LATENT SEMANTIC INDEXING VIA A SEMI-DISCRETE MATRIX DECOMPOSITION

TAMARA G. KOLDA* AND DIANNE P. O'LEARY†

Abstract. With the electronic storage of documents comes the possibility of building search engines that can automatically choose documents relevant to a given set of topics. In information retrieval, we wish to match queries with relevant documents. Documents can be represented by the terms that appear within them, but literal matching of terms does not necessarily retrieve all relevant documents. There are a number of information retrieval systems based on inexact matches. Latent Semantic Indexing represents documents by approximations and tends to cluster documents on similar topics even if their term profiles are somewhat different. This approximate representation is usually accomplished using a low-rank singular value decomposition (SVD) approximation. In this paper, we use an alternate decomposition, the semi-discrete decomposition (SDD). For equal query times, the SDD does as well as the SVD and uses less than one-tenth the storage for the MEDLINE test set.

Key words. Information Retrieval, Latent Semantic Indexing, Singular Value Decomposition, Semi-Discrete Decomposition

1. Introduction.

With the electronic storage of documents comes the possibility of building search engines that can automatically choose documents relevant to a given set of topics. Information requests, or *queries*, are formatted according to the rules of the particular information retrieval (IR) system. For example, library catalogs are typically searched using a *Boolean framework* that connects key words using logical constructs such as AND, OR, and NOT [5]. Somewhat more complicated *text pattern searches* are used in systems such as the `grep` tool in UNIX [5]. Both the Boolean and text pattern search systems are based on exact matches: a search for "Samuel Clemens" would not retrieve documents that only contained the pseudonym "Mark Twain".

There are a number of information retrieval systems based on inexact matches. These systems use information about the distribution of terms among the stored documents. For instance, if many documents about Samuel Clemens also contained references to Mark Twain, then a query about "Samuel Clemens" might well produce a response including documents that refer to "Mark Twain" without mentioning Clemens. An example is the INQUERY system, which ranks documents according to the probability that they are relevant, determining the probability via an inference net [2].

* Applied Mathematics Program, University of Maryland, College Park, MD 20742. The work of this author was supported by the NSA, NPSC, and CCS. (`kolda@math.umd.edu`)

† Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. The work of this author was supported by the National Science Foundation under Grant CCR-95-03126 (`oleary@cs.umd.edu`)

The framework we are interested in here is the *vector space framework* such as that used in the SMART system [8]. In the vector space framework, indexing terms and documents are represented in a matrix – one row per term and one column per document. The (i,j)th entry in the matrix represents the importance of term i in document j. A *query* is a column vector with entries representing the importance of each term, and documents are scored for relevance by comparing the query with the corresponding column of the matrix. More details of this approach are given in Section 2.

Latent semantic indexing (LSI) is based on the assumption that exact matching of the query does not necessarily retrieve the most relevant documents. In an LSI system, only the most important features of the term-document matrix are stored, in hopes of revealing relations among documents while reducing the storage burden. Ideally, the representations give conceptual links based on the latent semantic information within the documents. The original approach to building an LSI representation, proposed by Deerwester *et al.*[3], uses a low-rank approximation derived from the *singular value decomposition* (SVD) of the term-document matrix. LSI via the SVD will be discussed further in Section 3.

The SVD has many nice theoretical properties, but we develop in this work a discrete decomposition to be used in place of the SVD. Our decomposition is far more economical in storage but equally useful for information retrieval. We introduce this decomposition in Section 4. Computational comparisons with the SVD approach are presented in Section 5.

2. Vector Space Framework. Suppose we have a collection of n documents and m indexing terms. We represent the collection as an $m \times n$ term-document matrix A. The entry a_{ij} represents the importance of term i in document j. This entry could be, for example, the number of times that the term appears in the document, although many other measures have been proposed in the literature. A query is represented as a vector $\mathbf{q}$ where the entry q_i represents the importance of term i in the query. Documents are ranked by computing the *inner product score*

$$\mathbf{s} = \mathbf{q}^T \mathbf{A}$$

and documents corresponding to the largest entries in $\mathbf{s}$ are deemed most relevant.

Although the matrix entries can be defined in many different ways, in this paper we use the definitions

$$a_{ij} = \frac{\log(f_{ij}+1)}{\sqrt{\sum_{k=1}^{m} \left(\log(f_{kj}+1)\right)^2}},$$

and

$$q_i = \chi(\hat{f}_i) \cdot \log\left(\frac{n - \sum_{j=1}^{n} \chi(f_{ij})}{\sum_{j=1}^{n} \chi(f_{ij})}\right),$$

where f_{ij} is the frequency of term i in doc j, $\hat{f}_i$ is the frequency of term i in the query and χ is the function that is one if its argument is nonzero and zero otherwise.

