SPEEDING UP LATENT SEMANTIC ANALYSIS
A Streamed Distributed Algorithm for SVD Updates

Radim Řehůřek
NLP lab, Masaryk University in Brno, Brno, Czech Republic

Keywords: Distributed LSA, Streamed SVD, One-pass SVD, Matrix decomposition, Subspace tracking.

Abstract: Since its inception 20 years ago, Latent Semantic Analysis (LSA) has become a standard tool for robust, unsupervised inference of semantic structure from text corpora. At the core of LSA is the Singular Value Decomposition algorithm (SVD), a linear algebra routine for matrix factorization. This paper introduces a *streamed distributed algorithm for incremental updates*, which allows the factorization to be computed rapidly in a single pass over the input matrix on a cluster of autonomous computers.

1 INTRODUCTION

The purpose of *Latent Semantic Analysis (LSA)* is to find hidden (*latent*) structure in a collection of texts represented in the Vector Space Model (Salton, 1989). LSA was introduced in (Deerwester et al., 1990) and has since become a standard tool in the field of Natural Language Processing and Information Retrieval. At the heart of LSA lies the *Singular Value Decomposition* algorithm, which makes LSA (sometimes also called Latent Semantic Indexing, or LSI) really just another member of the broad family of applications that make use of SVD's robust and mathematically well-founded approximation capabilities[1]. In this way, although we will discuss our results in the perspective and terminology of LSA and Natural Language Processing, our results are in fact applicable to a wide range of problems and domains across much of the field of Computer Science.

1.1 Singular Value Decomposition

Latent Semantic Analysis assumes that each document (*observation*) can be described using a fixed set of real-valued features (*variables*). These features capture the usage frequency of distinct words in the document, and are typically re-scaled by some TF-IDF scheme (Salton, 1989). However, no assumption is made on what the particular features are or how to extract them from raw data—LSA simply represents the input data collection of n documents, each described by m features, as a matrix $A \in \mathbb{R}^{m \times n}$, with documents as columns and features as rows.

In theory, the term-document matrix A can be directly mapped to physical memory and processed. However, A is commonly left implicit, as it is often distributed among many computers, unknown in its entirety and/or computed partially on-demand. Indeed, in many cases the implied matrix is even infinite in size, gradually growing as the collection grows.

Nevertheless, there is a reason why it is worthwhile to view the collection as a single gigantic matrix. It allows us to consider linear algebra decomposition algorithms that provide succint, mathematically well-founded ways of discovering latent structure of the matrix and, thereby, of the original data collection. In case of LSA, the algorithm of choice is the *Singular Value Decomposition, SVD*. Assuming $n \gg m$, the SVD of A yields $A_{m\times n} = U_{m\times m} S_{m\times m} V^T_{m\times n}$, where U, V are orthogonal matrices (called *left* and *right singular vectors*) and S is a diagonal matrix of *singular values* with diagonal entries in decreasing order.

Here and elsewhere, we use the subscripts $A_{x\times y}$ to denote that matrix A has x rows and y columns but omit the subscripts whenever there is no risk of confusion.

1.2 Related Work

Historically, most research on SVD optimization has gone into Krylov subspace methods, such as Lanczos-

[1] Another member of that family is the *discrete KarhunenLove Transform*, from Image Processing; or Signal Processing, where SVD is commonly used to separate signal from noise. SVD is also used in solving shift-invariant differential equations, in Geophysics, in Antenna Array Processing, …

Řehůřek R..
SPEEDING UP LATENT SEMANTIC ANALYSIS - A Streamed Distributed Algorithm for SVD Updates.
DOI: 10.5220/0003191304460451
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 446-451
ISBN: 978-989-8425-40-9

based iterative solvers (see e.g. (Vigna, 2008) for a recent large-scale SVD effort). Our problem is, however, different in that we can only afford a *single pass* over the input corpus (iterative solvers require $O(k)$ passes in general). Our scenario, where the decomposition must be updated on-the-fly and in constant memory, as the stream of observations cannot be repeated or even stored in off-core memory, can be viewed as an instance of *subspace tracking*. See (Comon and Golub, 1990) for an excellent overview on the complexity of various forms of matrix decomposition algorithms in the context of subspace tracking.

