A New Adaptive Folding-up Algorithm for Information Retrieval

Jane E. Mason [*]   Raymond J. Spiteri [†]

Abstract

Text collections can be represented mathematically as term-document matrices. A term-document matrix can in turn be represented using the matrix factorization method known as the partial (or truncated) singular value decomposition (PSVD). Recomputing the PSVD when changes are made to a text collection is very expensive. *Folding-in* is one method of approximating the PSVD when new documents are added to a term-document matrix; *updating* the PSVD of the existing term-document matrix is another method. The folding-in method is computationally inexpensive, but it may cause deterioration in the accuracy of the PSVD. The PSVD-updating method is more expensive than the folding-in method, but it maintains the accuracy of the PSVD. *Folding-up* is a method that combines folding-in and PSVD-updating. When a text collection expands in small increments, folding-up provides a significant improvement in computation time when compared with either recomputing the PSVD or PSVD-updating, and it reduces the loss of accuracy in the PSVD that can occur with the folding-in method. This paper introduces a new adaptive folding-up method in which a measure of the error in the PSVD is monitored to determine when it is most advantageous to switch from folding-in to updating.

1 Introduction

Latent semantic indexing (LSI) is an information retrieval (IR) method that represents a text collection as a term-document matrix [7]. LSI uses a matrix factorization method known as the partial (or truncated) singular value decomposition (PSVD) to reduce noise in the data by projecting the term-document matrix into a lower-dimensional vector space. Calculating the PSVD of a large term-document matrix is so computationally expensive that in LSI, most of the processing time is spent in performing this computation [3, 4]. In a dynamic environment, a term-document matrix is changed frequently as new documents and terms are added. Recomputing the PSVD of the term-document matrix each time these changes take place can be prohibitively expensive. A method known as *folding-in* can be used to modify the PSVD, rather than recomputing the PSVD each time changes are made to the document collection. Folding-in is computationally efficient; however its accuracy may degrade very quickly. A more accurate approach is to *update* the PSVD using a PSVD-updating method [9, 4, 14]. Updating methods modify the PSVD of the term-document matrix to reflect additions to the text collection. *Folding-up* is a combination of folding-in and PSVD-updating. When additions to a text collection are made in small increments, folding-up offers a significant improvement in computation time when compared with either recomputing or updating the PSVD, and yet it results in little or no loss of accuracy [11]. This paper introduces an effective and efficient new adaptive folding-up algorithm that uses an error threshold to determine when to switch from folding-in to updating.

The remainder of the paper proceeds as follows. Section 2 reviews background information about term-document matrices, the singular value decomposition (SVD), and the PSVD. Section 3.2 briefly describes methods of modifying the PSVD of a term-document matrix when new documents are added to the text collection and discusses the objective and approach of the new adaptive folding-up algorithm. Section 4 gives details of a selection of the experiments performed and the results obtained, and Section 5 presents conclusions and suggestions for future work.

2 Background

2.1 The Term-document Matrix. A term-document matrix $\mathbf{A}$ has t rows and d columns,

*(Formerly Jane E. Tougas.) jmason@cs.dal.ca Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada. The work of this author is supported by NSERC Canada and the Killam Foundation.

†spiteri@cs.usask.ca Department of Computer Science, University of Saskatchewan, Saskatoon, SK, S7N 5C9, Canada. The work of this author is supported by a grant from NSERC Canada.

where t is the number of *semantically significant* terms and d is the number of documents. Semantically significant terms are those terms that are useful in differentiating between documents. Words that occur in at least 80% of the documents are known as *stopwords*; they are typically not included in the term-document matrix [1]. In processing the text, a process known as *stemming* is usually performed to reduce words to their root form. By reducing words to a common concept, retrieval performance may be enhanced. It is also possible that stemming may relate nonrelevant terms, causing nonrelevant documents to be retrieved. Perhaps the main benefit of stemming is that it reduces the number of index terms and thus the size of the term-document matrix. When the number of documents is very large, stemming can provide a significant saving in the storage needed for the term-document matrix. There are a number of automatic stemming algorithms; the most popular is Porter's algorithm [1].

2.2 The SVD. The SVD is a matrix factorization that captures important characteristics of a matrix. Every matrix has an SVD, and the *singular values* of a matrix are always uniquely determined [13]. Given a real matrix $\mathbf{A}$ with t rows and d columns, its SVD has the form $\mathbf{A} = \mathbf{U\Sigma V^T}$, where $\mathbf{U} \in \Re^{t\times t}$, $\mathbf{\Sigma} \in \Re^{t\times d}$, and $\mathbf{V} \in \Re^{d\times d}$. Matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices that contain the left and right *singular vectors* of $\mathbf{A}$ respectively. When $\mathbf{A}$ is a term-document matrix, the left singular vectors represent the term vectors, and the right singular vectors represent the document vectors. The matrix $\mathbf{\Sigma}$ has non-zero entries only on the diagonal, although not all diagonal entries are necessarily non-zero. These diagonal entries are the singular values of $\mathbf{A}$. By convention, the singular values are assumed to be in non-increasing order and denoted by σ_j, for $j = 1, 2, \ldots, \min(t, d)$. The singular values are the non-negative square roots of the eigenvalues of the product $\mathbf{AA}^T$ [8]. The number of non-zero singular values is the rank, r, of the matrix.

