Text Retrieval Using Linear Algebra

An Undergraduate Honors Thesis

by Joshua L. Drew

Dr. James Baglama (University of Rhode Island)
Thesis Advisor

Interactive component and source code available:
http://joshdrew.com/thesis/

Ball State University
Muncie, Indiana

December 2002
(Graduation date: 2002-12-15)

Table of contents

page

Abstract

Text retrieval is an important area of research. As information and methods of its storage have proliferated, the need to have efficient methods of locating subsets of this information has increased as well.

The Internet is serving to catapult the size of present-day text collections past those of only fifty years ago. Accordingly, Internet search engines are a hotbed for information retrieval research.

A widely-researched text searching method involves modeling a text collection in a term-by-document matrix, and evaluating the documents' relevance to a query with simple linear algebra.

This document presents one such system, the possibilities for future research incorporating ideas of that system, and computer code with a Web interface, written in C++ and Perl, that implements the Web search engine

Key words

Information retrieval, text retrieval, search engine, linear algebra, vector spaces, singular value decomposition, latent semantic indexing, C programming, C++ programming, Perl programming, Internet.

Acknowledgements

Dr. James Baglama, University of Rhode Island

Scott Hinkley

In the Spring of 2000, Dr. Baglama, then a mathematics professor at Ball State University, sought student volunteers for a linear algebra-related research project. Scott Hinkley and I expressed our interest. Scott, at the time, was pursuing his master's degree in computer science, at Ball State. I had just completed my third year in Ball State's undergraduate computer science program.

Over the Summer, Dr. Baglama revealed the project's specific topic: Internet search engines. When the Fall 2001 semester began, our project got underway as well.

Ball State's computer science server (bsu-cs) was our development area. We met weekly, to discuss goals and progress. Scott researched options for indexing Web pages served by bsu-cs, and contributed his objectivity during term-weighting discussions. Dr. Baglama and I wrote Perl scripts to create the index. I supplemented those scripts with programs to create the term-by-document matrix, evaluate document connectedness, and provide a basic Web interface. Meanwhile, Dr. Baglama wrote Matlab programs to compute the singular value decomposition of the term-by-document matrix, find the angle between a search vector and each column (document) of the aforementioned matrix, and provide a user-friendly interface to the fledgling system. Our work culminated in presentations as part of the Ball State Math Department's undergraduate colloquium series, on March 21, 2002, and at the 7th Annual Student Symposium at Ball State on March 26, 2002.

During the Summer of 2002, Dr. Baglama and I met to form a plan for our research project becoming my honors thesis. We decided that I would evaluate the state of the source code that had been developed. Then, I would convert the Matlab code to C++, and use it as the basis for a finished, usable, Web search engine, restricted the bsu-cs server.

Shortly after our Summer meeting, Dr. Baglama left Ball State for a position at the University of Rhode Island. He deserves my sincere gratitude for continuing in his role as thesis advisor.

Thank you, Dr. Baglama. Scott, thank you as well.

Introduction

Text retrieval, the Internet, and search engines

Over the past six centuries, the volume of written resources has grown exponentially. The timeline in Figure 1 shows the increase in size of Western text collections, primarily over the past two millennia.