3. LSI via the SVD. In latent semantic indexing, we represent the document and queries in a compact representation with the hope that documents with similar concepts will appear more similar. One way to do this is using the singular value decomposition. We will briefly review the SVD and then explain its use in LSI.

The rank-k SVD approximation to a matrix is a sum of k triplets

$$\mathbf{A} \approx \mathbf{A}_k \equiv \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where the *singular values*, σ_i, are nonnegative scalars in decreasing order, and the *left and right singular vectors*, $\mathbf{u}_i$ and $\mathbf{v}_i$, each form orthonormal sets; that is, each vector has length one and is orthogonal to all other vectors in the set. In matrix form, this is written as

$$\mathbf{A} \approx \mathbf{A}_k \equiv \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T.$$

It can be shown that $\mathbf{A}_k$ is the best rank-k approximation to $\mathbf{A}$ in the Frobenius norm and in the Euclidean norm [6].

To score documents against queries, we compute the inner product between the pseudo-query and the pseudo-documents where the pseudo-query is given by

$$\tilde{\mathbf{q}} = \mathbf{U}_k^T \mathbf{q},$$

and the pseudo-document matrix is given by

$$\tilde{\mathbf{A}} = \mathbf{\Sigma}_k \mathbf{V}_k^T \mathbf{N},$$

where $\mathbf{N}$ is a diagonal matrix of inverse column norms, i.e., it has the effect of normalizing the columns of $\mathbf{\Sigma}_k \mathbf{V}_k^T$.

The SVD has been used quite effectively for information retrieval, as documented in numerous reports. We recommend the original LSI paper [3], a paper by Dumais reporting the effectiveness of the LSI approach on the TREC-3 dataset [4], and a more mathematical paper by Berry, Dumais and O'Brien [1] for further information.

4. LSI via a Semi-Discrete Decomposition. The SVD contains a lot of information, probably more than is necessary for this application. To save storage, we propose replacing the SVD by a semi-discrete decomposition.

The decomposition we propose is not new. It was introduced by O'Leary and Peleg [7] in 1983 for digital image compression. We will briefly describe the decomposition but refer the reader to [7] for more detailed information. We still write the matrix approximation as a sum of triplets,

$$\mathbf{A}_k = \sum_{i=1}^{k} d_i \mathbf{x}_i \mathbf{y}_i^T,$$

but this time the m-vector $\mathbf{x}_i$ and the n-vector $\mathbf{y}_i$ have entries taken from the set $\{-1, 0, 1\}$, while the scalar d_i can be any positive number. We write this in matrix form as

$$\mathbf{A}_k = \mathbf{X}_k \mathbf{D}_k \mathbf{Y}_k^T.$$

This decomposition does not reproduce $\mathbf{A}$ exactly, even if $k = n$, but the rank-k approximation requires only the storage of $2k(n+m)$ bits plus k scalars and is thus much more economical than the SVD. A greedy algorithm is used to construct each triplet, and convergence is monotone.

To construct the kth triplet, we do the following: Form the residual matrix $\mathbf{A}^{(c)} = \mathbf{A} - \mathbf{X}_{k-1}\mathbf{D}_{k-1}\mathbf{Y}_{k-1}^T$. (Initially, the matrices $\mathbf{X}_{k-1}$, $\mathbf{D}_{k-1}$, and $\mathbf{Y}_{k-1}^T$ are null.) We would like to choose d, $\mathbf{x}$ and $\mathbf{y}$ such that $\|\mathbf{A}^{(c)} - d\mathbf{x}\mathbf{y}^T\|_F$ is minimized. We solve this problem inexactly and iteratively. First we choose an n-vector $\mathbf{y}$ with all entries in $\{-1, 0, 1\}$. Fixing that choice of $\mathbf{y}$, we solve

$$\min_{\substack{\mathbf{x} \in \{-1,0,1\}^m \\ d \in \Re}} \|\mathbf{A}^{(c)} - d\mathbf{x}\mathbf{y}^T\|_F.$$

We can solve this problem exactly for $\mathbf{x}$ and d. We then fix $\mathbf{x}$ and solve

$$\min_{\substack{\mathbf{y} \in \{-1,0,1\}^n \\ d \in \Re}} \|\mathbf{A}^{(c)} - d\mathbf{x}\mathbf{y}^T\|_F.$$

This problem too can be solved exactly for $\mathbf{y}$ and d. We repeat this process until the change in deviation is below a given threshold. The current values of $\mathbf{x}, \mathbf{y}$ and d are added to the current decomposition to form $\mathbf{X}_k, \mathbf{D}_k$ and $\mathbf{Y}_k$.

