.
11	0	She hosts a blog about books, literacy, and family...
12	0	Danica Page, Taking it one page at a time.
13	0	Danica hosts a book review blog.

Figure 13: Representation of the SENTENCE table

Thus, if array B contains indexes such as 1, 2, 3, and 4, then sentences with the same IDs are retrieved from the database and stored in the SUMMARY table. From the beginning of sentence 1 to the end of the original text, the above steps are repeated. In

the end, from sentence 1 to the maximum length of the sentence in the original text, the sentences are precomputed and stored in the SUMMARY table.

line	content
1	At this point, only two marketing techniques have ...
2	On the contrary, I found this hilarious. At this p...
3	On the contrary, I found this hilarious. At this p...
4	On the contrary, I found this hilarious. At this p...
5	On the contrary, I found this hilarious. At this p...
6	On the contrary, I found this hilarious. At this p...
7	However, I have to wonder if, by giving away so ma...
8	However, I have to wonder if, by giving away so ma...
9	However, I have to wonder if, by giving away so ma...
10	However, I have to wonder if, by giving away so ma...
11	However, I have to wonder if, by giving away so ma...
12	However, I have to wonder if, by giving away so ma...
13	However, I have to wonder if, by giving away so ma...
14	However, I have to wonder if, by giving away so ma...
15	However, I have to wonder if, by giving away so ma...

Figure 14: Representation of the SUMMARY table

4.7 Back-End Functions and Interfaces

All back-end functions are implemented in PHP, including adding blogs, removing blogs, displaying the summary for a blog, displaying individual posts from selected blogs, shortening long URLs, and posting to Twitter.

Since Twitter allows only 140 characters per post, shortening long URLs was needed to save space. I chose one of the most popular URL-shortening services, Bit.Ly API (Application Programming Interface), for this purpose.

4.8 User Interface Functions and Front-End Layer

Ajax requests were made using JQuery to provide a pleasant experience for the user. The code snippet below shows the use of $.ajax from the JQuery library to make a request to the server.

```
$.ajax(
{url:"requestBlog.php",
type: "GET",
data:   {blogUrl: blogUrlVal},
beforeSend: function () {
    $('#loading').css("display","block");

},

success: function(data){
                $("#loading").css("display","none");
                var parts = parseURL(blogUrlVal);
                $("#blog").append($("<option />").attr("value", parts.host).text(parts.host));
                alert("Successfully added your request!");
                disablePopup2();
            },
error: function(data){
            $("#loading").css("display","none");
            alert("Sorry error occured while getting your request!");
            disablePopup2();
        }
});
```

Figure 15: Sample code for using JQuery library to make a request to the server

The user interface is presented in the picture below. Each number indicates an area where the user can interact with the application or where a response from the application is displayed.

Figure 16: The user interface

① The Preferences options will let users save API keys such as the passwords required to post to their Twitter account, or clear existing API keys. This will eliminate the need for them to type their passwords every time they wish to post to Twitter.

② The slider will control the length of the summary to be displayed. Sliding to the far left will display only one sentence from the summary, while sliding to the far right will display all the sentences from the original text. Using Ajax, we can send the value of the slider and the name of the individual post or blog for which the summary is requested. Since the entire precomputed summary already has been stored in the SUMMARY table, obtaining it is very easy. The code snippet below shows how to get the summary, which depend upon the value of the slider and the name of the blog or post.

```
change: function( event, ui ) {
    $('#content').children().each(function() {
        var divId = $(this).attr('id');
        var me = $(this);
        $.ajax({
            url: "getSummary.php",
            type: "GET",
            data:    { blog: divId, sliderVal:ui.value},
            dataType: "json",
            beforeSend: function () {
                $('#loading').css("display","block");
                me.find('.blogentry').html("Loading...");

            },
            success: function(data){
                $("#loading").css("display","none");
                me.find('.blogentry').html(data.summary);

            },
            error: function(data){
                me.find('.blogentry').html("Error occured....");
                }
        });
```

Figure 17: Sample code for retrieving summary by using the slider control

③ The “Add Blog” link will let the user request that a blog be summarized.

④ The “Remove Blog” link will allow a user to employ a drop-down list box to delete an existing blog.

⑤ The drop-down list box lets the user select a blog from the list. Once this is done, he or she can click the Get This Blog’s Summary button, the “Individual Posts” button, or both.

⑥ The “Get This Blog’s Summary” button will retrieve the summary for the selected blog.

⑦ The “Show Individual Posts” button will display all posts from the selected blog.