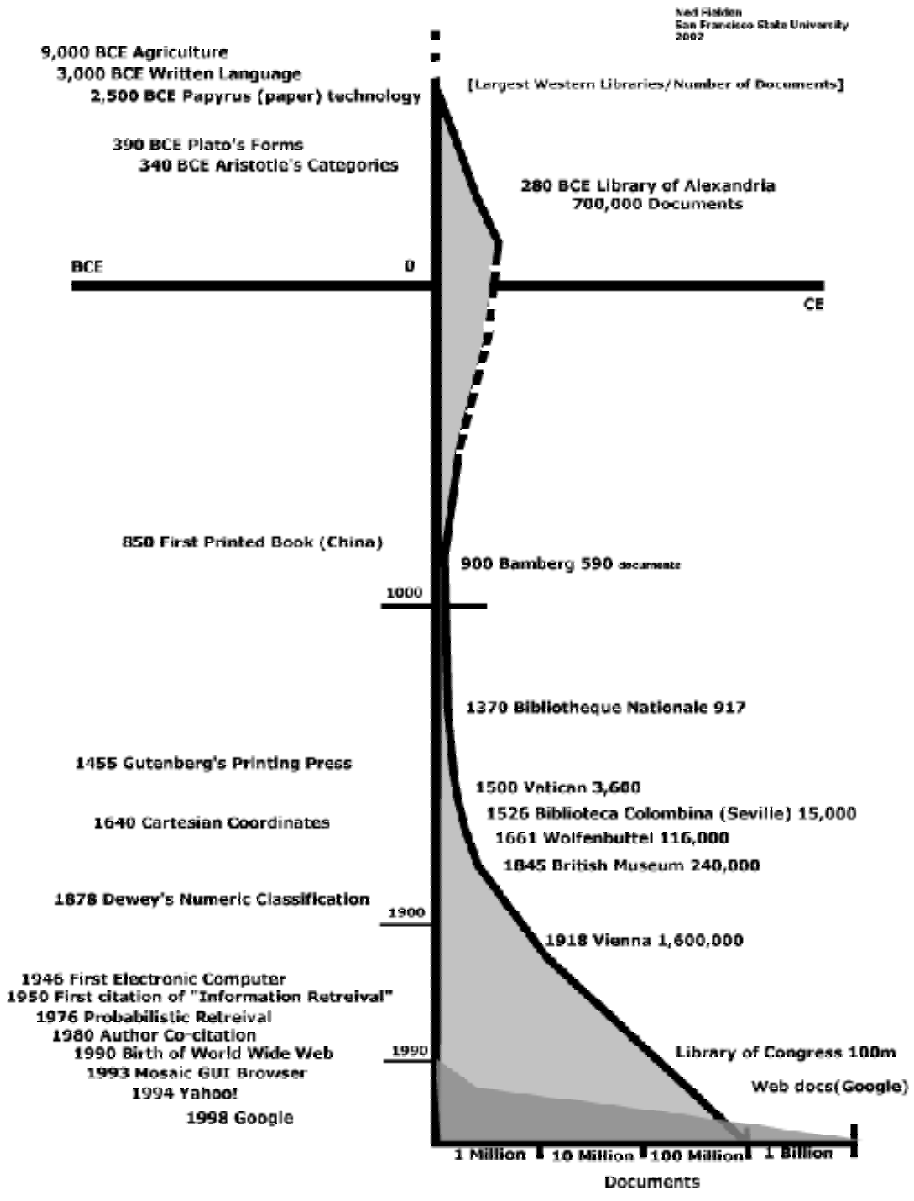

Figure 1: Increase of information over time: represented by quantity of information stored in Western libraries (with the exception of Alexandria); source: Ned Fielden, San Francisco State University.[1]

[1] Presented at Asilomar Monterey California Biennial California Academic & Research Librarians (CARL) Conference, May 10 2002; http://online.sfsu.edu/~fielden/historytime.htm

Such vast bodies of work require effective retrieval methods to ensure their utility. The advent of large, Internet-based text collections in the form of World Wide Web pages and newsgroup posts has increased the urgency of that need greatly.

Many researchers have sought to develop an information retrieval solution suitable for use on the Internet. Some details about major search engines are presented below:

1990: Archie

Before the advent of the World Wide Web, Alan Emtage, Peter Deutsch, and Bill Wheelan created Archie to index anonymous FTP sites. If a user knew the name of a file they intended to download, they could use Archie to find a FTP site where it was offered. Archie used a data gatherer, and a regular expression matcher for retrieving file names matching users' queries [17].

1993: Robots

The World Wide Web Wanderer, created by Matthew Gray, was the first robot on the Web. Archie used a similar program to automatically build its index, but that was via FTP. Several other Web-indexing robots were also created in 1993 [5].

1993: Excite

Six Stanford undergraduates built the Excite search engine to utilize word relationships in order to provide query results [5].

1994: Yahoo!

Originally a hand-maintained directory of hierarchically categorized Web content, Yahoo! is now more automatically maintained. In its formative years, Yahoo! experienced a great deal of success due to the inadequacy of robot-built indices [11].

