Updating the Partial Singular Value Decomposition in Latent Semantic Indexing

Jane E. Tougas [a,1,*]

[a] *Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada*

Raymond J. Spiteri [b,2]

[b] *Department of Computer Science, University of Saskatchewan, Saskatoon, SK, S7N 5C9, Canada*

Abstract

Latent semantic indexing (LSI) is a method of information retrieval that relies heavily on the partial singular value decomposition (PSVD) of the term-document matrix representation of a dataset. Calculating the PSVD of large term-document matrices is computationally expensive; hence in the case where terms or documents are merely added to an existing dataset, it is extremely beneficial to *update* the previously calculated PSVD to reflect the changes. It is shown how updating can be used in LSI to significantly reduce the computational cost of finding the PSVD without significantly impacting performance. Moreover, it is shown how the computational cost can be reduced further, again without impacting performance, through a combination of updating and folding-in.

Key words: latent semantic indexing, singular value decomposition, updating, folding-in

* Corresponding author. Tel: +1-902-494-2093; Fax: +1-902-492-1517.
Email addresses: tougas@cs.dal.ca (Jane E. Tougas), spiteri@cs.usask.ca (Raymond J. Spiteri).

[1] The work of this author is supported by NSERC Canada and the Killam Foundation.

[2] The work of this author is supported by a grant from NSERC Canada.

1 Introduction

The seemingly disparate fields of information retrieval (IR) and numerical linear algebra (NLA) are closely linked via latent semantic indexing (LSI) (Deerwester et al., 1990). LSI is an IR method based on the vector-space model where a dataset is represented as a *term-document matrix*. LSI uses a matrix factorization method known as the partial singular value decomposition (PSVD) in an attempt to reduce the problems of *precision* and *recall failure* caused by *polysemy* and *synonymy*. Many terms have more than one meaning (they are *polysemous*). When a polysemous term is used in a search query, irrelevant documents about the term's other meaning(s) may be retrieved, degrading the precision level of the results. Moreover, many terms have similar meanings (they are *synonymous*). When a term that has a synonym is used in a query, relevant documents containing the synonym, but not the query term, may be overlooked, degrading the recall level of the results. Research indicates that LSI is better at dealing with the problems caused by synonymy than those caused by polysemy (Deerwester et al., 1990), but this does not detract from the importance of LSI in IR.

As a vector-space model, LSI examines the document collection as a whole and determines which documents contain many of the same terms. The more terms that documents have in common, the more closely related the documents are considered to be. This process involves creating a term-document matrix $\mathbf{A} \in \Re^{t \times d}$, where t is the number of semantically significant terms, and d is the number of documents in the document collection. Each entry of $\mathbf{A}$ represents the weighted frequency of a particular term in a particular document. The term-document matrix represents a t-dimensional space with t-dimensional document vectors. Each vector contains the coordinates of that document's location in the t-dimensional space. Queries are also represented as t-dimensional vectors. The vectors of documents and queries with many terms in common are close together, whereas those with relatively few terms in common are far apart. The query vectors are projected into the term-document matrix using the PSVD.

Even using the most advanced NLA methods, computing the PSVD of a matrix is an extremely expensive process. Because of the tremendous size of modern databases, a term-document matrix can potentially be very large, with hundreds of thousands or even millions of entries. In LSI, this means that most of the processing time is spent in performing the PSVD calculation (Berry et al., 1995a,b). In a rapidly expanding environment, such as the Internet, the term-document matrix is altered often as new documents and terms are added. Recalculating the PSVD of the matrix each time these slight alterations occur is prohibitively expensive. Traditionally, LSI uses a process known as *folding-in* to modify the PSVD. Although this method is very effi-

cient, its accuracy may degrade, especially when word usage patterns change. An efficient and much more reliable approach is to *update* the PSVD, e.g., Zha and Simon (1999). In this approach the existing PSVD is modified to reflect the changes to the term-document matrix; i.e., the PSVD of the modified term-document matrix is obtained by modifying the existing PSVD of the original term-document matrix.

The purpose of this paper is not only to show that updating the PSVD can be more reliable than folding-in but also to show that a combination of folding-in and updating the PSVD (which we call *folding-up*) can be an even more attractive option than either folding-in or updating the PSVD on their own. Folding-up can offer a significant improvement in computation time when compared to either recomputing the PSVD or just updating the PSVD. At the same time, folding-up can provide a level of precision that is not statistically different from that given by recomputing the PSVD each time changes are made to the term-document matrix.