We evaluate queries, in much the same way as we did for the SVD. We have

$$\tilde{\mathbf{A}} = \mathbf{D}_k {\mathbf{Y}_k}^T \mathbf{N}, \quad \tilde{\mathbf{q}} = {\mathbf{X}_k}^T \mathbf{q}.$$

Here, $\mathbf{N}$ normalizes the columns of $\mathbf{D}_k \mathbf{Y}_k^T$.

5. Computational Results. Information retrieval systems are compared via an *average precision* measure. To compute this measure, we assume that we have scored the set of documents with respect to a given query and that we rank the documents in decreasing order of score. Let

r_i denote the number of relevant documents among the top i documents. The *precision* for the top i documents, p_i, is then defined as

$$p_i = \frac{r_i}{i},$$

i.e., the proportion of the top i documents that are relevant.

The N-point (interpolated) *average precision* for a single query is defined as

$$\frac{1}{N} \sum_{i=0}^{N-1} \tilde{p}\left(\frac{i}{N-1}\right).$$

where

$$\tilde{p}(x) = \max_{\frac{r_i}{r_n} \geq x} p_i.$$

Typically, 11-point interpolated average precision is used. Each of our data sets has multiple queries, so we compare the mean average precision and the median average precision, expressed as percentages. In other papers, average precision generally refers to mean average precision.

We did experiments with the MEDLINE data set. Characteristics of the data set are listed in Table 5.1. The MEDLINE test set comes with a

TABLE 5.1
Characteristics of the MEDLINE Collection

Number of Documents:	1033
Number of (Indexing) Terms:	5526
File size:	0.4 MB
Avg. No. of Terms/Document:	48
Avg. No. of Documents/Term:	9
% Nonzero Entries in Matrix:	0.87
Number of Queries:	30
Avg. No of Terms/Query:	10
Avg. No. Relevant/Query:	23

document file, a query file and a relevancy judgment file. We first removed all the *stop words* (common words such as "the" or "because") from the document and query files using the stop word removal program described in [5]. Any word that appears in two different documents after stop word removal was used as an indexing term. Then we determined the entries in A and q as described in Section 4.

In Figure 5.1, we present the results of our tests. The upper right figure compares the mean average precision to query time, and the upper left graph compares the median average precision to query time. The query time is the total time required to execute all 30 queries. Observe that

TABLE 5.2
Comparison of the SDD and SVD methods on the MEDLINE data at the query time where the SDD has the highest mean average precision.

	SDD	SVD
Query Time (Sec)	2.9	3.1
Dimension (k)	120	10
Mean Avg Prec	63.2	34.9
Median Avg Prec	68.8	32.1
Decomp Storage (MB)	0.2	0.5
Decomp Time (Sec)	194.2	2.6
Rel F-Norm of Resid	0.87	0.94

the SDD method has maximal precision at a query time of 3.1 seconds, corresponding to $k = 120$, a mean average precision of 63.2 and a median average precision of 68.8. The SVD method reaches its peak at 8.4 seconds, corresponding to $k = 110$, and mean and median average precisions of 65.5 and 71.7 respectively.

A comparison of the two methods on individual queries is given in Figure 5.2. The dimensions are chosen to be the best values for each method. The performance of the SDD method is on par with the SVD method except for queries 26 and 27. We have no explanation for the SDD behavior on these two queries.

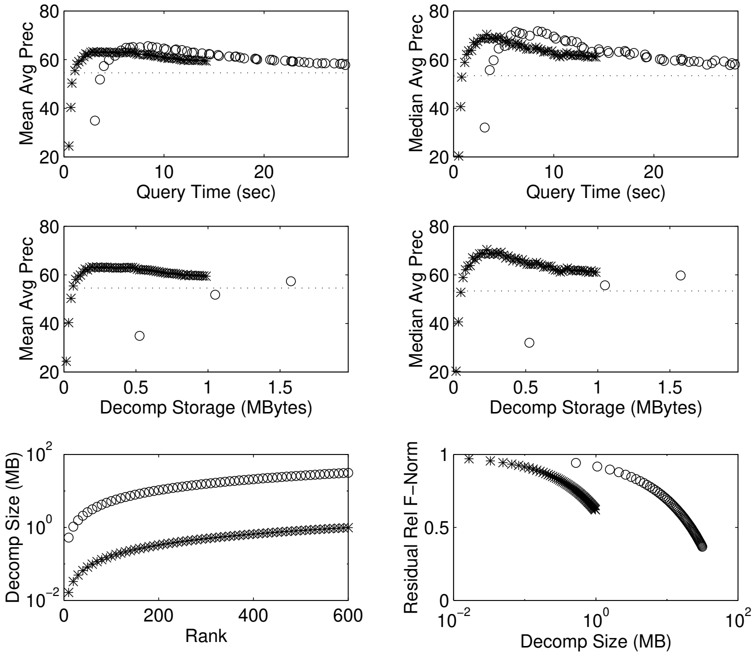
In terms of storage, the SDD method is extremely economical. The middle left graph of Figure 5.1 plots mean average precision vs. decomposition (in megabytes (MB)) size and the middle right graph plots median average precision vs. the decomposition size. Note that a significant amount of extra storage space is required in the computation of the SVD; this is not reflected in these numbers. From these plots, we see that even a rank-30 SVD takes 50% more storage than a 600-dimensional SDD, and each increment of 10 in rank adds approximately 0.5 MB of additional storage to the SVD. The SVD requires over 1.5 MB before it even begins to come close to what the SDD can do in less than 0.2 MB.

