

CLSI: A Flexible Approximation Scheme from Clustered Term-Document Matrices^{*†}

Dimitrios Zeimpekis[‡]

Efstratios Gallopoulos[‡]

Abstract

We investigate a methodology for matrix approximation and IR. A central feature of these techniques is an initial clustering phase on the columns of the term-document matrix, followed by partial SVD on the columns constituting each cluster. The extracted information is used to build effective low rank approximations to the original matrix as well as for IR. The algorithms can be expressed by means of rank reduction formulas. Experiments indicate that these methods can achieve good overall performance for matrix approximation and IR and compete well with existing schemes.

Keywords: Low rank approximations, Clustering, LSI.

1 Introduction and motivation

The purpose of this paper is to outline aspects of a framework for matrix approximation and its application in LSI¹. This framework is designed in the context of the vector space model [18], where a collection of n documents is represented by a term-document matrix (abbreviated as tdm) of n columns and m rows, where m is the number of terms (or phrases) used to index the collection. Each element α_{ij} of the tdm “measures the importance” of term i w.r.t. the document and the entire collection. One of the most successful VSM based models is LSI, whose effective implementation requires the singular value decomposition (SVD) and other matrix operations. Well known disadvantages of LSI are the cost of the kernel SVD, the difficulty of a priori selection of the approximation rank and the handling of updates. Such topics continue to challenge researchers. Interesting proposals to address these issues include special decompositions (e.g. [14, 13, 22, 19, 3, 5, 15]) and randomized techniques (cf. [1, 8, 7, 9]). This note outlines methods for matrix approximation and IR that are based on a combination of clustering, partial SVD on cluster blocks and recombination to achieve fast and accurate matrix approximation. Our approach can be viewed as a generalization of LSI (hence our

term Clustered LSI or CLSI for short) and is shown, experimentally, to achieve good performance compared to SVD and other matrix approximation methods as well as in terms of precision vs. LSI.

2 Low rank matrix approximations

The central idea here is to obtain, after preprocessing, a matrix of reduced dimension from the original data matrix subsequently used a) to approximate the underlying matrix; b) to enable information retrieval that is competitive with existing methods; c) to enable the use of computing resources that offer large scale, large grain parallelism. This latter item is not elaborated further as it is under current investigation. We first address issue (a). The method relies on the following sequence of steps:

[tdm clustering] \searrow
(partial svd on blocks) \searrow
(projected approximation)

Methods that employ clustering for approximation and have motivated us include [6, 17]. We also use the Guttman formula (see [11], [5, 16, 12] and references therein).

Approximation with partial SVD Denote by A_j the set of columns of A that were found to belong to cluster j , that is write $[A_1, \dots, A_k] = AT$, where T is a permutation matrix that achieves the ordering corresponding to the output of the clustering algorithm. The next step of the algorithms in [6, 17] is to construct the “centroid matrix” $C = [c^{(1)}, \dots, c^{(k)}] \in \mathbb{R}^{m \times k}$, so called because each column $c^{(j)}$ is the centroid of the j^{th} cluster. The final, crucial step of the centroid algorithm is to compute the closest approximation of A of the form CY where $Y \in \mathbb{R}^{k \times n}$, that is solve $\min_{Y \in \mathbb{R}^{k \times n}} \|A - CY\|_F$. This is done by computing the orthogonal projection of A on the subspace generated by the columns of C . Instead of this approach, we approximate the “topic” subspace by computing the left singular vector corresponding to the maximum singular value (let’s call it maximum left singular vector) for each one of the submatrices $A_j \in \mathbb{R}^{m \times n_j}$ containing the data columns corresponding to cluster j for $j = 1, \dots, k$ so that $\sum_{j=1}^k n_j = n$. Let now $u_1^{(j)}$ be the maximum left singular vector of A_j and $\mathcal{U} = [u_1^{(j)} \dots u_1^{(k)}]$. Then we can seek an approximation of

^{*}This work was conducted in the context of and supported in part by a University of Patras KARATHEODORI grant. The first author was also supported in part by a Bodossaki Foundation graduate fellowship.

[†]E-mails: [dsz, stratis]@hpclab.ceid.upatras.gr

[‡]Computer Engineering & Informatics Dept., Univ. of Patras, Greece.