An explicitly formulated $O(m(k+c)^2)$ method (where c is the number of newly added documents) for incremental LSA updates is presented in (Zha and Simon, 1999). They also give formulas for updating rows of A as well as rescaling row weights. Their algorithm is completely streamed and runs in constant memory. It can therefore also be used for online subspace tracking, by simply ignoring all updates to the right singular vectors V. The complexity of updates was further reduced in (Brand, 2006) who proposed a linear $O(mkc)$ update algorithm by a series of c fast rank-1 updates. However, in the process, the ability to track subspaces is lost (despite their tentative claim to the contrary). Their approach is akin to the *k-Nearest Neighbours (k-NN)* method of Machine Learning: the lightning speed during training is offset by memory requirements of storing the intermediate model.

These incremental methods concern themselves with adding new documents to an existing decomposition. What is needed for a distributed version of LSA is a slightly different task: given two existing decompositions, merge them together into one. We did not find any explicit, efficient algorithm for merging decompositions in the literature. In this research paper we therefore seek to close this gap, provide such algorithm and use it for computing distributed LSA. The following section describes the algorithm and states conditions under which the merging makes sense when dealing with only truncated rank-k approximation of the decomposition.

2 DISTRIBUTED LSA

In this section, we derive an algorithm for distributed computing of LSA over a cluster of autonomous computers.

2.1 Algorithm Overview

Parallel computation will be achieved by column-partitioning the input matrix A into several smaller submatrices, called *jobs*, $A^{m\times n} = \left[A_1^{m\times c_1}, A_2^{m\times c_2}, \cdots, A_j^{m\times c_j}\right]$, $\sum c_i = n$. Since columns of A correspond to documents, each job A_i amounts to processing a chunk of c_i input documents. The sizes of these chunks are chosen to fit available resources of the processing nodes: bigger chunks mean faster overall processing but on the other hand consume more memory.

Jobs are then distributed among the available cluster nodes, in no particular order, so that each node will be processing a different set of column-blocks from A. The nodes need not process the same number of jobs, nor process jobs at the same speed; the computations are completely asynchronous and independent. Once all jobs have been processed, the decompositions accumulated in each node will be merged into a single, final decomposition $P = (U, S)$.

What is needed are thus two algorithms:

1. Find $P_i = (U_i^{m\times c_i}, S_i^{c_i\times c_i})$ eigen decomposition of a single job $A_i^{m\times c_i}$ such that $A_iA_i^T = U_iS_i^2U_i^T$. These P_i decompositions will form the base case for merging.
2. Merge two decompositions $P_i = (U_i, S_i)$, $P_j = (U_j, S_j)$ of two jobs A_i, A_j into a single decomposition $P = (U, S)$ such that $\left[A_i, A_j\right]\left[A_i, A_j\right]^T = US^2U^T$.

We'd like to highlight the fact that the first algorithm will perform decomposition of a *sparse* input matrix; the second algorithm will merge two *dense* decompositions into another dense decomposition. This is in contrast to incremental updates discussed in the literature (Levy and Lindenbaum, 2000; Zha and Simon, 1999; Brand, 2006), where the existing decomposition and the new documents are mashed together into a single matrix, losing any potential benefits of sparsity.

2.2 Solving the Base Case

In LSA (using the same notation introduced in Section 1), the truncation factor k is typically in the hundreds or thousands, the number of features m between 10^4 to 10^6 and the number of documents n tends to infinity, so that $k \ll m \ll n$. Density of the job matrices is well below 1%, so a sparse solver is called for, that makes efficient use of the sparse matrix structure. Also, a direct sparse SVD solver of A_i is preferable to the roundabout eigen decomposition of $A_iA_i^T$, for memory-conserving as well as numerical accuracy reasons (see e.g. (Golub and Van Loan, 1996)). Finally, because $k \ll m$, a partial decomposition is required which only returns the k greatest factors—computing the full spectrum would be a terrible overkill.

For these reasons, we solve base cases with a "black-box" in-core sparse partial SVD algorithm.

2.3 Merging Decompositions

No explicit algorithm (as far as we know) exists for merging two truncated eigen decompositions (or SVD decompositions) into one. We therefore propose our own, novel algorithm here, starting with its derivation and summing up the final version in the end.

The problem can be stated as follows. Given two truncated eigen decompositions $P_1 = (U_1^{m\times k_1}, S_1^{k_1\times k_1})$, $P_2 = (U_2^{m\times k_2}, S_2^{k_2\times k_2})$, which come from the (by now lost and unavailable) input matrices $A_1^{m\times c_1}$, $A_2^{m\times c_2}$, $k_1 \leq c_1$ and $k_2 \leq c_2$, find $P = (U,S)$ that is the eigen decomposition of $\left[A_1, A_2\right]$.