2.3 The PSVD. Let matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ be the first k columns of $\mathbf{U}$ and $\mathbf{V}$ respectively, and let matrix $\mathbf{\Sigma}_k$ be the leading submatrix of $\mathbf{\Sigma}$ with k rows and k columns. Then $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ is a lower-rank approximation of $\mathbf{A}$, where $k < r$. This is called the PSVD of matrix $\mathbf{A}$. Using this approximation of the term-document matrix gives a dimensional reduction that has the effect of diminishing the noise and emphasizing the latent patterns in the data. For this reason, the matrix $\mathbf{A}_k$ can be a better representation of the text collection than the original term-document matrix $\mathbf{A}$. The optimal rank (number of singular values and corresponding left and right singular vectors) to use in the PSVD varies; it is also database dependent [4]. Deciding what rank to use is based on empirical testing [2]. In general, using a higher rank does not necessarily give better retrieval performance [7]. As the rank is increased, retrieval performance tends to increase to a certain point, but then it tends to decrease as the rank continues to be increased [7, 4].

3 Updating Methods

When new documents and terms are added to a text collection, the PSVD of the term-document matrix needs to be modified in order to reflect the changes to the text collection. As discussed in Section 1, recomputing the PSVD is computationally expensive. Other options are to modify the existing PSVD either by folding-in the new documents and terms, by using PSVD-updating algorithms, or by using a folding-up algorithm that combines folding-in and PSVD-updating. Note that although this paper focuses on the addition of documents, for each method discussed, the addition of terms follows a similar process. In the following sections, let $\mathbf{D} \in \Re^{t\times p}$ 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 rank (number of singular values and corresponding left and right singular vectors) used.

3.1 Folding-in. The folding-in method projects the new documents into the lower-dimensional vector space by constructing the matrix $\mathbf{D}_k = \mathbf{D}^T\mathbf{U}_k\mathbf{\Sigma}_k^{-1}$ and then appending $\mathbf{D}_k \in \Re^{t\times p}$ to the bottom of matrix $\mathbf{V}_k$ to form $\hat{\mathbf{V}}_k$. 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. Folding-in terms follows a similar process.

3.2 Updating. PSVD-updating algorithms for LSI were introduced by O'Brien in his Master's thesis [9] and published in Berry, Dumais, and O'Brien in 1995 [4]. In 1999, Zha and Simon [14] presented PSVD-updating algorithms that give superior results when compared to the PSVD-updating algorithms of [4, 9]. More recently, Brand [5] developed methods for modifying the PSVD in the case that $k \leq \sqrt{\min(t,d)}$, where k is the rank of the PSVD of a $t \times d$ matrix. This condition is not met in our experiments, and consequently we do not consider this method further. Methods for updating the PSVD when new documents or new terms are added to the LSI-database are a compromise between recomputing the PSVD and folding-in. Although these methods are slower than folding-in, they are much faster than recomputing the PSVD, and they can produce a more accurate PSVD than folding-in [14, 10, 11]. Section 3.2.1 describes Zha and Simon's algorithm for updating the PSVD of a term-document matrix when documents are added to the document collection. See [14] for further details.

3.2.1 Updating Documents.

Again, let $\mathbf{D} \in \Re^{t\times p}$ be the term-document matrix containing the document vectors to be appended to $\mathbf{A}$, where p is the number of new documents, and let $\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}]$ be the updated term-document matrix. Let $\mathbf{I}_n$ denote the identity matrix of size n. We assume that the PSVD of $\mathbf{A}$ is available prior to updating. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{D}} = \left(\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T\right)\mathbf{D} \in \Re^{t\times p}$. Form the QR decomposition of $\hat{\mathbf{D}}$, $\mathbf{Q_D}\mathbf{R_D} = \hat{\mathbf{D}}$, where $\mathbf{Q_D} \in \Re^{t\times p}$ is orthonormal and $\mathbf{R_D} \in \Re^{p\times p}$ is upper triangular. Then

$$\begin{aligned} \tilde{\mathbf{A}} &= [\mathbf{A}, \mathbf{D}] \\ &\approx [\mathbf{A}_k, \mathbf{D}] \\ &= [\mathbf{U}_k, \mathbf{Q_D}] \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T\mathbf{D} \\ \mathbf{0} & \mathbf{R_D} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{bmatrix}. \end{aligned}$$

Now let $\hat{\mathbf{A}} \in \Re^{(k+p)\times(k+p)}$ be the matrix defined by

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T\mathbf{D} \\ \mathbf{0} & \mathbf{R_D} \end{bmatrix}.$$

Form the SVD of $\hat{\mathbf{A}}$ such that

$$\hat{\mathbf{A}} = \left[\hat{\mathbf{U}}_k, \hat{\mathbf{U}}_p\right] \begin{bmatrix} \hat{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{\Sigma}}_p \end{bmatrix} \left[\hat{\mathbf{V}}_k, \hat{\mathbf{V}}_p\right]^T,$$

where $\hat{\mathbf{U}}_k \in \Re^{(k+p)\times k}$, $\hat{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, and $\hat{\mathbf{V}}_k \in \Re^{(k+p)\times k}$. Then the rank k PSVD of $\tilde{\mathbf{A}}$ (the updated PSVD) is

$$\tilde{\mathbf{A}}_k = \left([\mathbf{U}_k, \mathbf{Q_D}]\,\hat{\mathbf{U}}_k\right)\hat{\mathbf{\Sigma}}_k \left(\begin{bmatrix} \mathbf{V}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{bmatrix}\hat{\mathbf{V}}_k\right)^T.$$

3.3 Folding-up.

Tougas and Spiteri [11] introduced the original folding-up algorithm and showed it to be an attractive alternative to either folding-in or PSVD-updating alone. The idea behind folding-up is that when there are new documents (or terms) to be added, they are first folded-in, and the original document vectors are stored for later use. After a certain amount of folding-in has been done, the changes that have been made by the folding-in process are discarded, and the PSVD is updated to represent the current document collection using a PSVD-updating method, such as that of Zha and Simon [14]. The original document or term vectors that have been added to the LSI database by the PSVD-updating process can then be discarded. The goal of the folding-up method is to switch from the process of folding-in to that of using a PSVD-updating method before the accuracy of the PSVD degrades significantly from the folding-in process. After the PSVD-update has been performed, the cycle begins again; documents and/or terms are again folded-in until it is deemed necessary to switch to a PSVD-updating method. In folding-up, the PSVD-updating process can be thought of as a correction step, but it is important to decide at what point this step becomes necessary in order to best exploit both the computational efficiency of folding-in and the computational accuracy of PSVD-updating.