¹See <http://scgroup.hpclab.ceid.upatras.gr> for more detailed treatment and more complete references.

the data matrix A by $\mathcal{U}X$ where

$$(2.1) \quad X = \arg \min_{Y \in \mathbb{R}^{m \times k}} \|A - \mathcal{U}Y\|_F.$$

As in the Centroids Method ([16]), we can find the solution as $\tilde{A} = \mathcal{U}(\mathcal{U}^\top \mathcal{U})^{-1} \mathcal{U}^\top A$. The columns of \mathcal{U} are not orthogonal, in general, though $G = \mathcal{U}^\top \mathcal{U}$ is oblique. We generalize the above idea and consider, for better approximation, using more leading left singular vectors from each cluster block A_j . Let the number of cluster blocks be $l < k$. We solve (2.1), where matrix \mathcal{U} becomes

$$(2.2) \quad \begin{aligned} \mathcal{U} &= [u_1^{(1)}, \dots, u_{k_1}^{(1)}, u_1^{(2)}, \dots, u_{k_2}^{(2)}, \dots, u_1^{(l)}, \dots, u_{k_l}^{(l)}] \\ &= [U_{k_1}^{(1)}, \dots, U_{k_l}^{(l)}], \quad U_{k_j}^{(j)} \in \mathbb{R}^{m \times k_j}, \end{aligned}$$

and $\{u_1^{(j)}, \dots, u_{k_j}^{(j)}\}$ are the k_j leading left singular vectors of A_j . We reserved k for the total number of columns of \mathcal{U} rather than for the number of clusters. To fully specify the algorithm we need a strategy for selecting the k_j 's. We would like to select these values so that A is not too far from the subspace spanned by the columns of \mathcal{U} . More precisely, we want to estimate the number of singular triplets that are necessary for each submatrix so as to have similar (and small) approximation error for all submatrices. Because of the relation between the Frobenius norm and singular values, $\|A\|_F^2 = \sum_{i=1}^l \|A_i\|_F^2 = \sum_{i=1}^l \sum_{j=1}^{r_i} (\sigma_j^{(i)})^2$ we use the ratio

$$(2.3) \quad f(\|A_i\|_F) = \frac{\|A_i\|_F}{\|A\|_F}.$$

as gauge for the number of singular vectors to use from each submatrix. It is worth noting that a strategy based on the ratio of the Frobenius norms of the columns of A vs. A_i was key to the implementation of a recent randomized SVD algorithm [7]. Note that the described heuristic is one of many possibilities. Related ideas are used in Image Processing and Compression. In particular, k_i is selected to be the minimum of the integer nearest to $k f(\|A_i\|_F)$ and the number of columns of A_i . We would be calling the special case of the above algorithm, when $k_j = 1$ for $j = 1, \dots, l$ and $l = k$, as Algorithm 1 and the more general case we just described Algorithm 2; see Table 1.

Rank reduction framework The general result we use is the following:

THEOREM 2.1. (GUTTMAN) Let $A \in \mathbb{R}^{m \times n}$, $X \in \mathbb{R}^{n \times k}$, $Y \in \mathbb{R}^{m \times k}$. Then:

$$\text{rank}(A - AX\Omega^{-1}Y^\top A) = \text{rank}(A) - k$$

if and only if $\Omega = Y^\top AX$ is nonsingular.

If we set $\mathcal{P} = I - AX\Omega^{-1}Y^\top$, it is straightforward to show that $\mathcal{P}^2 = \mathcal{P}$ though, in general $\mathcal{P} \neq \mathcal{P}^\top$. Therefore, \mathcal{P} is an oblique projector and since

$$A - AX\Omega^{-1}Y^\top A = \mathcal{P}A,$$

Input: $m \times n$ tdm A , number of clusters l , integer k

Output: Matrices $X \in \mathbb{R}^{m \times k}$, $Y \in \mathbb{R}^{k \times n}$

I. Run selected clustering algorithm for l clusters;

Reorder the tdm $A = [A_1 \ A_2 \ \dots \ A_l]$;

II. Let $k_i = \min(\text{round}(kf(\|A_i\|_F)), n_i)$;

Compute leading left singular vectors for each A_i :

$$\mathcal{U} = [u_1^{(1)}, \dots, u_{k_1}^{(1)}, u_1^{(2)}, \dots, u_{k_2}^{(2)}, \dots, u_1^{(l)}, \dots, u_{k_l}^{(l)}]$$