Our first approximation will be the direct naive

$$U,S^2 \xleftarrow{eigen} \left[U_1S_1, U_2S_2\right]\left[U_1S_1, U_2S_2\right]^T. \qquad (1)$$

This is terribly inefficient, and forming the matrix product of size $m\times m$ on the right hand side is prohibitively expensive. Writing SVD$_k$ for truncated SVD that returns only the k greatest singular numbers and their associated singular vectors, we can equivalently write:

Algorithm 1: SVD merge.

```
Input: Truncation factor $k$, Decay factor $\gamma$,
        $P_1 = (U_1^{m\times k_1}, S_1^{k_1\times k_1})$, $P_2 = (U_2^{m\times k_2}, S_2^{k_2\times k_2})$
Output: $P = (U^{m\times k}, S^{k\times k})$
1 $U,S,V^T \xleftarrow{SVD_k} \left[\gamma U_1S_1, U_2S_2\right]$
```

This is more reasonable, with the added bonus of increased numerical accuracy over the related eigen decomposition. Note, however, that the computed right singular vectors V^T are not used at all, which is a sign of further inefficiency. This leads us to break the algorithm into several steps:

Algorithm 2: QR merge.

```
Input: Truncation factor $k$, Decay factor $\gamma$,
        $P_1 = (U_1^{m\times k_1}, S_1^{k_1\times k_1})$, $P_2 = (U_2^{m\times k_2}, S_2^{k_2\times k_2})$
Output: $P = (U^{m\times k}, S^{k\times k})$
1 $Q,R \xleftarrow{QR} \left[\gamma U_1S_1, U_2S_2\right]$
2 $U_R,S,V_R^T \xleftarrow{SVD_k} R$
3 $U^{m\times k} \leftarrow Q^{m\times(k_1+k_2)}U_R^{(k_1+k_2)\times k}$
```

On line 1, an orthonormal subspace basis Q is found which spans both of the subspaces defined by columns of U_1 and U_2, span(Q) = span$(\left[U_1, U_2\right])$. Multiplications by S_1, S_2 and γ provide scaling for R only and do not affect Q in any way, as Q will always be orthogonal (with columns of unit length). Our algorithm of choice for constructing the new basis is QR factorization, because we can use its other product, the upper trapezoidal matrix R, to our advantage. Now we're almost ready to declare (Q,R) our target decomposition (U,S), except R is not diagonal. To diagonalize R, we perform an SVD on it, on line 2. This gives us the singular values S we need as well as the rotation of Q necessary to represent the basis in this new subspace. The rotation is applied on line 3. Finally, both output matrices are truncated to the requested rank k. The costs are $O(m(k_1+k_2)^2)$, $O((k_1+k_2)^3)$ and $O(m(k_1+k_2)^2)$ for line 1, 2 and 3 respectively, for a combined total of $O(m(k_1+k_2)^2)$.

Although more elegant than Algorithm 1, the baseline algorithm is only marginally more efficient than a direct SVD. This comes as no surprise, as the two algorithms are quite similar and SVD of rectangular matrices is often internally implemented by means of QR in exactly this way. Luckily, we can do better.

First, we observe that the QR decomposition makes no use of the fact that U_1 and U_2 are already orthogonal. Capitalizing on this will allow us to represent U as an update to the existing basis U_1, $U = \left[U_1, U'\right]$, dropping the complexity of the first step to $O(mk_2^2)$. Secondly, the application of rotation U_R to U can be rewritten as $UU_R = \left[U_1, U'\right]U_R = U_1R_1 + U'R_2$, dropping the complexity of the last step to $O(mkk_1 + mkk_2)$. Plus, the algorithm can be made to work by modifying the existing matrices U_1, U_2 in place inside BLAS routines, which is a considerable practical improvement over Algorithm 2, which requires allocating additional $m(k_1+k_2)$ floats.