3.4 Adaptive Folding-up.

In the original folding-up algorithm, documents are folded-in until the number of folded-in documents reaches a pre-selected percentage of the number of documents in the text collection, not including folded-in documents. The choice of the percentage used is empirically based. Adaptive folding-up is a modification of that algorithm; in the adaptive folding-up method, a measure of the error in the PSVD produced by folding-in is monitored to determine when it is most advantageous to switch from folding-in to PSVD-updating. This adaptive folding-up algorithm eliminates the need to choose the percentage of documents that may be folded-in before a PSVD-update occurs; instead it computes a measure of the accumulated error, based on the loss of orthogonality in the right singular vectors of the PSVD produced by folding-in. The algorithm uses an empirically determined error threshold, τ, to determine whether or not a PSVD-update is necessary.

Before each new group of documents is folded-in, the adaptive folding-up method computes the error that would be introduced by folding-in these documents. If adding this error to the accumulated error causes it to exceed the error threshold of $\tau = 0.01$, then PSVD-updating is performed instead of folding-in. The accumulated error is then reset to zero. The error threshold $\tau = 0.01$ was established through extensive empirical testing on the Medline, Cranfield, and CISI datasets. Section 4.2 gives results using this error threshold on larger datasets.

3.4.1 Establishing the Error Measure. Recall from Section 2.3 that the PSVD of the term-document matrix $\mathbf{A}$ is $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$, where k is the number of singular values used in the approximation. Also recall that the columns of matrix $\mathbf{V}_k$ are mutually orthogonal, and that folding-in p documents appends p rows to the bottom of $\mathbf{V}_k$, yielding $\hat{\mathbf{V}}_k$. This process corrupts the orthogonality of the columns of $\hat{\mathbf{V}}_k$, potentially leading to a misrepresentation of the text collection.

The product of an orthogonal matrix, and its transpose is the identity matrix; therefore the loss of orthogonality in $\hat{\mathbf{V}}_k$ can be measured by

$$\|\hat{\mathbf{V}}_k^T \hat{\mathbf{V}}_k - \mathbf{I}_k\|_F, \tag{3.1}$$

where $\|\cdot\|_F$ is the Frobenius matrix norm [8], and $\mathbf{I}_k \in \Re^{k\times k}$ is the identity matrix. However, when the term-document matrix is very large, computing Equation (3.1) as written is computationally expensive. Because $\hat{\mathbf{V}}_k$ can be written as

$$\hat{\mathbf{V}}_k = \begin{bmatrix} \mathbf{V}_k \\ \mathbf{D}_k \end{bmatrix},$$

the product $\hat{\mathbf{V}}_k^T \hat{\mathbf{V}}_k$ can be expressed as

$$\begin{aligned} \hat{\mathbf{V}}_k^T \hat{\mathbf{V}}_k &= \begin{bmatrix} \mathbf{V}_k^T & \mathbf{D}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{V}_k \\ \mathbf{D}_k \end{bmatrix} \\ &= \left[\mathbf{V}_k^T \mathbf{V}_k + \mathbf{D}_k^T \mathbf{D}_k\right]. \end{aligned} \tag{3.2}$$

Substituting Equation (3.2) into Equation (3.1), and using the fact that the product of an orthogonal matrix and its transpose is the identity matrix, gives

$$\begin{aligned} \|\mathbf{V}_k^T \mathbf{V}_k + \mathbf{D}_k^T \mathbf{D}_k - \mathbf{I}_k\|_F &= \|\mathbf{I}_k + \mathbf{D}_k^T \mathbf{D}_k - \mathbf{I}_k\|_F \\ &= \|\mathbf{D}_k^T \mathbf{D}_k\|_F. \end{aligned} \tag{3.3}$$

In Equation (3.3) we take the Frobenius norm of the matrix $\mathbf{D}_k^T \mathbf{D}_k$, which is of size $k \times k$; in order to construct a largely dataset-independent error measure, we divide Equation (3.3) by k. Thus, the deterioration in the orthogonality of $\hat{\mathbf{V}}_k$ can be measured by accumulating the error

$$\frac{\|\mathbf{D}_k^T \mathbf{D}_k\|_F}{k} \tag{3.4}$$

with each folding-in step. Using the result from Equation (3.4) rather than Equation (3.1) to monitor the deterioration in orthogonality offers significant computational savings. This is because $\hat{\mathbf{V}}_k$ is of size $(d+p)\times k$ whereas $\mathbf{D}_k$ is of size $p \times k$, with typically $d \gg p$, thus the multiplication $\mathbf{D}_k^T \mathbf{D}_k$ is much less expensive than the multiplication $\hat{\mathbf{V}}_k^T \hat{\mathbf{V}}_k$. It is also less expensive to compute the Frobenius norm of the smaller matrix.

