

Updating the Partial SVD: Making LSI Run Faster

Jane E. Tougas*

Henry Stern†

1 Problem and Motivation

As the size of modern databases increases, the importance of having efficient methods of information retrieval (IR) increases accordingly. Latent Semantic Indexing (LSI) is an IR method that uses procedures from numerical linear algebra to represent a text collection as a *term-document matrix* [4]. The term-document matrix contains a column vector for each document in the text collection, and a row for each semantically significant term; words that contribute little to the meaning of a document, such as *a*, *the*, *is* and *and* are considered semantically insignificant, and are not included in the term-document matrix. A term-document matrix \mathbf{A} thus has t rows and d columns, where t is the number of semantically significant terms and d is the number of documents. Each entry $\mathbf{A}_{i,j}$ in the matrix indicates the importance of term i in document j , where $1 \leq i \leq t$ and $1 \leq j \leq d$. Search queries are represented as t -dimensional vectors.

The retrieval of information from a text collection is typically complicated by the fact that more than one word may have the same meaning (some words have *synonyms*). If one or more synonyms exist for a term that is entered in a search query, the documents containing the synonym(s) but not the search term may be overlooked, even though they are relevant. This is known as *recall failure*. If, on the other hand, a term that is entered in a search query has more than one meaning (some words are *polysemous*), then irrelevant documents about the term's other meaning(s) may be retrieved by the query. This is known as *precision failure*. LSI uses a matrix factorization method known as the *partial singular value decomposition* (PSVD) to reduce the problems of recall and precision failure. Using the PSVD, the data in the term-document matrix is projected into a lower-dimensional vector space, which has the effect of removing noise (caused by factors such as synonymy and polysemy) from the data and giving a better representation of the text collection. Query vectors are also projected into the lower-dimensional space using the PSVD. The vectors of documents and queries with many similar terms will be close together in the vector space, whereas those with few similar terms will be far apart. In LSI, the distance between two vectors is often measured using the cosine of the angle between the vectors; this is the *similarity measure*. LSI retrieves the documents that are most similar to a search query (those closest to the query in the vector space), even if the documents do not contain all (or any) of the terms contained in the query.

Unfortunately, traditional methods of computing the PSVD are computationally intensive; most of the processing time in LSI is spent in calculating the PSVD of the term-by-document matrix [1, 2].

*Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada, (tougas@cs.dal.ca). The work of this author is supported by NSERC Canada and the Killam Trust, and supervised by Raymond J. Spiteri, Department of Computer Science, University of Saskatchewan, Saskatoon, SK, S7N 5C9, Canada (spiteri@cs.usask.ca)

†Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada, (stern@cs.dal.ca).

In a dynamic environment, such as the Internet, the term-document matrix is altered often as new documents are added. Given the tremendous size of modern databases, recomputing the PSVD of the matrix each time such changes occur can be prohibitively expensive. LSI traditionally uses a method known as *folding-in* to modify the PSVD, in order to avoid recomputing the PSVD each time changes are made to the term-document matrix. The folding-in method has the benefit of being very fast, however its accuracy may degrade very quickly. A much more accurate approach is to *update* the PSVD using a method introduced by Zha and Simon [6]. This updating method modifies the PSVD of the term-document matrix to reflect the additions that are to be made to matrix. A new method, *folding-up*, is a combination of folding-in and updating the PSVD that is an even more attractive option. Folding-up offers a significant improvement in computation time when compared with either recomputing the PSVD or updating the PSVD, and yet it results in little or no loss of accuracy. Note that for the sake of simplicity, we refer only to the addition of documents to the term-document matrix; similar methods and results apply when new terms are added to the term-document matrix.

2 Background and Related Work

The SVD is a matrix factorization that captures the most important characteristics of the matrix. Given a matrix \mathbf{A} with t rows and d columns, its SVD is written as $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where \mathbf{U} is an orthogonal matrix with t rows and columns, and \mathbf{V} is an orthogonal matrix with d rows and columns. \mathbf{U} and \mathbf{V} contain the left and right singular vectors of \mathbf{A} , respectively. Matrix $\mathbf{\Sigma}$ has t rows and d columns, with non-zero entries only on the diagonal. These diagonal entries are in non-increasing order, and are known as the *singular values* of matrix \mathbf{A} . The number of non-zero singular values of a matrix is its *rank*, r . Let matrices \mathbf{U}_k and \mathbf{V}_k be the first k columns of \mathbf{U} and \mathbf{V} respectively, and matrix $\mathbf{\Sigma}_k$ be the leading submatrix of $\mathbf{\Sigma}$ with k rows and k columns. $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ is a lower-rank approximation of \mathbf{A} , where $k < r$. This is the *partial* SVD (PSVD) of matrix \mathbf{A} . This approximation can be used to reduce the dimension of a term-by-document matrix. The dimensional reduction has the effect of reducing the noise and accentuating the latent patterns in the data. Thus, the matrix \mathbf{A}_k can be a better representation of the data than the original term-document matrix \mathbf{A} . Note that it is never necessary to actually form the matrix \mathbf{A}_k ; its PSVD, $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ is used instead. The optimal number of dimensions (singular values and corresponding left and right singular vectors) to keep in the reduced term-by-document matrix varies, but experiments indicate that between 100 and 300 give the best results [2]. This tremendous dimensional reduction demonstrates the power of the PSVD as a method of data compression.