III: Solve $\min_Y \|A - \mathcal{U}Y\|_F$: $Y = (\mathcal{U}^\top \mathcal{U})^{-1} (\mathcal{U}^\top A)$;

Compute $\tilde{A} = \mathcal{U}(\mathcal{U}^\top \mathcal{U})^{-1} \mathcal{U}^\top A$;

Table 1: Algorithm 2.

rank reduction follows by the oblique projection of the columns of A . Algorithms 1-2 can be expressed in this framework. We do this here for the former. Let

$$(2.4) \quad S = \text{diag}([\sigma_1^{(1)}, \dots, \sigma_1^{(k)}]) \in \mathbb{R}^{k \times k},$$

and $\mathcal{V} \in \mathbb{R}^{n \times k}$ the block diagonal matrix where each diagonal "block" is the vector $v_1^{(j)}$, $j = 1, \dots, k$. \mathcal{U} is not orthogonal in general, hence the decomposition is not an SVD. Therefore

$$E = A - \tilde{A} = A - \mathcal{A}\mathcal{V}S(\mathcal{U}^\top \mathcal{A}\mathcal{V}S)^{-1} \mathcal{U}^\top A.$$

This is the same as the rank reduction formula shown in Theorem 2.1 above with $X = \mathcal{V}S$ and $Y = \mathcal{U}$. The above can be extended to Algorithm 2.

Clustered LSI (CLSI). CLSI can be viewed as a block version of LSI. Whereas LSI achieves dimensional reduction by projecting on a number of the leading left singular vectors of A , say k , in CLSI we use k orthogonal vectors for the subspace spanned by a total of k vectors selected from the leading singular vectors of the l cluster blocks; in particular, the algorithm uses k_j leading vectors from each block $j = 1, \dots, l$. Irrespective of the strategy employed to compute the k_j 's, we enforce $k = \sum_{j=1}^l k_j$. Actually, CLSI is a family of methods; e.g. when $k_j = 1$, CLSI corresponds to Alg. 1, and when $l = 1$, $k_1 = k$ (in which case clustering phase I is redundant) it corresponds to LSI. CLSI represents documents by means of the columns of $(\mathcal{U}^\top \mathcal{U})^{-1} \mathcal{U}^\top A \in \mathbb{R}^{k \times n}$. In practice, it is sufficient to construct an orthogonal basis $Q_{\mathcal{U}}$ for \mathcal{U} . Assuming that $Q_{\mathcal{U}}R_{\mathcal{U}} = \mathcal{U}$, where $R_{\mathcal{U}} \in \mathbb{R}^{k \times k}$, it immediately follows that documents are equivalently represented by $R_{\mathcal{U}}(R_{\mathcal{U}}^\top R_{\mathcal{U}})^{-1} R_{\mathcal{U}}^\top Q_{\mathcal{U}}^\top A = Q_{\mathcal{U}}^\top A$. Similarly, any query q is projected by means of its coefficients with respect to the basis $Q_{\mathcal{U}}$, that is by $Q_{\mathcal{U}}^\top q$. To save space we do not list CLSI but note that it is similar to Algorithm 2. The differences are in phase II, where the scheme for selecting the parameters k_i is not specified but left to the user and in phase III, where we explicitly construct an orthogonal basis $Q_{\mathcal{U}}$ for $\text{range}(\mathcal{U})$ and set $X = Q_{\mathcal{U}}$, $Y = Q_{\mathcal{U}}^\top A$.

CLSI adds the extra clustering overhead of phase I, which, according to our experience with the method, is small. On the other hand, the necessary SVD's are applied on l matrix blocks of size $m \times n_j, j = 1, \dots, l$ each, whereas LSI would require the leading k singular vectors of the full, $m \times n$, matrix. This can lead to significant reductions in runtime, especially when l is close to k . We assume, of course, that the SVD is computed by means of some iterative algorithm; cf. [2]. The gain in runtime is expected to be larger for Algorithm 1 and the approximation better for Algorithm 2. Further time gains are possible in Algorithm 2 and CLSI, with block methods to compute consecutive singular triplets. We defer discussion of updating strategies for a full version of this paper.