The remainder of the paper proceeds as follows. Section 2 covers background information on the PSVD and on the folding-in process, Section 3 gives a description of the algorithms used for updating the PSVD (Zha and Simon, 1999), Section 4 gives a description of the folding-up process, and Section 5 gives experimental results using the document updating algorithm and the Medline (Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/med), Cranfield (Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/cran), and a subset of TREC HARD track, http://trec.nist.gov/data/t12_hard.html (2003) document collections. Finally, Section 6 presents our conclusions.

2 Background

2.1 SVD

The SVD (see, e.g., (Golub and Van Loan, 1996; Demmel, 1997)) is a matrix factorization that can be used to capture the salient features of a matrix by determining important vectors (directions) and quantifying their importance via weighting factors. Given a matrix $\mathbf{A} \in \Re^{t \times d}$, its SVD is written as $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U} \in \Re^{t \times t}$, $\mathbf{V} \in \Re^{d \times d}$, and $\mathbf{\Sigma} \in \Re^{t \times d}$. $\mathbf{U}$ 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, $\mathbf{U}$ represents the term vectors, and $\mathbf{V}$ represents the document vectors. The matrix $\mathbf{\Sigma}$ can have non-zero entries only on the diagonal. These diagonal entries, denoted σ_j for $j = 1, 2, ..., \min(t, d)$ and arranged in non-increasing order, are known as the singular values of matrix $\mathbf{A}$. The number of non-zero singular values of a matrix is known as its

rank, r. For further details on the SVD, the interested reader is referred to, e.g., (Golub and Van Loan, 1996; Demmel, 1997) and references therein.

The SVD can be interpreted as the weighted sum of r rank-one matrices, $\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$, where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the jth columns of matrices $\mathbf{U}$ and $\mathbf{V}$, respectively. Replacing r in this sum by any k with $0 \leq k < r$ gives the *partial* SVD of $\mathbf{A}$, $\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \approx \mathbf{A}$. In matrix form, this is equivalent to taking $\mathbf{U}_k$ and $\mathbf{V}_k$ to be the first k columns of $\mathbf{U}$ and $\mathbf{V}$, and $\mathbf{\Sigma}_k$ to be the leading $k \times k$ submatrix of $\mathbf{\Sigma}$, yielding $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$. In LSI, the effect of this huge dimensional reduction on the data is a muting of the noise caused by synonymy and an enhancing of the latent patterns that indicate semantically similar terms. This means that $\mathbf{A}_k$ can actually be a better representation of the data than the original term-document matrix. The number of dimensions k to keep in the reduced term-document matrix when d is very large is still open to study and debate, but experiments indicate that values of k between 100 and 300 typically give the best results (Berry et al., 1995b).

2.2 Folding-In

In LSI, when new documents and terms are added to a dataset, it is necessary to modify the PSVD of the term-document matrix to reflect these changes. Because recomputing the PSVD is very expensive, the method of folding-in new documents and terms is often used.

Let $\mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ be the PSVD of the term-document matrix $\mathbf{A} \in \Re^{t \times d}$, where t is the number of terms, d is the number of documents, and k is the number of dimensions used in the PSVD, such that $\mathbf{U}_k \in \Re^{t \times k}$, $\mathbf{\Sigma}_k \in \Re^{k \times k}$, and $\mathbf{V}_k \in \Re^{d \times k}$. Let $\mathbf{D} \in \Re^{t \times p}$ be the term-document matrix that contain the document vectors to be appended to $\mathbf{A}$, where p is the number of new documents. Because we are using the PSVD, $\mathbf{D}$ must be projected into the k-dimensional space, giving $\mathbf{D}_k$: $\mathbf{D}_k = \mathbf{D}^T \mathbf{U}_k \mathbf{\Sigma}_k^{-1}$. The projection $\mathbf{D}_k \in \Re^{p \times k}$ is folded-in to the existing PSVD of $\mathbf{A}$ by appending it to the bottom of $\mathbf{V}_k$, giving the modified matrix $\hat{\mathbf{V}}_k \in \Re^{(d+p) \times k}$. $\mathbf{U}_k$ and $\mathbf{\Sigma}_k$ are not modified in any way with this method. Folding-in documents requires $2ptk$ operations (O'Brien, 1994).