1994: Lycos

Lycos, having been developed by Michael Mauldin at Carnegie Mellon, was launched in July of 1994. Features key to its success included: ranked relevance retrieval, stemming [13, 15], and word proximity matching. Lycos' index was also larger than that of any other search engine. At launch, Lycos had cataloged 54,000 documents. In January 1995, the number had increased to 1.5 million. By November 1996, Lycos had indexed over 60 million documents [4].

1995: AltaVista

Digital Equipment Corporation (DEC) developed AltaVista, and included a number of substantial innovations [5]:

- Natural language queries allowed users to ask questions.
- Advanced searching techniques allowed users the luxury of Boolean operators.
- Newsgroup indexing allowed such content to be searched.
- An open database allowed users to add and delete their own sites.
- Link indexing allowed users to search for all pages linked to a given document.

1997: Google

Google, founded by Sergey Brin and Larry Page, is now established as the leading Internet search engine. As of the writing of this document, Google has indexed 3,083,324,652 Web pages. The system uses a combination of linear algebra, probability, and word and document relationship analysis, computed on a cluster of over 10,000 computers to provide search results [10].

Today, thousands of search engines are available on the Internet. While Google is comfortably in the lead, other groups continue to foster innovation as well. Current areas of interest include ontology (knowledge representation), belief networks, inference networks, the Naive Bayes learning algorithm, and document clustering [12].

Structure of this document

This document describes the ideas and decisions behind the development of a simple, linear algebra-based Internet search engine.

First, key concepts are discussed. With an understanding of these ideas, anyone should be able to see the possibilities for using linear algebra in text retrieval.

Next, important choices made during implementation are explained. The rationale behind these choices may lend direction to those interested in building upon this work.

Finally, several possibilities for future research are listed, along with brief explanations.

The references section of this document contains, in addition to those works cited directly, information about resources that served as inspiration. These un-cited publications contain enough technical details to satisfy the most inquisitive of readers.

The appendices contain details concerning the implementation of this text retrieval system.

A Web search that implements the ideas presented in this thesis, for the bsu-cs server, can be found at http://joshdrew.com/thesis/. In addition, this document and the associated source code are available.

Discussion

Concepts that form the foundation of this project

A large collection of documents can be represented as a term-by-document matrix

A matrix *A* can be constructed such that the frequency of each term *i* in each document *j*, is stored at A_{ij}. To build this matrix for bsu-cs, Dr. Baglama and I wrote a simple index-creator[2]. Beginning with a list of known Web pages, the script analyzed each document encountered. Word frequencies were recorded, and links to other bsu-cs Web pages were extracted. Those links were both added to the page list, and tabulated as possible indicators of document importance (more links to page *x* implies that page *x* is more important). Note that common words such as "the" and "how" have little intrinsic meaning [1]. Therefore, we created a list[3] of "stop words" which were excluded by the matrix-forming code. After several hours of parsing, the index-creator, in conjunction with a script[4] to organize each page's raw data, produced a 26257 x 3557 sparse matrix. The pattern of that data, represented in a matrix, can be seen in Figure 2.