Reference [21] addresses the following important issue. If we possess truncated SVD approximations for blocks of A , how does this information help us to approximate A ? This is directly related to our methodology, since it uses the leading left singular vectors of each block. Let each cluster block A_j be approximated using the optimal rank-1 matrix. Note now that the matrix $[\sigma_1^{(1)} u_1^{(1)} (v_1^{(1)})^\top, \dots, \sigma_1^{(k)} u_1^{(k)} (v_1^{(k)})^\top]$, whose columns are the best rank-1 approximations of each block A_j , can be rewritten as $\mathcal{B} := [\text{best}_1(A_1), \dots, \text{best}_1(A_k)] = \mathcal{U}\mathcal{S}\mathcal{V}^\top$, where $\text{best}_1(A_j)$ is the optimal rank 1 approximation of A_j and $\mathcal{U}, \mathcal{S}, \mathcal{V}$ are as discussed in formula 2.4 above. Consider next using \mathcal{B} or a lower rank approximation of it to approximate A . The error is $\|A - \mathcal{B}\|_F = \|A - \mathcal{U}\mathcal{S}\mathcal{V}^\top\|_F$ in the former case. How does this compare with the approximation offered by Algorithm 1? There we use $\min_{Y \in \mathbb{R}^{k \times m}} \|A - \mathcal{U}Y\|_F$, and thus search the space of $k \times m$ matrices to minimize the Frobenius norm; instead, with the former approach, \mathcal{B} is used directly and no minimization is performed. It immediately follows, therefore, that

$$(2.5) \quad \min_{Y \in \mathbb{R}^{k \times m}} \|A - \mathcal{U}Y\|_F \leq \|A - \mathcal{B}\|_F$$

The above argument can easily be extended to the case where each cluster block is optimally approximated with respect to the Frobenius norm by matrices of rank-1 or higher, except that this time $\mathcal{B} := [\text{best}_{k_1}(A_1), \dots, \text{best}_{k_l}(A_l)]$ so that $\mathcal{B} = \mathcal{U}\mathcal{S}\mathcal{V}^\top$, where \mathcal{U} is as in (2.3) and \mathcal{S}, \mathcal{V} modified accordingly (cf. formula 2.4). Proceeding as above we can show that for the above \mathcal{U} and \mathcal{B} , an inequality as (2.5) holds, except that $Y \in \mathbb{R}^{K \times m}$, where $K = \sum_{j=1}^l k_j$ and $K \geq k$. We will call an algorithm based on approximation \mathcal{B} and will denote the method based on this approach by BSVD and will use it in our experiments. We will assume that the clustering phase results in the same groups as Algorithm 2 and use as rank of the partial SVD approximation of each cluster A_i the value k_i , obtained in Algorithm 2.

A related approach to CLSI is CSVD ([4]) that, like ours, has strong connections with methods that attempt to find local structure in data space. CSVD uses clustering

and then obtains the principal components of each block $\hat{A}_i \hat{A}_i^\top$, where $\hat{A}_i = A_i - A_i e e^\top / n_i$ and e is the vector of all 1's. All mk eigenvalues of these matrices are sorted and components are selected for the reconstruction. For the size of problems we are considering here, a disadvantage of this well-designed method is cost. For related work see also [10], where clustering is first used and LSI is applied independently on each cluster though there is no discussion of low rank approximations of tdm. Of great interest are also the probabilistic techniques in [9]; we plan to study them in light of CLSI. Here we evaluate our methods vs. the powerful algorithms of [19] and further expanded upon together with MATLAB codes in [3]: these are SPQR (sparse pivoted QR approximation) and SCRA (sparse column-row approximation), that construct approximations of the form $A \cong XTY^\top$, where X, Y are sparse and T is small.