Folding-in terms follows a similar process. Let $\mathbf{T} \in \Re^{q \times d}$ be the term-document matrix containing the term vectors to be appended to $\mathbf{A}$, where q is the number of new terms. $\mathbf{T}$ must be projected into the k-dimensional space, giving $\mathbf{T}_k$: $\mathbf{T}_k = \mathbf{T} \mathbf{V}_k \mathbf{\Sigma}_k^{-1}$. The projection $\mathbf{T}_k \in \Re^{q \times k}$ is folded-in to the existing PSVD of $\mathbf{A}$ by appending it to the bottom of $\mathbf{U}_k$, giving the modified matrix $\hat{\mathbf{U}}_k \in \Re^{(t+p) \times k}$. $\mathbf{V}_k$ and $\mathbf{\Sigma}_k$ are not modified in any way with this method. Folding-in terms requires $2qdk$ operations (O'Brien, 1994).

3 Updating Methods

Updating the PSVD when the term-document matrix changes is a more complicated process than folding-in. However, the end result (in the absence of roundoff errors) is the exact PSVD of the modified term-document matrix without recomputing it from scratch. Typically, the PSVD is updated to reflect the new documents that have been added to the document collection. As with folding-in, adding these new documents often means that new terms must also be added, so the PSVD is updated to reflect these changes. Finally, the PSVD typically needs to be updated as well to reflect the changes to the term weights. The following subsections respectively describe document updating, term updating, and term weight updating. Each method described is based on the updating method introduced in Zha and Simon (1999). This method requires one QR decomposition and one SVD per update; however, these potentially expensive computations are performed on small intermediate matrices, where the computational complexity depends on the size of the update and/or the reduced dimension k but not on the size of the original matrix (see below).

In the following, we let $\mathbf{I}_n$ denote the identity matrix of size n, and we assume that the PSVD of $\mathbf{A}$ is available prior to updating.

3.1 Updating documents

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. 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 $\hat{\mathbf{D}}$, $\mathbf{Q_D R_D} = \hat{\mathbf{D}}$, where $\mathbf{Q_D} \in \Re^{t \times p}$ is orthonormal, and $\mathbf{R_D} \in \Re^{p \times p}$ is upper triangular. Then

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

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 PSVD of $\tilde{\mathbf{A}}$ in k dimensions (the updated PSVD) is

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

This updating procedure has a complexity of $\mathcal{O}(k^3+(d+t)k^2+(d+t)kp+p^3)$.

3.2 Updating terms

Let $\mathbf{T} \in \Re^{q\times d}$ be the term-document matrix containing the term vectors to be appended to $\mathbf{A}$, where q is the number of new terms, and let $\tilde{\mathbf{A}} = [\mathbf{A}; \mathbf{T}]$ be the updated term-document matrix. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{T}} \in \Re^{d\times q} = \left(\mathbf{I_d} - \mathbf{V}_k\mathbf{V}_k^T\right)\mathbf{T}^T$. Form the QR decomposition of $\hat{\mathbf{T}}$, $\mathbf{Q_T}\mathbf{R_T} = \hat{\mathbf{T}}$, where $\mathbf{Q_T} \in \Re^{d\times q}$ is orthonormal, and $\mathbf{R_T} \in \Re^{q\times q}$ is upper triangular. Then

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{T} \end{bmatrix} \approx \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix} [\mathbf{V}_k, \mathbf{Q_T}]^T.$$

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

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{TV}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix}.$$

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

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

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

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

This updating procedure has a complexity of $\mathcal{O}(k^3 + (d+t)k^2 + (d+t)kq + q^3)$.

3.3 Updating term weights

Let $\mathbf{S} \in \Re^{t\times s}$, where s is the number of terms whose term weights are to be adjusted, be a selection matrix in which each column contains one 1, and all other entries are zero. Let $\mathbf{W} \in \Re^{d\times s}$ be the matrix in which each column $\mathbf{W}_i$ contains the difference between the old term weights and the new term weights for term i. Let $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{SW}^T$ be the adjusted term-document matrix. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$.