⑧ The summary will be displayed in this area. If the area is clicked, a pop-up window will appear that will allow the user post to Twitter. See the figure below for an example.

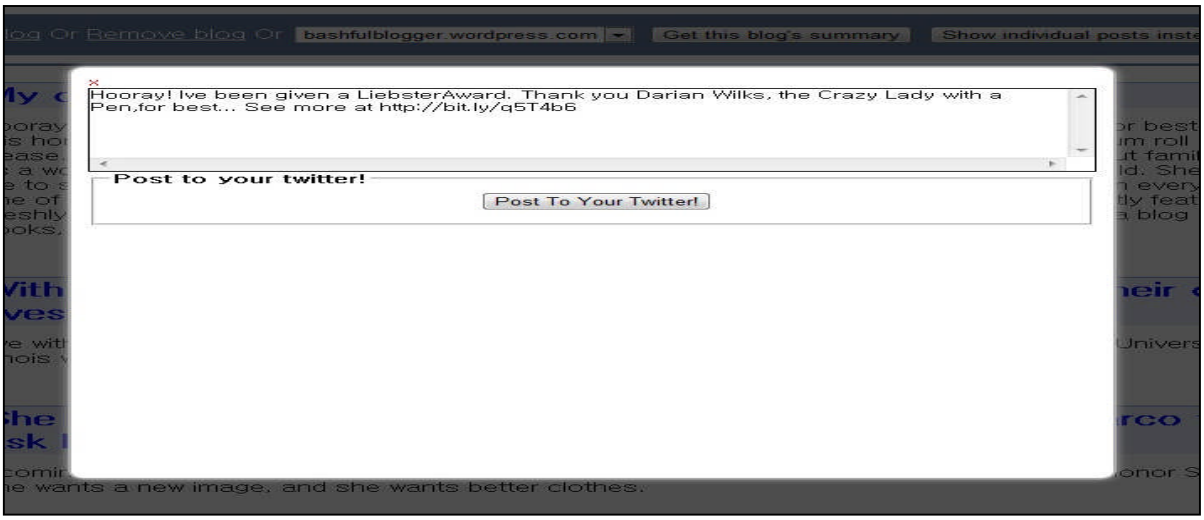


Figure 18: Pop-up window for posting to Twitter

⑨ After clicking the “Post to Your Twitter” button, either a success or failure message will appear. If the posting is successful, a new message will be displayed on the user’s blog, as shown in the figure below.

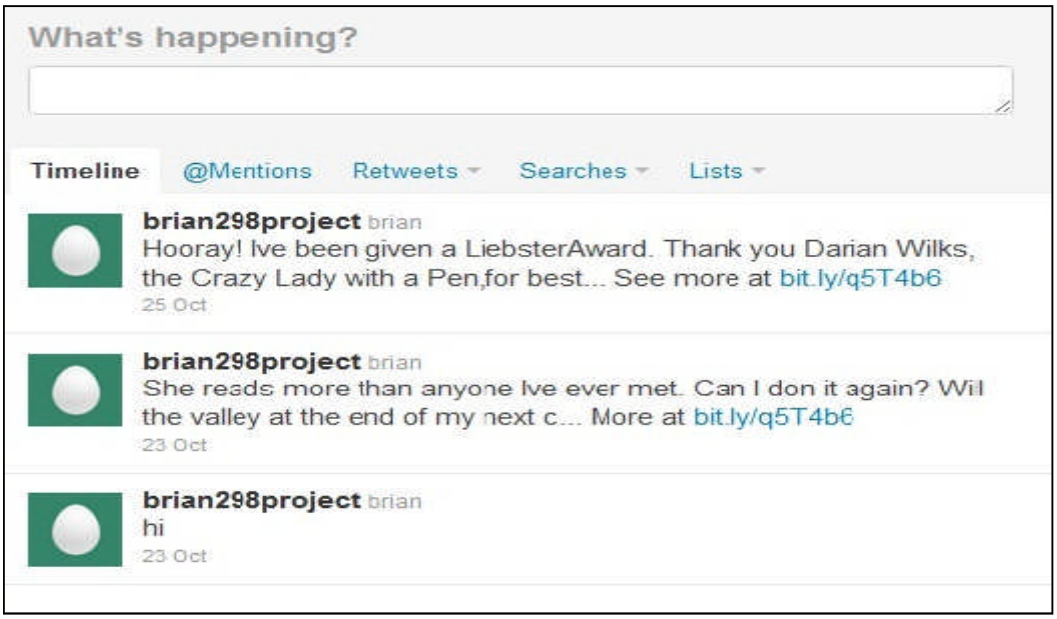


Figure 19: Twitter

4.9 Summary Evaluation

Evaluation methods attempt to determine how accurately an automatically generated summary captures the meaning of its original source. Generally, two forms of evaluation are used.

