Statistics and Its Interface Volume 9 (2016) 453–459

A split-and-merge approach for singular value decomposition of large-scale matrices

Faming Liang*, Runmin Shi, and Qianxing Mo

We propose a new SVD algorithm based on the split-and-merge strategy, which possesses an embarrassingly parallel structure and thus can be efficiently implemented on a distributed or multicore machine. The new algorithm can also be implemented in serial for online eigen-analysis. The new algorithm is particularly suitable for big data problems: Its embarrassingly parallel structure renders it usable for feature screening, while this has been beyond the ability of the existing parallel SVD algorithms.

Keywords and phrases: Feature screening, Parallel computation, Online eigen-learning, Singular value decomposition.

1. INTRODUCTION

The singular value decomposition (SVD) is a key linear algebraic operation at the heart of many statistical and data mining methods. For example, principal component analysis (PCA), which is a special case of SVD, is a major tool of dimension reduction and has played an important role in high dimensional data analysis. The theoretical property of PCA for high dimensional problems has attracted much interest in the recent literature, see e.g., Johnstone (2001), Paul (2007), and Lee *et al.* (2010, 2014). In data mining, SVD has been widely used in clustering, latent semantic analysis, anomaly detection, collaborative filtering, computer recommendation and more. See Patterson et al. (2006), Deerwester et al. (1990), Idé and Kashima (2004), Eagle and Pentland (2009), and Sarwar et al. (2002) for some examples.

Despite its popularity, SVD is often restricted by its high computational complexity, which makes it impractical for massive datasets. Yet massive datasets are increasingly common in practice, many of which require real-time responsiveness. To accelerate the computation of SVD, some approximation methods have been proposed. For example, Sarwar et al. (2002) proposed an incremental SVD algorithm based on the projection technique, which is not exact as the resulting matrix decomposition is not orthogonal any more. Recently, some sample-based SVD approximation methods have also been studied, see e.g., Deshpande and Vempala (2006) and Holmes et al. (2008).

*Corresponding author.

In this paper, we propose a SVD algorithm based on the split-and-merge strategy. The new algorithm possesses an embarrassingly parallel structure and thus can be efficiently implemented on a distributed or multicore machine. The new algorithm can also be implemented in serial for online eigen-analysis. Compared to the standard SVD algorithm, the new algorithm can lead to significant savings in computational time in either the parallel or serial implementation. Compared to the existing parallel SVD algorithms, see e.g. Berry et al. (2005) for an overview, the new algorithm is easy to implement based on the existing SVD algorithm. Further, it can be accelerated with the existing parallel SVD algorithms.

The remainder of this paper is organized as follows. Section 2 describes the proposed algorithm. Section 3 discusses two applications of the proposed algorithm, feature screening and online eigen-learning. Section 4 presents some numerical results. Section 5 concludes the paper with a brief discussion.

2. A SPLIT-AND-MERGE SVD ALGORITHM

Given an $m \times n$ matrix X with rank r_X, with $m \geq n$, the singular value decomposition is defined as

$$\text{SVD}(X) = U \times D \times V^T, \tag{1}$$

where U and V are both orthogonal matrices with dimensions $m \times m$ and $n \times n$, respectively; and D is an $m \times n$ rectangular diagonal matrix with exactly r_X non-zero diagonal elements. The columns of U and V represent orthogonal eigenvectors of XX^T and X^TX, respectively. For convenience, we write $U = (u_1, \ldots, u_m)$ and $V = (v_1, \ldots, v_n)$, where u_i is an m-vector and v_i is a n-vector. The u_i is also called a left eigen-vector and v_i a right eigen-vector.

There are a lot of SVD algorithms and the most commonly used one is by Golub and Reinsch (1970). In the rest of this paper, we will use the Golub-Reinsch SVD algorithm as the standard algorithm to exhibit the split-and-merge approach. Note that the detailed algorithm for solving the SVD problem is not the topic here. We only show that the split-and-merge strategy can significantly improve the performance of the standard SVD algorithm.

One way to quantify the volume of work associated with a computation is to count flops. A flop is a floating point add, subtract, multiply, or divide. The number of flops of a

given algorithm is usually obtained by summing the amount of arithmetics associated with the most deeply nested statements, and we will denote this sum as the computational complexity. Referring to Golub and van Loan (2013), the Golub-Reinsch SVD algorithm of an $m \times n$ matrix has a computational complexity of $4m^2n + 8mn^2 + 9n^3$ to get the full components U, D and V. However, in some applications, only the the first n columns of U are required, denoting the matrix $\bar{U} = \left[U(,1:n), 0_{m\times(m-n)}\right]$, and the Golub-Reinsch SVD algorithm has only a computational complexity of $14mn^2 + 8n^3$ to get the reduced components $\bar{U}, D$ and V.