Algorithm 3: Optimized merge.

```
Input: Truncation factor $k$, Decay factor $\gamma$,
        $P_1 = (U_1^{m\times k_1}, S_1^{k_1\times k_1})$, $P_2 = (U_2^{m\times k_2}, S_2^{k_2\times k_2})$
Output: $P = (U^{m\times k}, S^{k\times k})$
1 $Z^{k_1\times k_2} \leftarrow U_1^TU_2$
2 $U',R \xleftarrow{QR} U_2 - U_1Z$
3 $U_R,S,V_R^T \xleftarrow{SVD_k} \begin{bmatrix} \gamma S_1 & ZS_2 \\ 0 & RS_2 \end{bmatrix}^{(k_1+k_2)\times(k_1+k_2)}$
4 $\begin{bmatrix} R_1^{k_1\times k} \\ R_2^{k_2\times k} \end{bmatrix} = U_R$
5 $U \leftarrow U_1R_1 + U'R_2$
```

The first two lines construct orthonormal basis U' for the component of U_2 that is orthogonal to U_1,

$$\operatorname{span}(U') = \operatorname{span}((I - U_1 U_1^T) U_2) \tag{2}$$
$$= \operatorname{span}(U_2 - U_1(U_1^T U_2)). \tag{3}$$

As before, we use QR factorization because the upper trapezoidal matrix R will come in handy when determining the singular vectors S.

Line 3 is perhaps the least obvious, but follows from the requirement that the updated basis $[U, U']$ must satisfy

$$[U_1 S_1, U_2 S_2] = [U_1, U'] X, \tag{4}$$

so that

$$X = [U_1, U']^T [U_1 S_1, U_2 S_2]. \tag{5}$$

By multiplying,

$$\begin{bmatrix} U_1^T \\ U'^T \end{bmatrix} [U_1 S_1, U_2 S_2] = \begin{bmatrix} U_1^T U_1 S_1 & U_1^T U_2 S_2 \\ U'^T U_1 & U'^T U_2 S_2 \end{bmatrix} \tag{6}$$

and using the equalities $R = U'^T U_2$, $U'^T U_1 = 0$ and $U_1^T U_1 = I$ (all by construction) we obtain

$$X = \begin{bmatrix} S_1 & U_1^T U_2 S_2 \\ 0 & U'^T U_2 S_2 \end{bmatrix} = \begin{bmatrix} S_1 & Z S_2 \\ 0 & R S_2 \end{bmatrix}. \tag{7}$$

Line 4 is just a way of saying that on line 5, U_1 will be multiplied by the first k_1 rows of U_R, while U' will be multiplied by the remaining k_2 rows.

Finally, line 5 seeks to avoid realizing the full $[U_1, U']$ matrix in memory and is a direct application of the equality

$$[U_1, U']^{m \times (k_1 + k_2)} U_R^{(k_1 + k_2) \times k} = U_1 R_1 + U' R_2. \tag{8}$$

As for complexity of this algorithm, it is again dominated by the matrix products and the dense QR factorization, but this time only of a matrix of size $m \times k_2$. The SVD of line 3 is a negligible $O(k_1 + k_2)^3$, and the final basis rotation comes up to $O(mk \max(k_1, k_2))$. Overall, with $k_1 \approx k_2 \approx k$, this is an $O(mk^2)$ algorithm.

2.4 Effects of Truncation

While the equations above are provably correct when using matrices of full rank (putting aside the technical question of gradual loss of accuracy due to finite machine precision and rounding errors for now), it is not at all clear how to justify truncating all intermediate matrices to rank k in each update. What effect does this have on the merged decomposition? How do these effects stack up as we perform several updates in succession?

In (Zha and Zhang, 2000), the authors did the hard work and identified conditions under which operating with truncated matrices produces exact results. Moreover, they show by way of perturbation analysis that the results are stable (though no longer exact) even if the input matrix only approximately satisfies this condition. They investigate matrices of the so-called *low-rank-plus-shift* structure, which (approximately) satisfy

$$A^T A / m \approx C W C^T + \sigma^2 I_n, \tag{9}$$

where C, W are of rank k, W positive semi-definite, σ^2 the variance of noise. That is, $A^T A$ can be expressed as a sum of a low-rank matrix and a multiple of the identity matrix. They show that matrices coming from natural language corpora do indeed possess this structure and that in this case, a rank-k approximation of A can be expressed as a combination of rank-k approximations of its submatrices $A = [A_1, A_2]$ without a serious loss of precision.