Firstly, the quality of an automatically generated summary can be measured by directly comparing it to a human-made counterpart. Users can rate the generated summary on items such as whether it contains the main ideas of the text and its fluency, or they can compare it with ideal summaries created by other contributors. However, since different users may choose different sentences to be included in the summary and the process is very time consuming, this is not a satisfactory method of evaluation.

Another approach is to use automatic summarization evaluation tools such as ROUGE, which is based on the n-gram co-occurrence, the longest common subsequence, and the weighted longest common subsequence between the ideal and the extracted summaries.

ROUGE was developed by Chin-Yew Li and delivered to the research community in 2004. By 2010, more than 150 research sites worldwide had downloaded this software package. ROUGE was chosen as the official automatic evaluation tool by the Document Understanding Conference in 2004, 2005, and 2006, and the U.S. government has sponsored summarization evaluation efforts using this tool.

Currently, ROUGE (v1.5.5) includes the following automatic evaluation methods.

A. ROUGE-N: N-gram-based co-occurrence statistics
B. ROUGE-L: LCS-based statistics
C. ROUGE-W: Weighted LCS-based statistics that favor contiguous common subsequences.
D. ROUGE-S: Skip-bigram-based co-occurrence statistics
E. ROUGE-SU: Skip-bigram plus unigram-based co-occurrence Statistics

An n-gram can be defined as one or more consecutive words. The ROUGE scores tell us the n-gram overlap between the source text and the model summary. To apply LCS in our evaluation, we view a summary sentence as a sequence of words. The intuition is that the longer the LCS of two summary sentences is, the more similar are the two summaries. In the ROUGE settings, we have used the Porter stemmer to reduce the words to their root forms in both the computed and the model summaries. A high ROUGE score indicates the greater relevance of the automatically generated summary. Its upper boundary is 1 and its lower boundary is zero. Although we obtained all ROUGE results (ROUGE-1, ROUGE-2, ROUGE-3, ROUGE-W and ROUGE-L) in our evaluation, we report only the ROUGE-1 results and the 1-gram matching between the automatic and model summaries in this paper, which is shown to correlate with human judgment (Lin & Hovy, 2003).

4.9.1 Preparation

To assess a summary using the ROUGE evaluation toolkit, the correct directories and formats must be set up beforehand. There are two types of summaries. One is the system-generated summary, also known as the candidate summary, and the other is the gold standard summary, also known as the model summary. The model summary is usually written by humans. Once the proper directories are set up and the files are in the right format, we can run commands such as the one given below. Depending on one's preference, the options may differ.

./ROUGE-1.5.5.pl -e data -u < project-name>/settings.xml.

4.9.2 Test Plan and Datasets

I used a publicly available article from Yahoo News about a battle between Samsung Electronics and Apple over Samsung's attempt to get a preliminary injunction against the iPhone 4S in Paris, France, based on an alleged infringement of its 3G patents [11]. I picked a news article over a blog post on the assumption that writing by a professional

journalist would be more detailed and concise. Additionally, a news article deals with current events of public interest, while a blog post usually deals with personal and private stories. I prepared five summaries written by five different individuals. They were asked to pick the one and five most important sentences from the original text. Four summaries by person A, person B, person C and person D are used as model summaries. Model or gold standard summaries are considered ideal examples and are generally created by humans. The candidate summary is then measured against the model summaries.

I will compare a summary generated by my application with three baselines: a summary by a human (person E); a summary generated by MEAD [12], a well-known open-source summarizer; and a selection of one and five random sentences from the text. I will use the ROUGE-1 scores as the measure of the summary's quality.

4.9.3 Results

Once the preparations were finished, the ROUGE toolkit was run to evaluate the summaries and assess metrics such as recall, precision, and FScore. "Precision" is a metric that determines how closely the candidate summary mirrors the model summary. Thus, a higher precision score indicates that the candidate summary accurately reflects more of the model summary in a smaller number of sentences. "Recall" measures how well the candidate summary echoes the corpus or original text. A summary that returns all available sentences would give a very high recall score, but a very low precision score. Precision and recall are often combined to form an aggregate score, called the "FScore," which is the harmonic mean of both.