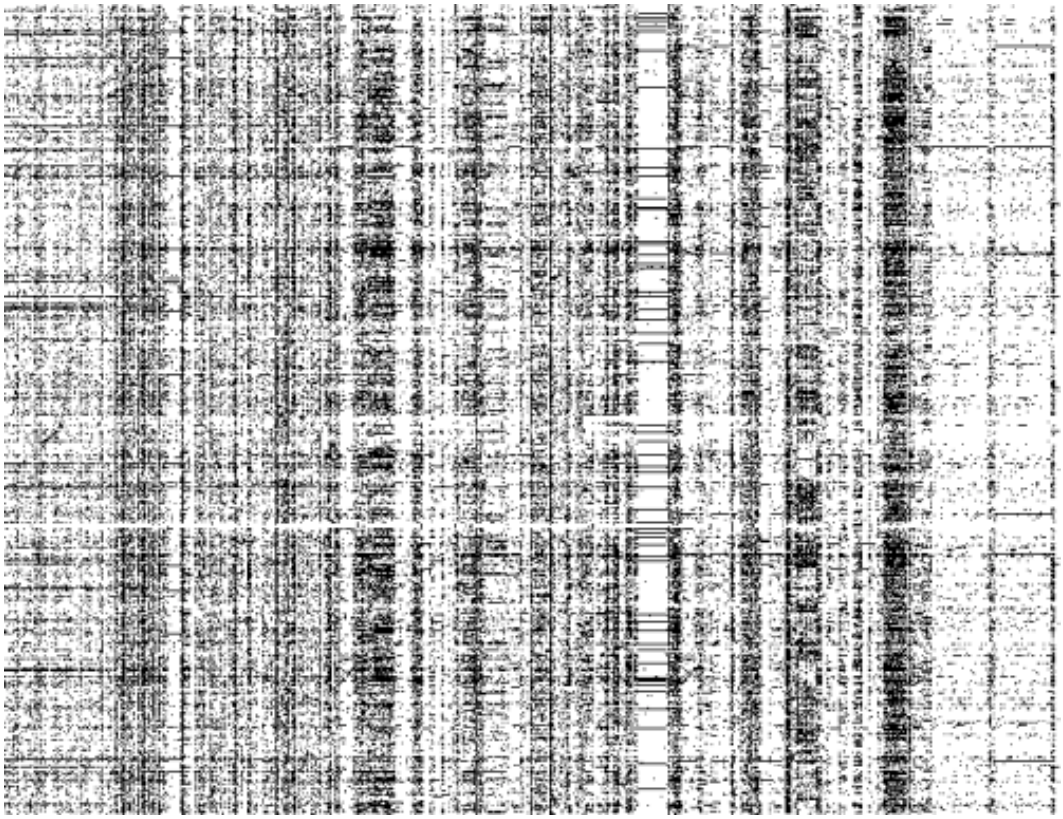

Figure 2: A graphical representation of the 26257 x 3557 bsu-cs term-by-document matrix.

[2] The source code for this script, "getpage", can be found on the project Web site.
[3] The list of stop words used in this project can be found on the project Web site.
[4] The source code for this script, "parse_wordlist", can be found on the project Web site.

A key word query can be compared to each column of a term-by-document matrix

If a user supplies a list of key words, the relevance of those words to each of the documents in the term-by-document matrix $A=[a_1, a_2, \ldots, a_n]$ can be determined. A common measure of such is the cosine of the angle between the query vector, q and each document (column) vector, a_j [3]. If the query is represented by a vector q and the term-by-document matrix as A, the cosine of the angle between q and a document a_j is found by:

$$\cos \theta_j = (a_j{}^T * q) / (\| a_j \|_2 * \| q \|_2) \quad j=1, \ldots, n$$

where the Euclidean vector norm $\| x \|_2$ is defined by $\| x \|_2 = (x^T x)^{1/2}$, the square root of the dot product of x with itself [3].

The results of this computation can be stored for all documents, $j = 1,\ldots,n$. If the cosine of the angle between two vectors is one, the vectors are parallel. However, if the cosine of the angle is zero, the vectors are orthogonal [14]. Therefore, a cosine closer to one implies that a document, a_j, is relevant to a user's search vector, q. In general, documents not exceeding some cosine threshold, which is determined experimentally, are returned to the user [7]. The C++ program[5], "search", uses this idea, and a threshold of 0.132, to provide users with query results.

Rank-reduction improves the term-by-document matrix

The rank of a matrix is the dimension of the column space of that matrix [14].]. For example, the term-by-document matrix in Figure 1, has rank ≤ 3557. So, for the term-by-document matrix A, a rank-reduction involves removing unnecessary documents, e.g. a Web page mirror. These are eliminated with the consequence of improved search results.

This project uses a truncated singular value decomposition (SVD) [9] to approximate A.

Let A be an m by n matrix, then the truncated SVD is given as

$$AU = VS$$

where U is an n by k orthogonal matrix ($U^T U = I$), V is an n by k orthogonal matrix, and S is an k by k diagonal matrix. For a small value of k we have $A \approx VSU^T$.

Several benefits arise from this usage:

- Noise and uncertainty, present in all large databases, are reduced [1].