2.5 Putting It Together

Let $N = \{N_1, \ldots, N_p\}$ be the p available cluster nodes. Each node will be processing incoming jobs sequentially, running the base case decomposition from Section 2.2 followed by merging the result with the current decomposition by means of Algorithm 1, 2 or 3:

```
Algorithm 4: LSA Node.
    Input: Truncation factor $k$, Queue of jobs $A_1, A_2, \ldots$
    Output: $P = (U^{m \times k}, S^{k \times k})$ decomposition of
            $[A_1, A_2, \ldots]$
1   $P = (U, S) \leftarrow \mathbf{0}^{m \times k}, \mathbf{0}^{k \times k}$
2   foreach job $A_i$ do
3       $P' = (U', S') \leftarrow$ BaseCase_Decomposition$(k, A_i)$
4       $P \leftarrow$ Merge_Algo$\{1,2,3\}(k, P, P')$
5   end
```

To construct decomposition of the full matrix A, we let the nodes work in parallel, distributing the jobs as soon as they arrive, to whichever node seems idle. We do not describe the technical issues of load balancing and recovery from node failure here, but standard practices apply. Once we have processed all the jobs (or temporarily exhausted the input job queue, in the infinite streaming scenario), we merge their individual results into one:

```
Algorithm 5: Distributed LSA.
    Input: Truncation factor $k$, Queue of jobs
           $A = [A_1, A_2, \ldots]$
    Output: $P = (U^{m \times k}, S^{k \times k})$ decomposition of $A$
1   $P_i = (U_i, S_i) \leftarrow$ LSA_Node_Algo4($k$, subset of jobs
    from $A$), for $i = 1, \ldots, p$
2   $P \leftarrow$ Reduce(Merge_Algo, $[P_1, \ldots, P_p]$)
```

Here, line 1 is executed in parallel, making use of all p nodes at once, and is the source of parallelism of the algorithm. On line two, *Reduce* applies the function that is its first argument cummulatively to the sequence that is its second argument, so that it effectively merges P_1 with P_2, followed by merging that result with P_3, etc.

We note that other divide-and-conquer schemes are also possible, such as one where the merging in line 2 of Algorithm 5 happens on pairs of decompositions coming from approximately the same number of input documents, so that the two sets of singular values are of comparable magnitude. Doing so could lead to improved numerical properties, but we have not investigated this effect yet.

The algorithm is formulated in terms of a (potentially infinite) sequence of jobs, so that when more jobs arrive, we can continue updating the decomposition in a natural way. The whole algorithm can in fact act as a continuous daemon service, providing decomposition of all the jobs processed so far on demand.

3 EXPERIMENTS

In this section, we describe a set of experiments measuring numerical accuracy of the proposed algorithm.

In all experiments, the decay factor γ is set to 1.0, that is, there is no discounting in favour of new observation. The number of requested factors is $k = 200$ in all cases.

3.1 Set Up

We will be comparing four implementations for partial Singular Value Decomposition:

SVDLIBC A direct sparse SVD implementation due to Douglas Rohde[2]. SVDLIBC is based on the SVDPACK package by Michael Berry (Berry, 1992). We use the LAS2 routine (Lanczos of the related implicit A^TA or AA^T matrix with selective orthogonalizations) to retrieve only the k dominant singular triplets. The implementation works in-core and therefore doesn't scale.

ZMS Implementation of the incremental one-pass algorithm from (Zha et al., 1998). The right singular vectors and their updates are completely ignored so that our implementation of their algorithm also realizes subspace tracking.

DLSA Our proposed method. We will be evaluating three different versions of merging, Algorithms 1, 2 and 3, calling them $DLSA_1$, $DLSA_2$ and $DLSA_3$ in the results, respectively. Sparse SVD during base case decomposition is realized by an adapted LAS2 routine from SVDLIBC, see above.

With the exception of SVDLIBC, all the other algorithms operate in a streaming fashion (ehek and Sojka, 2010), so that the corpus need not reside in core memory all at once. Although memory footprint of all algorithms is independent of the size of the corpus, it is still linear in the number of features, $O(m)$. It is assumed that the decomposition $(U^{m\times k}, S^{k\times k})$ fits entirely into core memory.

For the experiments, we will be using a corpus of 3,494 mathematical articles collected from the digital libraries of NUMDAM, arXiv and DML-CZ. After the standard procedure of pruning out word types that are too infrequent (hapax legomena, typos, OCR errors, etc.) or too frequent (stop words), we are left with 39,022 distinct features. The final matrix $A^{39,022\times 3,494}$ has 1.5 million non-zero entries. This corpus was chosen so that it fits into core memory of a single computer and its decomposition can therefore be computed directly. This will allow us to establish the "ground-truth" decomposition and set an upper bound on achievable accuracy and speed.