Let $\hat{\mathbf{S}} \in \Re^{t\times s} = \left(\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T\right)\mathbf{S}$; let $\hat{\mathbf{W}} \in \Re^{d\times s} = \left(\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T\right)\mathbf{W}$.

Form the QR decomposition of $\hat{\mathbf{S}}$ such that $\mathbf{Q_M}\mathbf{R_M} = \hat{\mathbf{S}}$, where $\mathbf{Q_M} \in \Re^{t\times s}$ is orthonormal, and $\mathbf{R_M} \in \Re^{s\times s}$ is upper triangular.

Form the QR decomposition of $\hat{\mathbf{W}}$ such that $\mathbf{Q_N}\mathbf{R_N} = \hat{\mathbf{W}}$, where $\mathbf{Q_N} \in \Re^{d\times s}$ is orthonormal, and $\mathbf{R_N} \in \Re^{s\times s}$ is upper triangular. Then

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{SW}^T$$

$$\approx \mathbf{A}_k + \mathbf{S}\mathbf{W}^T = [\mathbf{U}_k, \mathbf{Q}_\mathbf{M}] \left(\begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R}_\mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R}_\mathbf{N} \end{bmatrix}^T \right) [\mathbf{V}_k, \mathbf{Q}_\mathbf{N}]^T .$$

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

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R}_\mathbf{M} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R}_\mathbf{N} \end{bmatrix}^T .$$

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

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

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

$$\tilde{\mathbf{A}}_k = \left([\mathbf{U}_k, \mathbf{Q}_\mathbf{M}]\, \tilde{\mathbf{U}}_k\right) \tilde{\mathbf{\Sigma}}_k \left([\mathbf{V}_k, \mathbf{Q}_\mathbf{N}]\, \tilde{\mathbf{V}}_k\right)^T .$$

This updating procedure has a complexity of $\mathcal{O}(k^3 + (d+t)k^2 + (d+t)ks + s^3)$.

4 Folding-up

It is well known that folding-in is a very inexpensive way to incorporate new information into an existing term-document matrix compared to recomputing its PSVD (Berry et al., 1995b). However, because the matrices $\mathbf{V}_k$ and $\mathbf{\Sigma}_k$ are never changed, the quality of the results produced by folding-in may deteriorate (perhaps even rapidly) after even only a small number of updates, especially if word usage patterns change significantly. On the other hand, updating the PSVD gives the same result (to within rounding errors) as recomputing the PSVD, but with significantly less computational expense. The use of parallelism could further reduce the computational cost of updating (Berry et al., 2005; Berry and Martin, 2005). We now describe a hybrid updating method, which we call *folding-up*, that uses a combination of folding-in and updating

at each increment in order to reduce the computational expense of updating even further without significantly degrading the results.

The idea behind folding-up is to fold-in documents until the number of folded-in documents reaches a pre-selected percentage of the current term-document matrix. If no updates have previously been done, the current term-document matrix is the initial matrix; otherwise it is the last updated term-document matrix. Once the number of documents that have been folded-in reaches the pre-selected percentage of the current matrix, the vectors that have been appended to $\mathbf{V}_k$ during folding-in are discarded. The PSVD is then updated to reflect the addition of all the document vectors that have been folded-in since the last update. These document vectors are then discarded. The process continues with folding-in until the next update. We note that the choice of the percentage used is empirically based. Further study of measures to determine when to update during the folding-up process is currently in progress.

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 compared to recomputing coupled with the precision advantages of updating. The complexity of the process is of the same order as updating, but reduced by a factor that is dependent on the number of iterations in which updating is replaced by folding-up. We demonstrate by means of examples below that folding-in produces results that are not statistically different from those produced by recomputing the PSVD.