In order to determine whether the error measure in Equation (3.4) is a valid measure of the error the folding-in method causes in the PSVD, we compare the error measure against the difference between the *optimal* and *relative* error in the PSVD. The optimal-error measures the difference between the original matrix $\mathbf{A}$ and the lower-rank matrix $\mathbf{A}_k = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^T$, where $\mathbf{A}_k = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^T$ is the matrix formed when the PSVD of the term-document matrix is recomputed each time changes are made to the document collection. In this paper, the computation is performed using *Matlab's* `svds` function for sparse matrices. The relative error measures the difference between the original matrix $\mathbf{A}$ and the lower-rank matrix $\hat{\mathbf{A}}_k = \hat{\mathbf{U}}_k \hat{\mathbf{S}}_k \hat{\mathbf{V}}_k^T$, where $\hat{\mathbf{A}}_k = \hat{\mathbf{U}}_k \hat{\mathbf{\Sigma}}_k \hat{\mathbf{V}}_k^T$ is the PSVD of $\mathbf{A}$ that has been modified by the folding-in method. The optimal and relative error are defined by

$$\frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{\|\mathbf{A}\|_F^2} \quad \text{and} \quad \frac{\|\mathbf{A} - \hat{\mathbf{A}}_k\|_F^2}{\|\mathbf{A}\|_F^2}, \tag{3.5}$$

respectively. The optimal and relative error are generally too computationally expensive to compute in practice, but they can be computed for relatively small test sets such as the Medline text collection (1033 documents), the Cranfield collection (1398 documents), and the CISI text collection (1460 documents). Henceforth, the difference between the optimal and relative error will be referred to as the *PSVD-error*.

Figures 1, 2, and 3 show a comparison of the new PSVD-error introduced at each iteration with the error computed using the error measure in Equation (3.4), as documents are added to the text collections. In each case, the vectors have been normalized. Figure 1 shows the error comparison for the Medline text collection; the initial matrix contains 533 documents, and there are 100 iterations of 5 documents added at each iteration. Figure 2 shows the error comparison for the Cranfield text collection; the initial matrix contains 598 documents, and there are 400 iterations of 2 documents added at each iteration. Figure 3 shows the error comparison for the CISI text collection; the initial matrix contains 460 documents, and there are 100 iterations of 10 documents added at each iteration. The figures show that although there is not an exact correlation between the two error measures, the loss of orthogonality error measure from Equation (3.4) follows a very similar pattern to that of the PSVD-error, suggesting that the error measure in Equation (3.4) is a useful measure of the error in the PSVD caused by the folding-in method. The results shown in Figures 1–3 are a representative subset of the tests performed, and are intended to present a variety of increment sizes. Each of the datasets was tested with several increment sizes; the number of documents added at each iteration for a particular test was chosen from the range of 1–20. The correlation as displayed in Figures 1–3 was similar in each case.

With the adaptive folding-up method, the deterioration of the orthogonality of $\hat{\mathbf{V}}_k$ when documents are being added (or $\hat{\mathbf{U}}_k$ when terms are being added) is monitored by accumulating the error computed by Equation (3.4) each time documents are folded-in. This corruption of orthogonality is a measure of how much inaccuracy has been introduced to the representation of the text collection with the addition of new documents (and terms). When the loss of orthogonality reaches the empirically established error threshold of $\tau = 0.01$, the changes that have been made by the folding-in method are discarded. PSVD-updating methods are then applied such that the modified PSVD reflects the addition of all of the document (and term) vectors that have been folded-in since the last update. The process then continues with folding-in until the next update.The process of folding-up has the overhead of saving the document vectors that are being folded-in between updates; however, it repays this cost with a saving in computation time, coupled with the precision advantages of updating.

4 Experiments

For the sake of brevity, we present a representative selection of the experiments we have performed. The experiments for this paper are run on five text collections. These are the Medline, Cranfield, and CISI collections [6], and two subsets of the TREC 2003 HARD track [12]. These subsets, subsequently referred to as HARD-1 and HARD-2, contain 15000 and 30000 documents respectively. They were created by randomly selecting documents from the HARD track text collection, with no document duplication within the subsets. In each experiment the term-document matrix for the whole text collection is partitioned into an initial matrix and a number of smaller submatrices. The initial matrix is incrementally enlarged by iteratively appending the submatrices, and the average precision for each method is plotted at each increment. Note that this precision is averaged not only over the number of queries, but also over the 11 standard recall levels $(0\%, 10\%, \cdots, 100\%)$. The measure of similarity for each experiment is the cosine similarity measure, which is the measure of the cosine of the angle between each pair of vectors [1, 2]. Note that search queries are represented as t-dimensional vectors. For each text collection, a local normalized term frequency weighting scheme is used; no global weighting is used.

In each experiment, the PSVD of the initial matrix is computed using the `svds` function for sparse matrices in *Matlab*. For the Medline collection, we use $k = 125$, where k is the number of singular values (and corresponding left and right singular vectors) computed. For the CISI collection, we use $k = 150$, and for the Cranfield, HARD-1, and HARD-2 text collections, we use $k = 300$. The choice of k is based on empirical testing and/or available computational resources; the use of an optimal value of k was not essential for these experiments. For the sake of brevity, the experiments described here use only document updating. We note that similar results are produced using term updating. Each experiment compares the average precision for recomputing, folding-in documents, PSVD-updating, the original folding-up method, and the adaptive folding-up method. In these experiments, Zha and Simon's PSVD-updating method is used; Zha and Simon have clearly shown [14] that their updating method outperforms O'Brien's PSVD updating method [4, 9] in terms of retrieval performance. We note that in each experiment, the original folding-up method uses a fixed percentage of 10% to determine how many documents to fold-in before performing a PSVD-update. Although this percentage is not necessarily optimal for these datasets, previous work has shown that it performs very well on a variety of datasets. In each experiment, the error threshold for the adaptive folding-up method is set at 0.01. Table 1 gives a comparison of CPU times (in seconds) for each of the experiments. These times are computed using the `cputime` function in *Matlab*.

4.1 Medline, Cranfield, and CISI Collections. The Medline text collection contains 1033 medical abstracts and 30 queries. After stopword removal and stemming, there are 5735 terms. The 5735×1033 term-document matrix is partitioned into an initial matrix with 533 documents, and 100 submatrices of size 5 are appended incrementally. Figure 4 shows the average precision for each method. The average precision for the folding-in method deteriorates rapidly, whereas the average precisions for the updating and adaptive folding-up methods are very similar to that of recomputing the PSVD each time new documents are added.