3.2 Accuracy

Figure 1 plots the relative accuracy of singular values found by DLSA, ZMS, SVDLIBC and HEBB algorithms compared to known, "ground-truth" values S_G. We measure accuracy of the computed singular values $\overline{S}$ as $r\_i = |\overline{s\_i} - s\_G\_i|/s\_G\_i$, for $i = 1, \dots, k$. The ground-truth singular values S_G are computed directly with LAPACK's DGESVD routine.

We observe that the largest singular values are practically always exact, and accuracy quickly degrades towards the end of the returned spectrum. This leads us to the following refinement: When requesting x factors, compute the truncated updates for $k > x$, such as $k = 2x$, and discard the extra $x - k$ factors only when the final projection is actually needed. The error is then below 5%, which is comparable to the ZMS algorithm (while DLSA is at least an order of magnitude faster even without any parallelization).

4 CONCLUSIONS

We developed and presented a novel single-pass eigen decomposition method, which runs in constant memory w.r.t. the number of observations. The method is embarrassingly parallel, so we also give its distributed version.

[2]http://tedlab.mit.edu/∼dr/SVDLIBC/

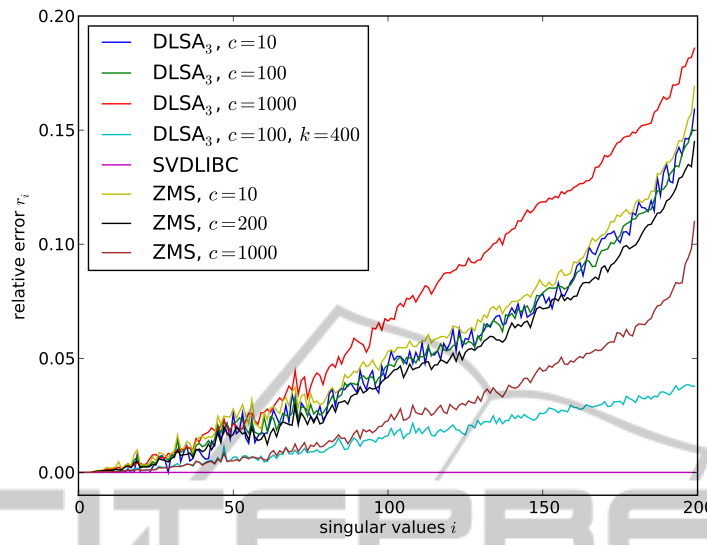

Figure 1: Accuracy of singular values for various decomposition algorithms.

This method is suited for processing extremely large (possibly infinite) sparse matrices that arrive as a stream of observations, where each observation must be immediately processed and then discarded. It is therefore best suitable for environments where the input stream cannot be repeated and must be processed in constant memory.

Future work will include a more thorough evaluation of the algorithm's accuracy and performance, with a side-by-side comparison to other decomposition algorithms.

ACKNOWLEDGEMENTS

This study was partially supported by the LC536 grant of MMT CR.

REFERENCES

Berry, M. (1992). Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications*, 6(1):13–49.

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

Comon, P. and Golub, G. (1990). Tracking a few extreme singular values and vectors in signal processing. *Proceedings of the IEEE*, 78(8):1327–1343.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American society for information science*, 41(6):391–407.

ehek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of LREC 2010 workshop on New Challenges for NLP Frameworks*, pages 45–50.

Golub, G. and Van Loan, C. (1996). *Matrix computations*. Johns Hopkins University Press.

Levy, A. and Lindenbaum, M. (2000). Sequential Karhunen–Loeve basis extraction and its application to images. *IEEE Transactions on Image processing*, 9(8):1371.

Salton, G. (1989). Automatic text processing: the transformation, analysis, and retrieval of information by computer. *Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA*.

Vigna, S. (2008). Distributed, large-scale latent semantic analysis by index interpolation. In *Proceedings of the 3rd international conference on Scalable information systems*, pages 1–10. ICST.

Zha, H., Marques, O., and Simon, H. (1998). Large-scale SVD and subspace-based methods for Information Retrieval. *Solving Irregularly Structured Problems in Parallel*, pages 29–42.

Zha, H. and Simon, H. (1999). On updating problems in Latent Semantic Indexing. *SIAM Journal on Scientific Computing*, 21:782.

Zha, H. and Zhang, Z. (2000). Matrices with low-rank-plus-shift structure: Partial SVD and Latent Semantic Indexing. *SIAM Journal on Matrix Analysis and Applications*, 21:522.