5 Experiments

Examples 1 and 2 in Section 5.1 use the Medline text collection (Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/med), containing 1033 documents and 30 queries. Removing semantically insignificant terms and stemming the remaining terms give a term-document matrix $\mathbf{A}_{\text{Med}} \in \Re^{5735\times1033}$. Examples 3 and 4 in Section 5.2 use the Cranfield text collection (Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/cran), containing 1400 documents and 225 queries. For this collection, no stemming is done, but semantically insignificant words are removed, giving a term-document matrix $\mathbf{A}_{\text{Cran}} \in \Re^{5321\times1400}$. Examples 5 and 6 in Section 5.3 use a subset of the TREC HARD track, http://trec.nist.gov/data/t12_hard.html (2003), with 30,000 documents and 50 queries. Removing semantically insignificant terms and stemming the remaining terms give a term-document matrix $\mathbf{A}_{\text{HARD}} \in \Re^{60,547\times30,000}$. For the Medline and Cranfield text collections, we use a *term frequency inverse document frequency* (tfidf) weighting scheme (Baeza-Yates et al., 1999), and for the HARD collection we use a normalized *term frequency* (tf) weighting scheme. The measure of similarity is the cosine of the angle

between query and document vectors.

For each example, the original term-document matrix is incrementally updated with document vectors until the number of columns has approximately doubled. Because the results from recomputing the PSVD represent the "ideal" result, we compare the final average precision obtained for folding-in, updating, and folding-up with recomputing the PSVD. Statistical comparisons are made pairwise using a non-parametric Kruskal-Wallis test at significance level 0.05. In each case, the average precision for each of the queries at 11 standard recall levels $(0\%, 10\%, \cdots, 100\%)$ is averaged to produce the overall average precision after each increment. For each method used, we plot the average precision at each increment, starting with the initial term-document matrix. All PSVDs are computed using the *Matlab* function `svds`, with $k = 125$ for the Medline collection and $k = 300$ for the Cranfield and HARD collections, where k is the number of singular values and corresponding left and right singular vectors computed. For brevity, the experiments described use only document updating; similar results are produced using term updating.

5.1 Medline Examples

We partition $\mathbf{A}_{\text{Med}} \in \Re^{5735 \times 1033}$ so that the first 533 columns form the initial term-document matrix, and the rest are added incrementally in groups of 10 (Figure 1) and 25 (Figure 2). We compare the average precision for four methods: recomputing the PSVD at each increment, folding-in at each increment, updating at each increment, and folding-up with updates when the number of documents folded-in is approximately 8% of the current matrix size in Figure 1 and 14% in Figure 2, as described in Section 4. Figures 1 and 2 show that the average precision for folding-in deteriorates rapidly relative to recomputing the PSVD; the final average precision is significantly different from that of recomputing the PSVD ($p = 0.02$ in each case).

In Figure 1, the average precision for updating does not begin to deteriorate until the initial matrix is more than one and a half times its original size, and the increments are less than 1.25% of the size of the matrix; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.89$). Although the deterioration is slight, it does indicate that doing many updates that are very small relative to the size of the matrix may eventually have a negative affect on the average precision. However, the savings in computation time compared to recomputing, as shown in Table 1, may more than compensate for this potential deficiency; in this case, updating is more than 100 times faster than recomputing. Figure 1 shows that in this example, folding-up actually outperforms the other methods for much of the time, and the final average precision is not significantly different than recomputing the

SVD ($p = 0.77$); it is also faster than either recomputing or just updating. See Table 1 for a comparison of CPU times.

In Figure 2, the average precision for updating does not deteriorate relative to recomputing the PSVD, and indeed it is at times slightly better; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.84$). These results suggest that updating in larger increments, relative to the size of the matrix, can give better average precision. Again, Table 1 shows that updating the PSVD is much faster than recomputing each time the term-matrix changes, but in these examples, folding-in is by far the fastest method. Folding-up again outperforms the other methods in terms of precision at various points of the experiment; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.88$). It also takes less computation time than recomputing or simply updating.

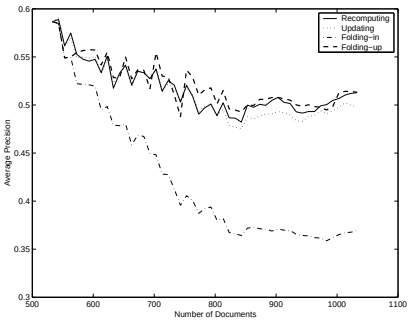

Fig. 1. Average precisions for the Medline collection with 500 documents added in 50 groups of 10.