The Cranfield text collection contains 1398 aerospace systems abstracts and 225 queries. After stopword removal and stemming, there are 4563 terms, giving a 4563×1398 term-document matrix. This matrix is partitioned into an initial matrix with 598 documents, and 400 submatrices of size 2 are appended incrementally. The average precision for each method is shown in Figure 5. Note that in this case the adaptive folding-up method is more than 9 times faster than updating, and more than 1600 times faster than recomputing.

The CISI text collection contains 1460 information retrieval abstracts and 35 queries. After stopword removal and stemming, there are 5544 terms. The 5544×1460 term-document matrix is partitioned into an initial matrix with 460 documents, and 100 submatrices of size 10 are appended incrementally. Figure 6 shows the average precision for each method; as with Figures 4 and 5, the average precision for the folding-in method deteriorates compared with the average precision for recomputing, whereas the average precisions for updating and adaptive folding-up do not degrade. See Table 1 for a comparison of CPU times.

4.2 TREK 2003 HARD Track Subsets. The HARD-1 text collection contains 15000 documents and 50 queries. After stopword removal and stemming, there are 77250 terms. The 77250×15000 term-document matrix is partitioned into an initial matrix with 7500 documents, and 150 submatrices of size 50 are appended incrementally. The HARD-2 text collection contains 30000 documents and 50 queries. After stopword removal and stemming, there are 112113 terms, giving a 112113×30000 term-document matrix. This matrix is partitioned into an initial matrix with 15000 documents, and 150 submatrices of size 100 are appended incrementally. Figures 7 and 8 show the average precisions for each method for the HARD-1 and HARD-2 text collections respectively. As with Figures 5 and 6, the average precision for folding-in deteriorates; in Figure 7, this deterioration is dramatic. In each case, the average precisions for the updating and adaptive folding-up methods outperform those of the folding-in method. Table 1 gives a comparison of CPU times for each experiment. Note that in each case, the adaptive folding-up method is faster than either updating or recomputing the PSVD. Because the adaptive folding-up method has more computational overhead than the original folding-up method, it is somewhat slower than the original folding-up method for very small datasets; for larger datasets such as Hard-1 and Hard-2 however, Table 1 shows that the adaptive folding-up method is faster than the original folding-up method.

4.3 Discussion Folding-up is a method that combines folding-in and PSVD-updating. When a text collection expands in small increments, folding-up provides a significant improvement in computation time when compared with either recomputing the PSVD, or PSVD-updating, and it reduces the loss of accuracy in the PSVD that can occur with the folding-in method. The original folding-up method switches from folding-in to updating after a certain percentage of new documents are added. The goal of the new adaptive folding-in method is to establish a measure of the error in the PSVD that can be monitored to determine when it is most advantageous to switch from folding-in to updating. This eliminates the need to choose the percentage of new documents to be used in the original folding-up method. The percentage chosen in the original folding-up method is an empirically based best guess; in this paper we use 10% for each dataset. Although this gives good results in terms of average precision, fewer (larger) PSVD-updates are performed than with the adaptive folding-up method. Because the PSVD-updating method uses expensive QR and SVD computations on intermediate matrices, on large datasets, performing fewer large PSVD-updates can be more costly than performing a greater number of smaller PSVD-updates. This is why adaptive folding-up is more efficient than the original folding-up method for the larger datasets discussed in Section 4.2. For each of the experiments in Section 4, adaptive folding-up performs more PSVD-updates than the original folding-up method. Lowering the percentage used in the original folding-up method would increase the number of PSVD-updates performed and decrease their size, but this does not necessarily give better performance than the adaptive folding-up method. With the original folding-up method, PSVD-updates are evenly spaced, whereas with the adaptive method, updates are performed when the accumulated error exceeds the established threshold. For each dataset and each increment size there is a crossover point at which it is less efficient to do fewer larger PSVD-updates than to do more smaller PSVD-updates. Table 2 gives a comparison of the number of PSVD-updates performed by each folding-up method for each experiment in Sections 4.1 and 4.2.

5 Conclusions

The experiments demonstrate that over a variety of text collections, when new documents are added to the collection in small groups, the adaptive folding-up method is a viable alternative to either recomputing the PSVD or using only a PSVD-updating method. In these experiments, adaptive folding-up achieves average precision similar to that of recomputing the PSVD, but it requires significantly less computation time than either recomputing or updating the PSVD. The goal of the folding-up method is to switch from the process of folding-in to that of using a PSVD-updating method before the accuracy of the numerical representation of the database degrades significantly from the folding-in process. This goal is facilitated by using the empirically established error measure threshold $\tau = 0.01$ to determine when the loss of orthogonality has become too great during the folding-in stage of the adaptive folding-up algorithm. Further research with larger datasets is ongoing.

Loss of orthogonality caused by the folding-in process is not the only possible error measure to use in determining when it is time to stop folding-in and do a PSVD-update step. Another possibility is to measure the distance between new document vectors and their lower-dimensional projections. The idea is that if the distance between these vectors is small, then the documents could be folded-in, but if the distance is large, then the documents would be added using a PSVD-updating step.

6 Acknowledgements

The authors wish to express their gratitude to Chris Jordan of Dalhousie University for his assistance in preparing the HARD text collection.

References

[1] R. A. Baeza-Yates, R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] M. W. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval.* Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, first edition, 1999.

[3] M. W. Berry, S. T. Dumais, and T. A. Letsche. Computational methods for intelligent information access, 1995. Presented at the Proceedings of Supercomputing.

[4] 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.

[5] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and Its Applications*, 415(1):20–30, 2006.

[6] Cornell SMART System ftp://ftp.cs.cornell.edu/pub/smart.

[7] 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.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[9] G. W. O'Brien. Information tools for updating an SVD-encoded indexing scheme, 1994. Master's Thesis, The University of Knoxville, Tennessee.