3 Numerical experiments

We next explore the performance of our approach using MATLAB 6.5 running on a 2.4 GHz Pentium IV PC with 512 MB of RAM. We examine runtimes, approximation error and IR precision. The SVD and QR decompositions are computed via MATLAB's functions `svds` and ("economy size") `qr`. Our data sets were built using TMG², a MATLAB tool for the construction of tdm's from text collections [20]. Here we only present experiments with tdm `cran lec`. This was produced from the CRANFIELD data set using the best term weighting and normalization combinations according to results in [20]. Runtimes are in seconds and include clustering, though we refer to the extended paper for further analysis. The methods considered are: Centroids ([16]); Algorithms 1 and 2 (Table 1); CLSI (Section 2); SPQR and SCRA (Section 2); BSVD (Section 2). The baseline method is the truncated SVD method (when it is clear from the context we simply call it SVD). We first evaluate the matrix approximation algorithms³. The preprocessing algorithm used was Spherical k-means (S-kmeans) [6]. We record the approximation error $\|A - \hat{A}\|_F$ and compare it with $\|A - U_k S_k V_k^\top\|_F$ from partial SVD using ranks $k = 10 : 10 : 150$. Algorithm 2 employed $l = 2, 3, k$. Because of the nondeterminism in the clustering, we ran 5 experiments and recorded the mean approximation error. Fig. 1 depicts errors (top) and runtimes (bottom) for truncated SVD, SPQR, SCRA, BSVD, Centroids and Algorithms 1 and 2. For Algorithm 2 we depict the errors obtained when $l = 2, 3, k$. Centroids appears to provide reasonable approximations at low cost compared to truncated SVD. Algorithms 1 and 2 appear to return better approximations with the latter being more

²See <http://scgroup.hpclab.ceid.upatras.gr/scgroup/Projects/TMG/>.

³Due to space limitations we only present results with `cran lec` and defer analysis of the effect of clustering.

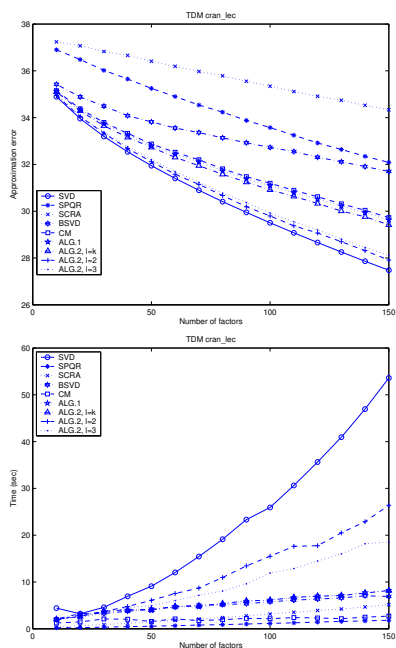


Figure 1: Approximation errors (top) and runtimes (bottom) for methods under review ($l = 2, 3, k$)

accurate. Algorithm 2 provides the best approximations than all other methods, with the approximations becoming better as l drops from k to 2. This is not surprising, as the $l = 1$ corresponds to the optimal (Eckard-Young) rank- k approximation. Fig. 1 (bottom) shows that runtimes increase as l decreases; in particular they are halved between $l = 2$ and $l = 1$, underlying the linearity of the expected runtime of (iterative) SVD algorithms with respect to the number of columns. The experiments indicate that Algorithms 1 and 2 can be economical effective alternatives to the classical rank- k approximation. Centroids also returns good overall performance; faster than Algorithms 1 and 2 with accuracy close to that returned by the former.

We next comment on SPQR, SCRA and BSVD. The compressed representation of A in Algorithms 1 and 2 requires \mathcal{U} (needing mk words) and Y (another nk words). SPQR and SCRA, on the other hand, need much less storage, namely $(n + 1)k$ and $(k + 2)k$ words, respectively, as well as some rows and columns of the tdm. From Fig. 1(top), for our datasets, Algorithms 1 and 2 return better approximations than SPQR, SCRA and BSVD. On the other hand, Centroids and SPQR are faster. Table 2 shows the runtimes that SPQR, SCRA, BSVD and Algorithms 1, 2, need to reach a specific error in the approximation achieved by the truncated SVD. Each table entry contains the number of factors required for such an approximation error as well as the precision in the querying process. Here we only present