[5] The source code for "search" can be found on the project Web site.

- Word usage is estimated across documents [2]. This helps to compensate for polysemy, the situation in which a word has multiple meanings.
- Queries may return relevant documents containing none of the user's search terms. The SVD models a latent semantic structure assumed to be present in the document collection [2].
- The speed with which calculations can be performed is increased because *A* is replaced by a SVD approximation.

Choices made and why

Use C++ for matrix operations

Several factors contributed to the decision to use C++ for performing matrix operations, instead of using another language like C, Java, or Perl:

- The C++ vector and string libraries eliminated many possible programming mistakes by handling pointer manipulation automatically.
- The ability to compile an executable binary of the search-performing code quickly eliminated Java, Perl, and other interpreted programming languages from contention.
- The C, C++, and Fortran programming languages are widely accepted as excellent environments for producing linear algebra programs.
- The C and C++ programming languages are the first choice in the Ball State University computer science department. As such, tools and knowledge are easily located.

Use Perl for indexing

Choosing to use Perl for building the term-by-document matrix was easy. The mature text-processing capabilities are well-suited for parsing thousands of documents. In addition, the Comprehensive Perl Archive Network[6] (CPAN) provides easily accessible documentation for Perl modules that were used.

Refrain from the use of complicated weighting schemes

Generally, linear algebra-based text retrieval systems use a combination of three weighting schemes: local term, global term, and document weighting [1]. The main ideas of these schemes are:

- Terms that appear many times in one document must be important to that document.
- Terms that are concentrated in a few documents must be descriptive terms that are present only in discussions of a certain topic.
- Terms should be assigned a relative value according to the size of the document in which they occur.

[6] http://www.cpan.org

These are all valid observations. However, after experimenting with weighting techniques, a decision was made to simplify. By experimentation, we found that too much weighting negated the benefits sought. So, only raw term frequency was used.

Possibilities for future research

Develop new weighting schemes

New weighting schemes could draw from many properties of text collections:

- location of term in document
- HTML elements, and their relation to the term
- punctuation of the term
- frequency of term in: collection, document, or section of document
- links to and from the document
- the content of linked documents
- the modification date of the document
- how often the document is selected, by a user, from query results

Increase the efficiency of the search process

The "search" program performs thousands of calculations for every query. Small improvements in the algorithms used can result in significant performance increases. Since the term-by-document matrix is sparse, there may be more efficient methods for operating on it than those provided by the C++ libraries used in this project project, such as M. Berry's SVDPACK, http://www.cs.utk.edu/~lsi/.

Use an adjacency matrix of documents to rank pages

An adjacency matrix is a representation of edges between vertices in a graph. Rows and columns are labelled with graph vertices. For each element of the adjacency matrix M, the value, m_{ij}, is one if there is an edge between the vertices i and j [19]. A collection of Web pages can be modeled in this way[7].

An interesting application is that the number of k-step sequences between vertex i and j in a graph with adjacency matrix M is the (i,j) entry in M^k. So, the number of links separating two documents can be determined. This information could, then, be used to augment a weighting scheme.

Modularize the server-side processes

[7] http://slashdot.org/articles/02/03/07/201259.shtml?tid=134

Loading a large matrix from disk for every query is extremely resource-intensive. Currently, the search program performs this and other expensive operations during every execution. A straightforward improvement to the system would involve separating the C++ code into two programs: a client, and a daemon. The two applications would communicate via sockets.

The daemon would prepare the collection index and perform calculations:

1. Load term-by-document matrix, term vector, and document vector.
2. Perform SVD of term-by-document matrix.
3. Await signal from client.

Upon reception of a user's query from the search client, the daemon would:

1. Form a query vector from the search terms.
2. Compare the query vector with each document vector.
3. Return a list of relevant documents to the client

The client would simply receive users' search terms from the interface, and send the appropriate signal to the sleeping (resident in memory) search daemon. Then, it would wait to receive the search results.

Conclusion

Advanced information retrieval methods are crucial to making today's large text collections useful. This project establishes a foundation on which students at the undergraduate or graduate level can explore new possibilities. The final results of this project produced a working search engine for the bsu-cs sever, located at http://joshdrew.com/thesis/.