[10] J. E. Tougas and R. J. Spiteri. Two uses for updating the partial singular value decomposition in latent semantic indexing. *Applied Numerical Mathematics*, 2007. To appear.

[11] J. E. Tougas and R. J. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. *Computational Statistics and Data Analysis*, 52(1):174–183, 2007.

[12] TREC 2003 HARD track http://trec.nist.gov/data/t12_hard.html.

[13] L. N. Trefethen and D. Bau, III. *Numerical linear algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[14] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21(2):782–791, 1999.

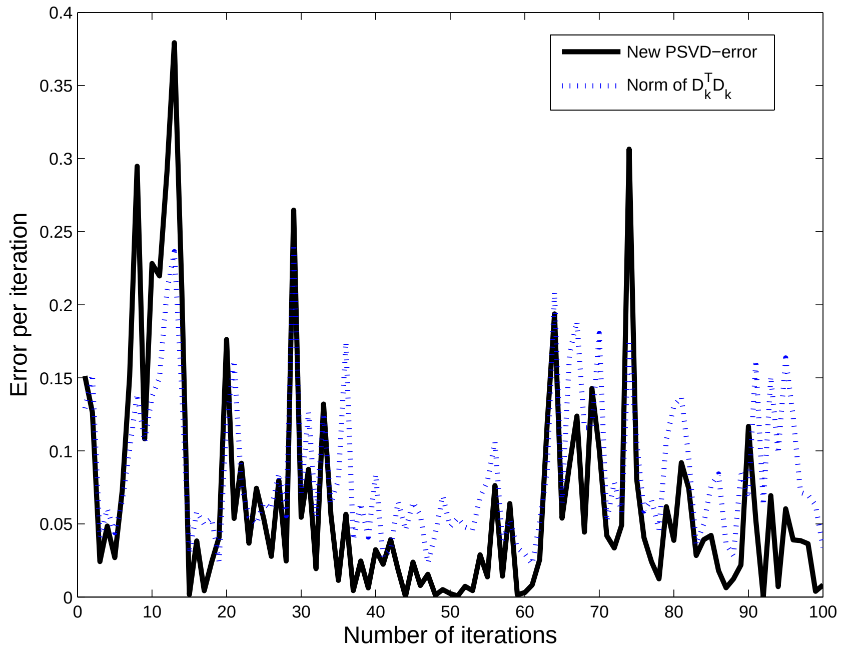

Figure 1: Comparison of normalized error for the Medline collection: 500 documents are added in 100 groups of 5.

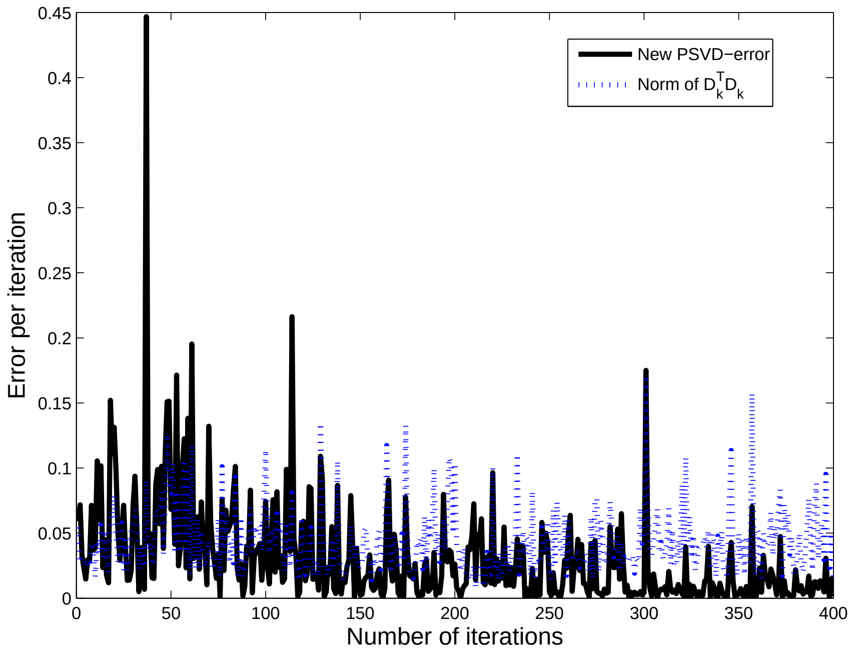

Figure 2: Comparison of normalized error for the Cranfield collection: 800 documents are added in 400 groups of 2.

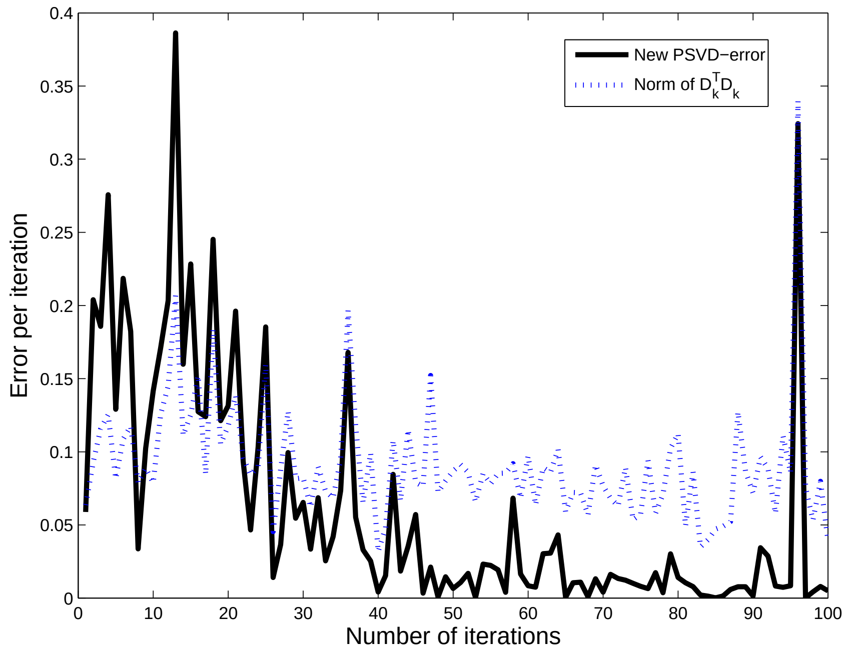

Figure 3: Comparison of normalized error for the CISI collection: 1000 documents are added in 100 groups of 10.