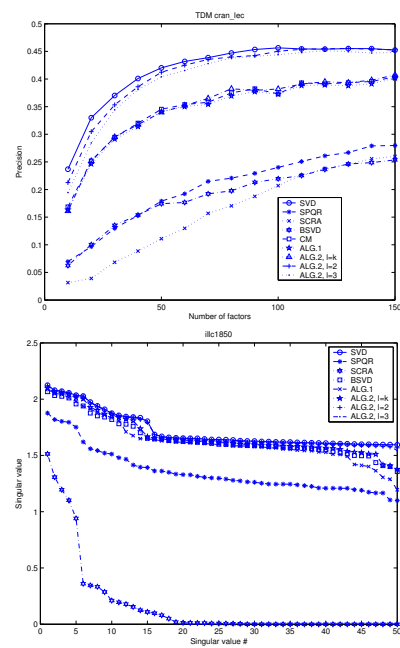


Figure 2: Precision (top); approximate singular values (bottom) for methods under review ($l = 2, 3, k$)

results for the `cran_lec` tdm, with $k = 70, 80$. It appears that SPQR is the fastest, BSVD and SCRA the slowest, while Algorithms 1 and 2 take time comparable with SPQR (for $l = k$). SPQR, SCRA and BSVD lead to larger values of k before the sought error threshold is reached. We also considered matrices unrelated to IR: Fig. 2(bottom) illustrates the 50 largest singular values of matrix `illc1850` (from www.nist.gov/MatrixMarket) approximated using SPQR, SCRA, BSVD and Algorithm 1 and 2. The latter two appear to return the best approximations.

| Alg.- k | 70 | 80 |
|----------------|-----------------|-----------------|
| SVD | 15.3, 70, 0.44 | 17.7, 80, 0.45 |
| SPQR | 2.7, 200, 0.30 | 3.3, 220, 0.31 |
| SCRA | 16.4, 310, 0.34 | 17.7, 330, 0.35 |
| BSVD | 8.9, 200, 0.28 | 9.4, 230, 0.27 |
| Alg.1 | 6.1, 110, 0.38 | 6.8, 130, 0.39 |
| Alg.2, $l = k$ | 6.1, 110, 0.39 | 6.4, 120, 0.40 |
| Alg.2, $l = 2$ | 11.4, 80, 0.44 | 12.3, 90, 0.44 |
| Alg.2, $l = 3$ | 8.0, 80, 0.44 | 9.8, 90, 0.44 |

Table 2: Results for SVD, SPQR, SCRA, BSVD, Algorithm 1 and 2 (runtime, k , precision) for the same approximation error in `cran_lec` tdm.

We next examine the effectiveness of the schemes for IR. Fig. 2(top) illustrates the $N = 11$ -point interpolated av-

erage precision for the algorithms. Centroids and Algorithm 1 appear to be competitive with LSI. Interestingly, for small values of l , Algorithm 2 can return better precision than LSI. Table 3 tabulates the maximum precision and corresponding k for each method and confirms that Centroids and Algorithms 1 and 2 can be quite effective for IR. Furthermore, for small l , Algorithm 2 may achieve higher precision than LSI.

| Alg.-Tdm | med_lfx | cran_lec | cisi_lfc |
|----------------|-----------------|------------------|-----------------|
| LSI | 0.70, 100 | 0.46, 100 | 0.25, 90 |
| SPQR | 0.56, 150 | 0.28, 150 | 0.17, 150 |
| SCRA | 0.51, 150 | 0.26, 150 | 0.18, 120 |
| BSVD | 0.46, 150 | 0.25, 150 | 0.14, 150 |
| CM | 0.65, 120 | 0.40, 150 | 0.20, 110 |
| Alg.1 | 0.65, 110 | 0.40, 150 | 0.20, 140 |
| Alg.2, $l = k$ | 0.65, 110 | 0.41, 150 | 0.20, 140 |
| Alg.2, $l = 2$ | 0.70, 70 | 0.46, 130 | 0.23, 110 |
| Alg.2, $l = 3$ | 0.69, 80 | 0.45, 120 | 0.23, 110 |

Table 3: Best precision and corresponding value of k pairs for LSI, SPQR, SCRA, BSVD, Centroids Method and Algorithms 1-2. Boldface indicates best precision/dataset.

Overall, it appears that our methodology can produce approximations with good performance. All proposed methods require smaller runtimes than partial SVD. Our experiments also indicate that CLSI is suitable for the querying process as in some cases it gives better results, faster than LSI. Therefore, CLSI could be used as basis of new algorithmic suites for IR from large and dynamic collections and merits additional study.

Acknowledgements We thank M. Berry, P. Drineas, E. Kokiopoulou and C. Bekas for helpful discussions and the reviewers for their comments regarding this work; due to lack of space, however, several issues and further experiments must be addressed in the full version of the paper; see also our group's website <http://scgroup.hpclab.ceid.upatras.gr>.