$$\boxed{\mathbf{A}} = \boxed{\mathbf{U}} \boxed{\mathbf{\Sigma}} \boxed{\mathbf{V}^T}$$

The SVD of \mathbf{A}

$$\boxed{\mathbf{A}_k} = \boxed{\mathbf{U}_k} \boxed{\mathbf{\Sigma}_k} \boxed{\mathbf{V}_k^T}$$

The PSVD of \mathbf{A}

3 Approach and Uniqueness

We investigate the use of PSVD updating methods proposed by Zha and Simon [6]. Although updating methods have also been proposed by O’Brien [5] and Berry, Dumais and O’Brien [2], research indicates that these methods give inferior results when compared to the methods introduced by Zha and Simon [6]. Our research confirms these findings, and therefore we focus on Zha and Simon’s updating methods. The available research on these updating methods deals with performing a few relatively large updates [6]. Our focus is in simulating a much more dynamic environment in which it is necessary to do many small updates, relative to the size of the term-document matrix. As well, we compare the existing methods of recomputing the PSVD, folding-in, and updating the PSVD with the new hybrid method that we call folding-up.

Let $\mathbf{D} \in \mathfrak{R}^{t \times p}$ contain p document vectors to be added to the term-document matrix \mathbf{A} , and let $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ be the PSVD of \mathbf{A} , where k is the number of dimensions (singular values and corresponding left and right singular vectors) used. The recomputing method simply recalculates the PSVD of the term-document matrix from scratch each time changes are made to the matrix. Although this is very accurate, it is also very expensive. The much cheaper folding-in method projects the new documents into the lower-dimensional vector space by performing a matrix multiplication, $\mathbf{D}_k = \mathbf{D}^T \mathbf{U}_k \mathbf{\Sigma}_k^{-1}$, and then appending the result, $\mathbf{D}_k \in \mathfrak{R}^{t \times p}$, to the bottom of matrix \mathbf{V}_k . Note that matrices \mathbf{U}_k and $\mathbf{\Sigma}_k$ are not changed in any way with this method. This means that as more and more documents are folded in, the representation of the dataset becomes less and less accurate. The updating method of Zha and Simon is more complicated and expensive than the folding-in method, but has the benefit of modifying each of the matrices \mathbf{U}_k , $\mathbf{\Sigma}_k$, and \mathbf{V}_k ; see [6] for details. In the absence of roundoff errors, the result is the exact PSVD of the modified term-document matrix, in significantly less computation time than if it were recomputed from scratch. The new folding-up method uses a combination of folding-in and updating to modify the PSVD of the term-document matrix. The method determines when to update based on the number of documents that have been folded-in, relative to the size of the initial term-document matrix, or to the size of the last updated matrix if updates have already taken place. The method begins by folding-in documents, but only until the number of documents folded-in reaches a predetermined percentage p of the documents in the original matrix. The changes that have been made to matrix \mathbf{V}_k by the folding-in process are then discarded, and the PSVD is updated using the updating methods of Zha and Simon [6]. Folding-in is then resumed until the number of new documents folded-in reaches a predetermined percentage p of the updated matrix, and so on. This process requires saving the document vectors that have been folded-in between updates, but repays this overhead with faster computation times than updating alone, and better accuracy than folding-in alone.

4 Results and Contributions

Examples for this paper are run using `Matlab` Release 13 on an Ultra3 Sunfire V880. These results are produced using the MEDLINE text collection [3], containing 1033 documents, 5735 semantically significant terms, and 30 queries. The measure of similarity is the cosine of the angle between the query and document vectors. The term-document matrix \mathbf{A}_{Med} is partitioned such that the first 433 columns form the initial matrix, and the remaining 600 columns are added incrementally. In each example, the average precision for each of the queries at the eleven standard recall levels

(0%, 10%, . . . , 100%) is averaged to produce the overall average precision at each increment. The average precisions for the four methods discussed in this paper (recomputing, folding-up, updating, and folding-in) are compared. For each example, $k = 125$, where k is the number of singular values and corresponding left and right singular vectors computed, and $p = 8$; when the folding-up method has folded-in documents equal to 8% of the initial term-document matrix, the PSVD is updated, and then folding-in resumes until the number of new documents folded-in reaches 8% of the updated matrix, and so on.