The table below indicates the ROUGE-1 scores obtained from running the ROUGE evaluation toolkit to compare the most important sentence chosen by the three model summaries and the summary generated by my application.

Method	Average Recall	Average Precision	Average FScore
My Application	0.39394	0.34821	0.36967
Random	0.01010	0.03125	0.01527
MEAD	0.27273	0.35526	0.30857
Person E	0.46465	0.35938	0.40529

Table 2: ROUGE-1 scores for the most important sentence

As was expected, for the most important sentence, person E performed the best, and the random method performed the worst. My application performed better than did MEAD.

The figure below indicates the ROUGE-1 scores obtained from running the ROUGE evaluation toolkit to compare the five most important sentences selected by the three model summaries and the summary generated by my application.

System	Average Recall	Average Precision	Average FScore
My Application	0.78571	0.72600	0.75468
Random	0.26623	0.35756	0.30521
MEAD	0.58874	0.57627	0.58244
Person E	0.74892	0.73305	0.74090

Table 3: ROUGE-1 scores for the five most important sentences

Surprisingly, my application performed the best in this example. As expected, the random method performed the worst. The results show that the recall score improves and the precision does not drop with increasing sentences. This shows that my application can generate summaries of different lengths while maintaining quality.

Overall, my application constantly performed better than did MEAD and a random selection of sentences, which shows that it is possible to generate a meaningful summary by determining and selecting sentences that best represent an underlying concept or topic.

5. Conclusion

In this paper, we have explored some relevant linear algebra concepts, attempted to automatically summarize text using the Lanczos algorithm and SVD, and examined the generated summary using the ROUGE evaluation toolkit. To speed up the processing time for calculating SVD, we also took advantage of parallelism, which is available to most current data management systems. However, to calculate SVD in parallel, the following requirements should be met. First, a computer with multiple CPUs is required. Secondly, the operating system must be able to manage multiple processors and to share common resources (memory, I/O, and system bus) among multiple processors. Third, some initialization parameters must be properly set for different database management systems. Lastly, small modifications to SQL statements are required, such as employing hints or using a PARALLEL clause at the statement or object-definition levels.

The results show that my application extracted sentences reasonably well from the original text. However, it has two disadvantages. I chose to select a sentence with the largest value in each row of V^{T} for each topic, assuming these sentences to be important. Other sentences might have been equally important, but were not selected as a consequence of their lower value. Therefore, sentence selection should be further refined. Also, because my application must simultaneously execute multiple SQL statements in the database through parallel processing, a more powerful computer is needed to analyze its performance quickly and efficiently to see whether these results apply across larger data sets.

References

[1] C.C. Paiget. *Error Analysis of the Lanczos Algorithm for Tridiagonalizing a Symmetric Matrix*. Montreal: McGill School of Computer Science, 1975.

[2] Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol. I: Theory*. SIAM, 2002.

[3] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1998.

[4] H.J. Bowdler, R.S. Martin, C. Reinsch, and J.H. Wilkinson. *Handbook for Auto. Comp., Vol. ii: Linear Algebra*. 1971, pp. 227-240.

[5] "Singular Value Decomposition." Wikipedia: http://en.wikipedia.org/wiki/Singular_value_decomposition [Nov. 15, 2011].

[6] Gene H. Golub and C. Reinsch. "Singular value decomposition and least squares solutions," *Numerical Mathematics,* vol. 14, pp. 403-420, 1970.

[7] William H. Press. *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press, 1998.

[8] G.H. Golub and C.F. Van Loan. *Matrix Computations,* 2nd ed. Baltimore: Johns Hopkins University Press, 1989.

[9] Jean-Pierre Dijcks, Hermann Baer, and Maria Colgan. "An Oracle White Paper: Oracle Database Parallel Execution Fundamentals." Oracle Corporation, 2010.

[10] “Cross-Document Structure Theory Corpus.” University of Michigan: http://tangra.si.umich.edu/clair/CSTBank/phase1.htm [Nov. 15, 2011].

[11] T. Worstall. “Apple Samsung Patent Battle: The French Round.” Yahoo! News: http://news.yahoo.com/apple-samsung-patent-battle-french-round-153155468.html, Nov. 18, 2011 [Nov. 20, 2011].

[12] MEAD. Online: http://www.summarization.com/mead/ [Nov. 17, 2011].

[13] Phil Sneiderman, Johns Hopkins Unversity. Online: http://releases.jhu.edu/2011/07/06/tracking-public-health-trends-from-twitter-messages/ [July, 6, 2011]