References

- [1] D. Achlioptas and F. McSherry, *Fast computation of low rank matrix approximations*, Proc. 33rd Annual ACM STOC, 2001, pp. 611–618.
- [2] M.W. Berry, *Large scale singular value decomposition*, Int. J. Supercomp. Appl. **6** (1992), 13–49.
- [3] M.W. Berry, S. A. Pulatova, and G. W. Stewart, *Computing sparse reduced-rank approximations to sparse matrices*, Tech. Rept. No. CS-04-525, Dept. Comput. Sci., Univ. Tennessee, Knoxville, May 2004.
- [4] V. Castelli, A. Thomasian, and C.-S. Li, *CSVD: Clustering and singular value decomposition for approximate similarity search in high-dimensional spaces*, IEEE Trans. Knowl. Data Eng. **15** (2003), no. 3, 671–685.
- [5] M. Chu and R. Funderlic, *The centroid decomposition: Relationships between discrete variational decompositions and SVDs*, SIAM J. Matrix Anal. Appl. **23** (2002), no. 4, 1025–1044.
- [6] I. S. Dhillon and D. S. Modha, *Concept decompositions for large sparse text data using clustering*, Machine Learning **42** (2001), no. 1, 143–175.
- [7] P. Drineas, R. Kannan, A. Frieze, S. Vempala, and V. Vinay, *Clustering of large graphs via the singular value decomposition*, Machine Learning **56** (2004), 9–33.
- [8] P. Drineas, R. Kannan, and M. Mahoney, *Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix*, Tech. Report TR-1270, Yale University, Department of Computer Science, February 2004.
- [9] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, Proc. 39th Annual Symp. FOCS, ACM, 1998, pp. 370–378.
- [10] J. Gao and J. Zhang, *Clustered SVD strategies in latent semantic indexing*, Tech. rept. no. 382-03, Dept. Comput. Science, Univ. Kentucky, Lexington, KY, 2003.
- [11] L. Guttman, *General Theory and Methods for Matrix Factorizing*, Psychometrica **9** (1944), 1–16.
- [12] L. Hubert, J. Meulman, and W. Heiser, *Two purposes for matrix factorization: A historical appraisal*, SIAM Rev. **42** (2000), no. 1, 68–82.
- [13] E. Kokiopoulou and Y. Saad, *Polynomial filtering in latent semantic indexing for information retrieval*, Proc. 27th ACM SIGIR (New York), ACM, 2004, pp. 104–111.
- [14] T. Kolda and D. O’Leary, *A semidiscrete matrix decomposition for latent semantic indexing information retrieval*, ACM Trans. Inform. Sys. **16** (1998), 322–346.
- [15] D. D. Lee and H. S. Seung, *Algorithms for non-negative matrix factorizations*, Advances in Neural Information Processing Systems **13** (2001), 556–562.
- [16] H. Park and L. Elden, *Matrix rank reduction for data analysis and feature extraction*, Tech. report, University of Minnesota CSE Tech. Rept., 2003.
- [17] H. Park, M. Jeon, and J.B. Rosen, *Lower dimensional representation of text data based on centroids and least squares*, BIT **43-2** (2003), 1–22.
- [18] G. Salton, C. Yang, and A. Wong, *A Vector-Space Model for Automatic Indexing*, CACM **18** (1975), no. 11, 613–620.
- [19] G. W. Stewart, *Four algorithms for the efficient computation of truncated pivoted QR approximations to a sparse matrix*, Numer. Math. **83** (1999), 313–323.
- [20] D. Zeimpekis and E. Gallopoulos, *TMG: A MATLAB toolbox for generating term-document matrices from text collections*, Technical Report, HPCLAB-SCG 1/6-04, Comput. Eng. & Inform. Dept., U. Patras, Greece, June 2004.
- [21] Z. Zhang and H. Zha, *Structure and perturbation analysis of truncated SVD for column-partitioned matrices*, SIAM J. Matrix Anal. Appl. **22** (2001), 1245–1262.
- [22] Z. Zhang, H. Zha, and H. Simon, *Low-rank approximations with sparse factors I: Basic algorithms and error analysis*, SIAM J. Matrix Anal. Appl. **23** (2002), no. 3, 706–727.