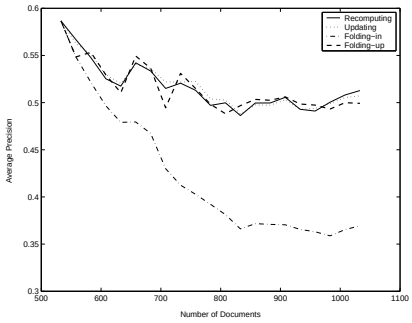

Fig. 2. Average precisions for the Medline collection with 500 documents added in 20 groups of 25.

5.2 Cranfield Examples

We partition $\mathbf{A}_{\text{Cran}} \in \Re^{5321 \times 1400}$ such that the first 700 columns form the initial term-document matrix, and the rest are added incrementally in groups of size 14 (Figure 3) and 28 (Figure 4). We compare the average precision for the four previously described methods, with updates for folding-up when the number of documents folded-in is approximately 8% of the current matrix size for Figure 3 and 14% for Figure 4. These figures show that the average precision for folding-in falls below that of the other methods. For both examples the final average precision for folding-in is significantly different from that of recomputing the PSVD ($p = 0.03$ in each case). We note that the overall average precision is low because no stemming of terms was done when the text collection was processed. The average precisions for recomputing and for updating the PSVD are very similar in Figures 3 and 4, with the final precisions not being significantly different ($p = 0.94$ and $p = 0.88$, respectively), even though Table 1 shows that in each case, updating is more than

150 times faster than recomputing the PSVD. Figures 3 and 4 also show that folding-up performs similarly to both updating and recomputing the PSVD; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.86$ and $p = 0.90$ respectively). Table 1 shows that for Figure 3, folding-up is more than three times faster than updating, and more than 580 times faster than recomputing the PSVD at each increment. Table 1 also shows that for Figure 4, folding-up is almost twice as fast as updating, and more than 580 times faster than recomputing the PSVD at each increment.

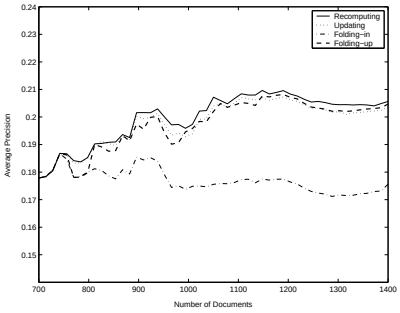

Fig. 3. Average precisions for the Cranfield collection with 700 documents added in 50 groups of 14.

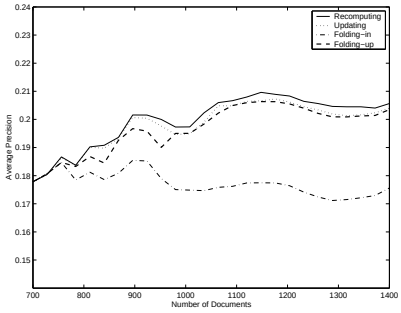

Fig. 4. Average precisions for the Cranfield collection with 700 documents added in 20 groups of 28.

5.3 HARD Examples

We partition $\mathbf{A}_{\text{HARD}} \in \Re^{60547 \times 30000}$ such that the first 15000 columns form the initial term-document matrix, and the rest are added incrementally in groups of size 50 (Figure 5) and 100 (Figure 6). We compare the average precision for the four methods described above, with updates for folding-in when the number of documents folded-in is approximately 4% of the current matrix size.

Figures 5 and 6 show that with this document collection, folding-in does not deteriorate. There is no significant difference between folding-in and recomputing for either example ($p = 0.97$ in each case). This emphasizes the need for a method to determine at what point the folding-up method should use an update. We experimented extensively with using the loss of orthogonality of the singular vectors as a measure of when to update. Unfortunately this method does not work equally well for all document collections; further investigation is on-going.

The average precisions for recomputing and updating the PSVD are very similar in Figures 5 and 6 with the final precisions not being significantly different ($p = 0.96$ and $p = 0.99$, respectively), even though Table 1 shows that in each case, updating is more than 50 times faster than recomputing the PSVD. Figures 5 and 6 also show that folding-up performs similarly to

both updating and recomputing the PSVD; the final average precision is not significantly different from that of recomputing the PSVD ($p = 0.99$ in both cases). Table 1 shows that in Figure 5 folding-up is more than three times faster than updating, and almost 200 times faster than recomputing the PSVD at each increment. Table 1 also shows that in Figure 6 folding-up is more than twice as fast as updating, and more than 130 times faster than recomputing the PSVD at each increment.