In the first example (see Figure 1), the initial term-document matrix of 5735 terms and 433 documents has 600 documents added to it in 120 increments of 5 documents each, simulating a dynamic environment in which frequent small changes are made to the term-document matrix. Note that the initial matrix more than doubles in size as a result of the incremental additions. As expected, Figure 1 indicates that the average precision for the folding-in method deteriorates rapidly compared with the other methods. The average precision for the updating method does not deteriorate until the initial matrix has approximately doubled in size, and then the deterioration is very slight. The folding-up method gives similar results to recomputing the PSVD at every increment, but as Table 1 illustrates, in this example the folding-up method is more than 400 times faster than recomputing. The folding-up method gives even better results than the updating method for much of Figure 1 and yet, in this example, it is more than three times faster than the updating method.

In the second example (see Figure 2), the initial term-document matrix with 5735 terms and 433 documents once again has 600 documents added to it, but in this case there are 60 increments of 10 documents each. As in the first example, this simulates a dynamic environment in which the term-document matrix is enlarged frequently. As in the previous example, Figure 2 shows that the average precision for the folding-in method deteriorates rapidly compared to the other methods. In this example both the updating method and the folding-up method give similar results when compared to the method of recomputing at each increment, but the updating method is more than 120 times faster than recomputing, and the folding-up method is more than 200 times faster than recomputing.

The contribution of these results is two-fold. First, we have demonstrated that the updating methods proposed by Zha and Simon [6] are effective in a dynamic environment in which there are many small updates made to the term-document matrix. This method of updating the PSVD achieves similar average precision to recomputing the PSVD, using only a fraction of the computation time. This in itself is significant, but we have also demonstrated that our new hybrid method, folding-up, is an even more attractive option than updating alone. As with the updating method, the new folding-up method achieves average precision similar to that of recomputing the PSVD, but the folding-up method requires less computation time than either recomputing or updating the PSVD.

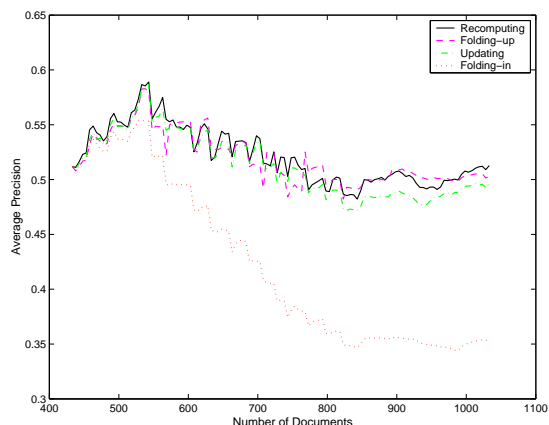


Figure 1: Average precisions for four methods using MEDLINE collection: 600 documents added in 120 groups of 5.

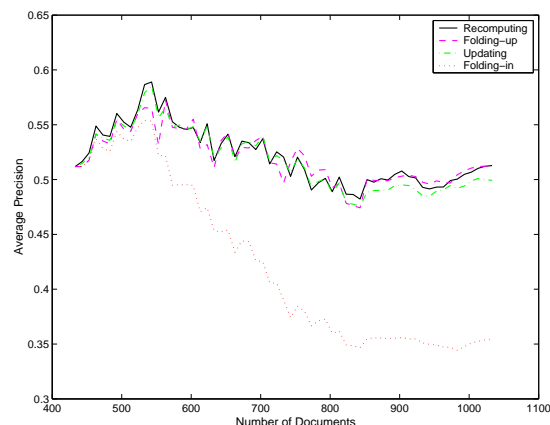


Figure 2: Average precisions for four methods using MEDLINE collection: 600 documents added in 60 groups of 10.

Method	CPU time	
	Increments of 5	Increments of 10
Recomputing	12689.14	6708.55
Folding-up	31.27	32.03
Updating	99.78	53.05
Folding-in	3.04	1.76

Table 1: CPU times (seconds): 600 documents added in groups of 5 and in groups of 10.

References

- [1] M. W. Berry, S. T. Dumais, and T. A. Letsche. Computational methods for intelligent information access, 1995. Presented at the Proceedings of Supercomputing.
- [2] M. W. Berry, S. T. Dumais, and G. W. O’Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.
- [3] Cornell SMART System <ftp://cs.cornell.edu/pub/smart>.
- [4] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [5] G. W. O’Brien. Information tools for updating an SVD-encoded indexing scheme, 1994. Master’s Thesis, The University of Knoxville, Tennessee.
- [6] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21(2):782–791, 1999.