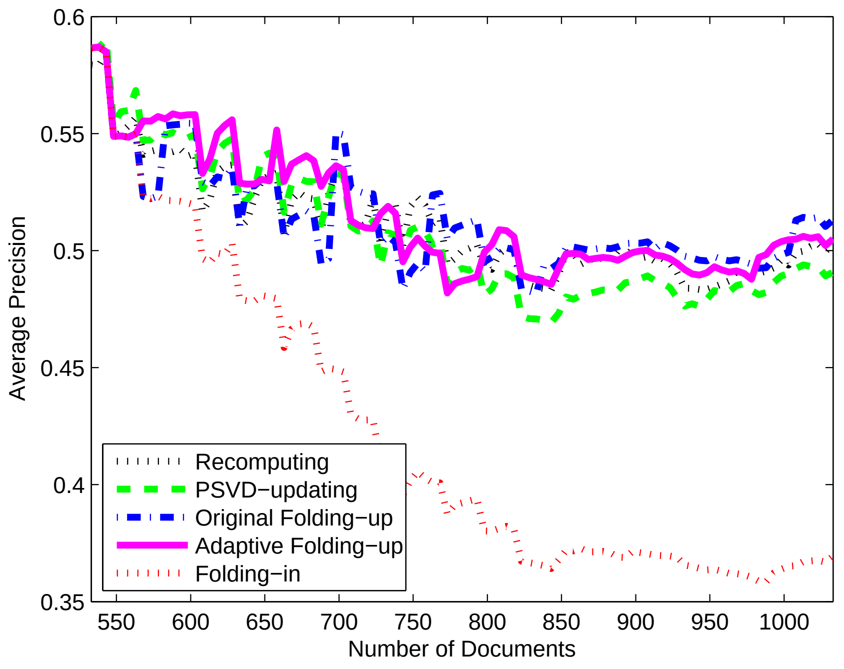

Figure 4: Average precisions for the Medline collection: 500 documents are added in 100 groups of 5.

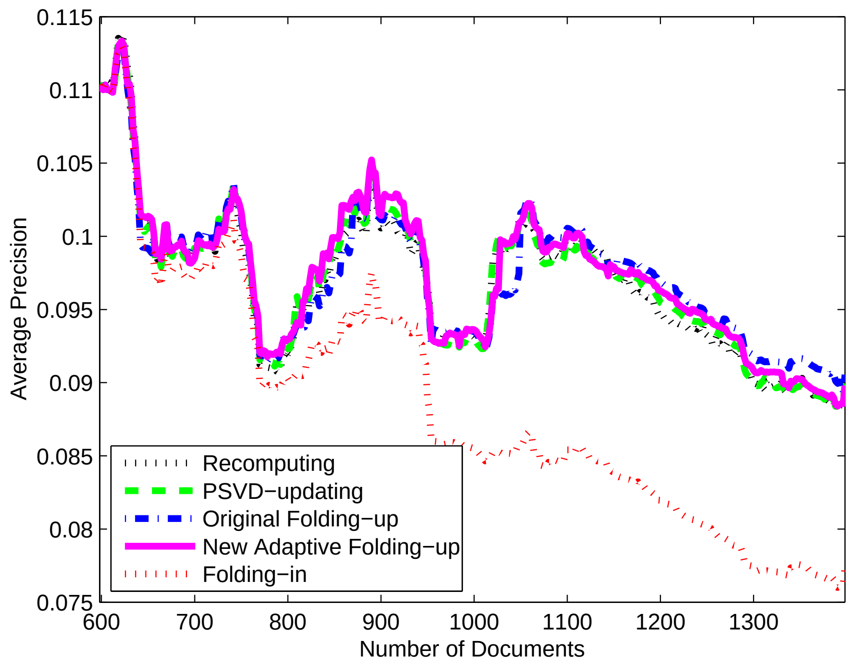

Figure 5: Average precisions for Cranfield collection: 800 documents are added in 400 groups of 2.

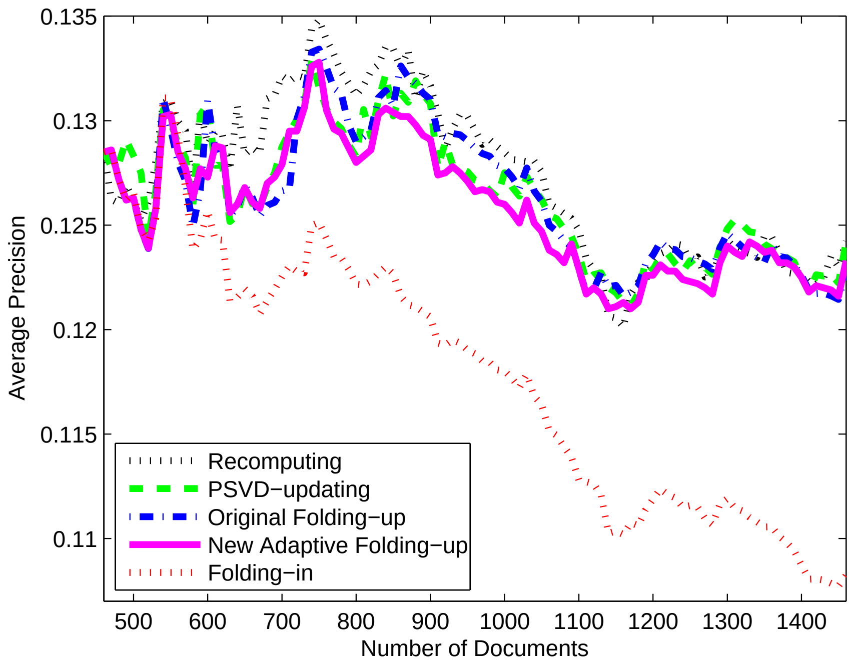

Figure 6: Average precisions for CISI collection: 1000 documents are added in 100 groups of 10.