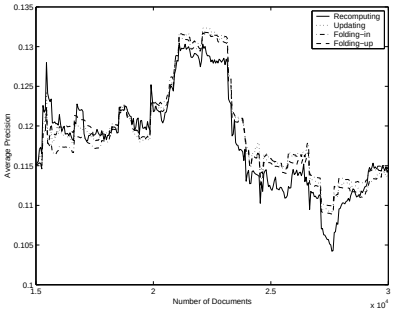

Fig. 5. Average precisions for a subset of the HARD collection with 15000 documents added in 300 groups of 50.

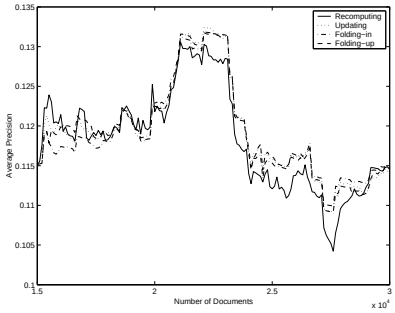

Fig. 6. Average precisions for a subset of the HARD collection with 15000 documents added in 150 groups of 100.

	Medline		Cranfield		HARD	
Increment	10	25	14	28	50	100
Method	CPU	CPU	CPU	CPU	CPU	CPU
Recomputing	5001.60	2045.80	51548.30	26335.33	716392.37	436486.92
Updating	43.07	22.33	294.28	162.11	12486.20	7199.16
Folding-in	1.35	0.75	7.27	4.13	61.96	42.98
Folding-up	15.76	13.14	88.82	81.57	3622.44	3259.36

Table 1
CPU times (seconds) for the Medline collection with 500 documents added in groups of 10 and 25, for the Cranfield collection with 700 documents added in groups of 14 and 28, and for a subset of the HARD collection with 15,000 documents added in groups of 50 and 100.

6 Conclusions

LSI makes heavy use of the PSVD in its implementation. Often, the term-document matrix changes frequently as new documents and terms are added to the data collection. In such cases, it is beneficial to exploit the previously computed PSVD via updating. We have demonstrated that updating the PSVD of the term-document matrix each time these types of changes are made to the

matrix is not only much faster (typically by an order of magnitude) than recomputing the PSVD, but it also gives better average precision than folding-in. We have also demonstrated that folding-up, a new approach that is a hybrid of folding-in and updating, can give better average precision than folding-in, with less computation time (typically by a factor of 2 or 3) than updating alone. Our examples also illustrate a viable method for determining when to update in the folding-up procedure based on the number of documents that are being added as a percentage of the size of the current term-document matrix. The folding-up method offers an excellent speed-up in computation time (typically by a factor of between 20 and 30) with no statistically significant loss of overall average precision compared to recomputing the PSVD.

7 Acknowledgements

The authors wish to express their gratitude to Henry Stern for his help with earlier parts of this work and to Chris Jordan of Dalhousie University for his assistance in preparing the HARD document collection.

References

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

Berry, M. W., Martin, D. I., 2005. Principal component analysis for information retrieval. In: Kontoghiorghes, E. J. (Ed.), Handbook of Parallel Computing and Statistics. Chapman and Hall/CRC.

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

Berry, M. W., Dumais, S. T., O'Brien, G. W., 1995b. Using linear algebra for intelligent information retrieval. SIAM Rev. 37 (4), 573–595.

Berry, M. W., Mezher, D., Philippe, B., Sameh, A., 2005. Parallel algorithms for the singular value decomposition. In: Kontoghiorghes, E. J. (Ed.), Handbook of Parallel Computing and Statistics. Chapman and Hall/CRC.

Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/cran.

Cornell SMART System, ftp://ftp.cs.cornell.edu/pub/smart/med.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., Harshman, R. A., 1990. Indexing by latent semantic analysis. Journal of the American Society of Information Science 41 (6), 391–407.

Demmel, J. W., 1997. Applied numerical linear algebra. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Golub, G. H., Van Loan, C. F., 1996. Matrix Computations, 3rd Edition. Johns

Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD.

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

TREC HARD track, http://trec.nist.gov/data/t12_hard.html, 2003.

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