The lower left graph illustrates the growth in required storage as the dimension of the decomposition grows. For a rank-600 approximation, the SVD requires over 30 MB of storage while the SDD requires less than 1 MB.

It is interesting to see how good these methods are at approximating the matrix. The lower right graph plots the ratio of the relative Frobenius norm (F-norm) of the residual to the Frobenius norm of A, as a function of storage (logarithmic scale).

6. Conclusions. In these limited experiments, the discrete decomposition was a competitive alternative to the SVD for latent semantic indexing and offers an improvement over the vector space method. The discrete

FIG. 5.1. *A comparison of the SVD (∘) and SDD (∗) on the MEDLINE data set. We plot 60 data points for each graph corresponding to* $k = 10, 20, \ldots, 600$. *The dotted lines show the corresponding data for the vector space method.*



model uses only a small fraction of the storage space.

In a future report we will discuss results of more extensive tests and explore the critical issue of dynamic updating of the document collection.

Acknowledgements. We are grateful to Duncan Buell, John Conroy, Ken Kolda, Steve Kratzer, Joe McCloskey, and Doug Oard for helpful comments.

REFERENCES

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

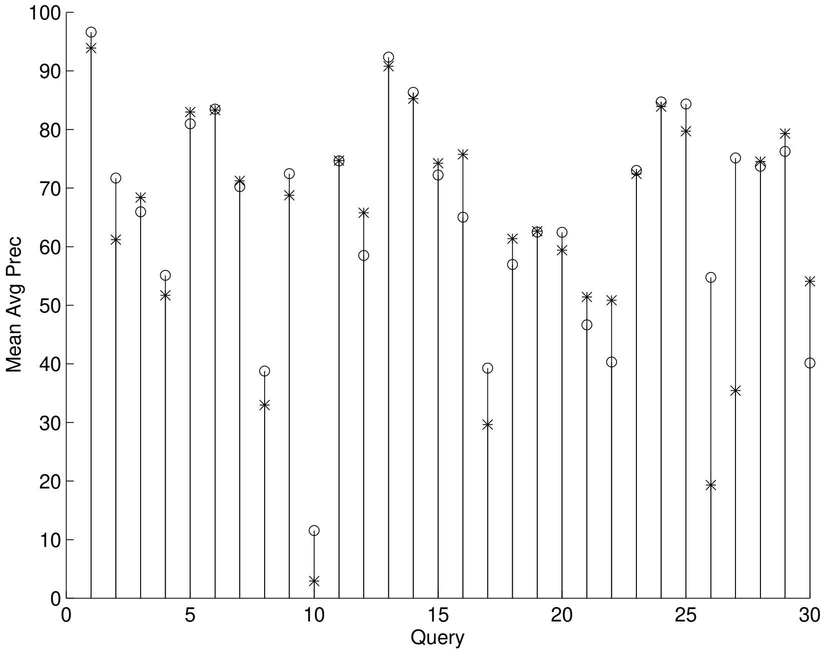
[2] J. P. Callan, B. Croft, and S. M. Harding, *The INQUERY retrieval system*, in Proceedings of the Third International Conference on Database and Expert Systems Applications, Springer-Verlag, 1992, pp. 78–83.

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

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

[5] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

FIG. 5.2. *A comparison of the SVD ($k = 110$) and SDD ($k = 120$) methods on the 30 individual queries from the MEDLINE data set. The asterisks (*) represent the SDD method and the circles (o) represent the SVD method.*



[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins Press, 2nd ed., 1989.

[7] D. P. O'LEARY AND S. PELEG, *Digital image compression by outer product expansion*, IEEE Transactions on Communications, 31 (1983), pp. 441–444.

[8] G. SALTON AND M. J. MCGILL, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.