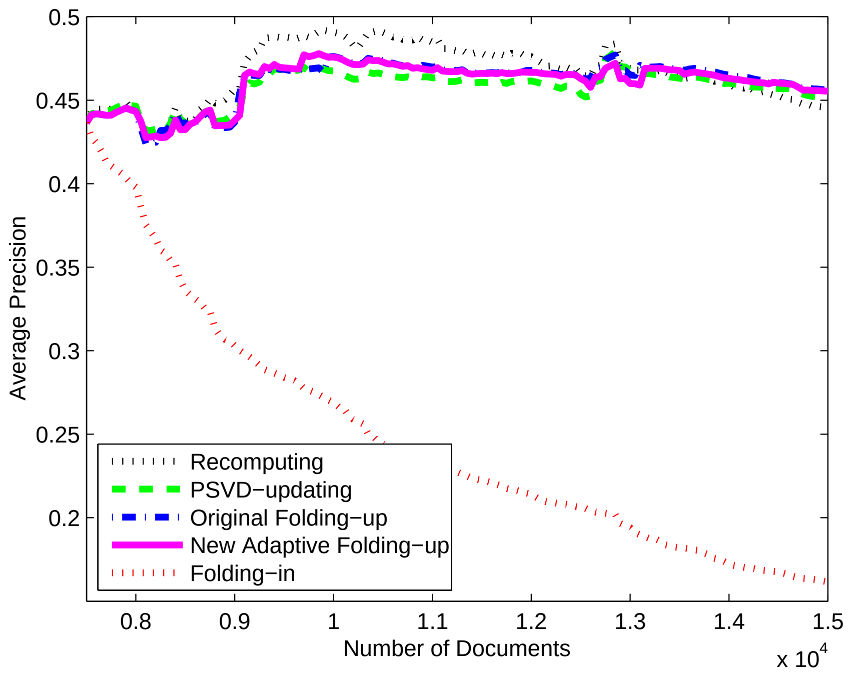

Figure 7: Average precisions for HARD-1 collection: 7500 documents are added in 150 groups of 50.

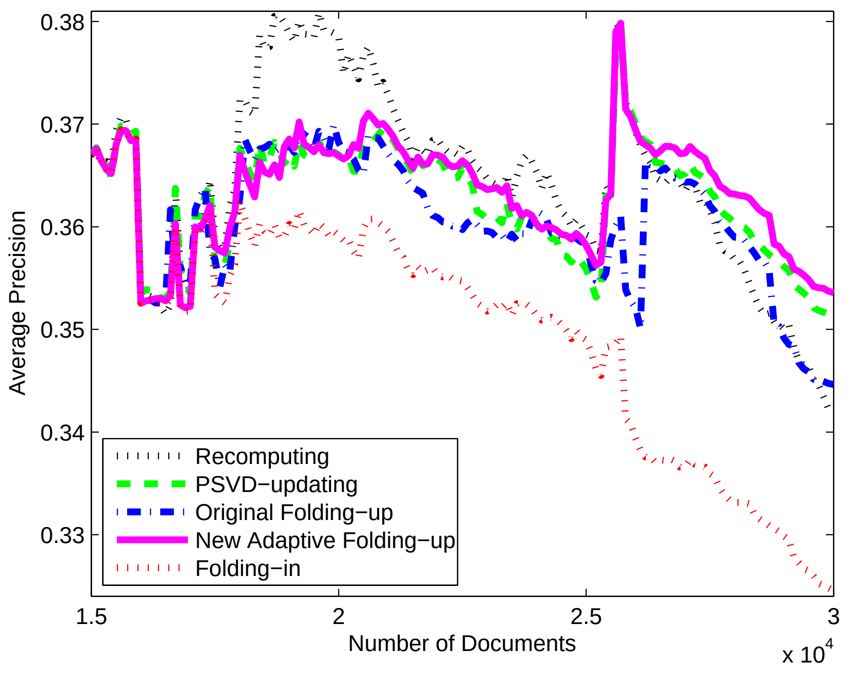

Figure 8: Average precisions for HARD-2 collection: 15000 documents are added in 150 groups of 100.

Method	Medline CPU times	Cranfield CPU times	CISI CPU times	HARD-1 CPU times	HARD-2 CPU times
Recomputing	9416.18	354556.25	17546.07	391487.60	587773.98
Updating	80.79	1972.97	131.09	1977.16	9020.10
Original Folding-up	16.54	91.70	36.46	1653.01	8175.15
Adaptive Folding-up	19.68	209.01	46.47	1306.58	6176.66
Folding-in	2.47	46.32	4.02	21.05	40.99

Table 1: Comparison of total CPU times (in seconds) for the Medline, Cranfield, CISI, HARD-1, and HARD-2 text collections.

Method	Medline PSVD-updates	Cranfield PSVD-updates	CISI PSVD-updates	HARD-1 PSVD-updates	HARD-2 PSVD-updates
Original Folding-up	7	9	13	7	7
Adaptive Folding-up	13	41	14	12	10

Table 2: Comparison of the number of PSVD-updates in the original and adaptive folding-up methods, for the Medline, Cranfield, CISI, HARD-1, and HARD-2 text collections.