References

[1] M. W. Berry, M. Browne, *Understanding search engines: Mathematical modeling and text retrieval*. Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 1999.

[2] M. W. Berry, S. T. Dumais, G. W. O'Brien, *Using linear algebra for intelligent information retrieval*. SIAM Rev., 37 (1995), pp. 573-595.

[3] M. W. Berry, Z. Drmač, E. R. Jessup, *Matrices, vector spaces, and information retrieval*. SIAM Review, Vol. 41, No. 2, pp. 335 – 362.

[4] P. Bradley, *Internet power searching: The advanced manual*. New York: Neal-Schuman Publishers Inc., 1999.

[5] M. Burke, E. Dunne, M. Power, *A history of search engines and some similar systems*. 2000-11-16. Waterford Institute of Technology. 2002-12-11. http://emhain.wit.ie/~p00csd12/firstreport/similar.htm

[6] J. A. Butler, *CyberSearch: Research techniques in the Electronic Age*. New York: the Penguin Group, 1998.

[7] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman, *Indexing by latent semantic analysis*. Journal of the American Society for Information Science, 41 (1990), pp. 391-407.

[8] S. Dumais, *Improving the retrieval of information from external sources*. Behavior Research Methods, Instruments, & Computers, 23 (1991), pp. 229-236.

[9] G. Golub, C. V. Loan, *Matrix computations*. Baltimore: Johns-Hopkins, second edition, 1989.

[10] *Google Technology Overview*. Google Inc. 2002-12-11. http://www.google.com/press/overview_tech.html

[11] C Gould, *Searching smart on the World Wide Web: Tools and techniques for getting quality results*. Berkely, California: Library Solutions Press, 1998.

[12] D. Howard, "You are here." *New Architect*. Jan. 2003: Vol 8, Issue 01, pp. 16 – 20.

[13] G. Kowalski, *Information retrieval systems: Theory and implementation*. Boston: Kluwer Academic Publishers, 1997.

[14] D. C. Lay, *Linear algebra and its applications*. Reading, Massachusetts: Addison-Wesley, second edition, 2000.

[15] J. Lovins, *Development of a stemming algorithm*. Mechanical Translation and Computational Linguistics, 11 (1968), pp. 22-31.

[16] J. Rennie, *Naïve Bayes algorithm for learning to classify text*. 1997-04-06. Carnegie Mellon University. 2002-12-11. http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/naïve-bayes.html

[17] S. Saba, *Information technology for libraries and information agencies: A short history of search engines*. Rutgers University. 2002-12-11. http://www.scils.rutgers.edu/~ssaba/550/Week05/History.html

[18] K. Sparck Jones. A statistical interpretation of term specificity and its applications in retrieval. J. Documentation, 28 (1972), pp. 11-21.

[19] E. W. Weisstein, *Adjacency matrix*. Wolfram Research. 2002-12-11. http://mathworld.wolfram.com/AdjacencyMatrix.html

Appendices

Appendix A: Relevant programming libraries

The code packages below are closely related to or used by this project. Many other, similar packages exist and should be evaluated when considering follow-up research.

GSL (GNU Scientific Library)

http://www.gnu.org/software/gsl/
language: ANSI C
version: 1.2

GSL is available under the GPL (GNU Public License). So, its source code is freely available and modifiable. The GSL provided matrix operations and data types for this project. Also, an interface to the CBLAS (C Basic Linear Algebra Subprograms - http://www.netlib.org/blas/) is provided. This proved useful for operations (dot product) not directly provided by the GSL.

LAPACK (Linear Algebra PACKage)

http://www.netlib.org/lapack/
language: Fortran 77
version: not used

There is a C version of LAPACK (CLAPACK). Neither the Fortran nor the C version of LAPACK was used in this project. The installation and documentation seemed less accessible than those of GSL. Note, however, that LAPACK is a respected linear algebra library, and, like GSL, is based upon BLAS.

Socket.pm

http://search.cpan.org/author/JHI/perl-5.8.0/ext/Socket/Socket.pm
language: Perl
version: 1.3

This Perl module was used by the index-creator (getpage), to open http connections to pages served by bsu-cs.