2.1 The algorithm

To describe the proposed algorithm, we first consider the scenario where m is much greater than n, but n is not very large. Note that we can always assume that m is greater than n; otherwise, for a low computational complexity, SVD can be done for X^T. Under this assumption, we can partition X by rows into a few submatrices:

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_s \end{pmatrix},$$

where s denotes the number of submatrices. Let $X_i = \bar{U}_i D_i V_i^T$ denote the SVD of X_i.

Define

$$\tilde{U} = \begin{pmatrix} \bar{U}_1 & & & \\ & \bar{U}_2 & & \\ & & \ddots & \\ & & & \bar{U}_s \end{pmatrix}, \qquad Y = \begin{pmatrix} D_1V_1^T \\ D_2V_2^T \\ \vdots \\ D_sV_s^T \end{pmatrix}.$$

Let $Y = U_y D_y V_y^T$ denote the SVD of Y. For convenience, we call Y the combined eigen-matrix. Then it is easy to verify that

$$X = \tilde{U}U_y D_y V_y^T$$

which forms a SVD of X. In summary, we have the following algorithm.

Algorithm 1. *(Split-and-Merge SVD: row partitioning only)*

(a) Partition X by rows into $X = [X_1^T, \ldots, X_s^T]^T$, where each X_i is of about the same size.
(b) Perform SVD for each X_i: $X_i = \bar{U}_i D_i V_i^T$.
(c) Perform SVD for the combined eigen-matrix $Y = [V_1D_1, \ldots, V_sD_s]^T$: $Y = U_y D_y V_y^T$,
(d) Output $\tilde{U}U_y$, D_y and V_y as the three components of SVD of X.

The algorithm requires that each X_i is of about the same size. This minimizes the waiting time between different nodes or cores of the computer and thus optimizes the performance of the algorithm.

2.2 Time complexity analysis

To analyze the time complexity of Algorithm 1, we assume that $n = o(m)$ and each X_i has the same rank r, i.e., $r_1 = r_2 = \cdots = r_s = r$. It is worth to mention that, when forming the combined eigen-matrix Y of the size $m \times n$, each $D_iV_i^T$ contains $\frac{m}{s} - r$ zero row-vectors in its bottom. That is, the matrix Y contains only sr non-zero row vectors. To see that the time complexity for solving the SVD on Y is equivalent to solving the same problem on the matrix which is only made by its sr non-zero row vectors, we let $\tilde{Y} = RY$ be a rearrangement in rows for Y such that its first sr row vectors are non-zero and $m - sr$ row zero vectors are in its bottom. Solve the SVD problem for its top non-zero part and we can get

$$Y = R\tilde{Y} = R\begin{bmatrix} UDV^T \\ 0 \end{bmatrix} = R\begin{bmatrix} U & 0 \\ 0 & I_{m-sr} \end{bmatrix}\begin{bmatrix} D \\ 0 \end{bmatrix} V^T,$$

where the rearrangement in row can be achieved by a delicate data structure and there is no extra work needed to get the left eigen-vector matrix $R\ diag(U, I_{m-sr})$.

2.2.1 Parallel implementation

The time complexity of Algorithm 1 is given by

(2)
$$\begin{aligned} T_{parallel} &= \left[14\frac{m}{s}n^2 + 8n^3\right] + [4s^2r^2n + 8srn^2 + 9n^3] \\ &\quad + [srn + m^2(2n-1)] \\ &= (2m^2 + 4s^2r^2 + sr)n + (14\frac{m}{s} + 8sr)n^2 + 17n^3 - m^2, \end{aligned}$$

where the first bracket $[\cdot]$ is for the time complexity of step (b), the second bracket $[\cdot]$ is for the time complexity of SVD(Y) performed in step (c), the term srn in the third bracket $[\cdot]$ is for the time complexity of forming the combined eigen-matrix $Y = [V_1D_1, \ldots, V_sD_s]$, and the term $m^2(2n-1)$ is for the time complexity of computing $\tilde{U}U_y$ by noting the sparse structure of $\tilde{U}$. Compared to $4m^2n + 8mn^2 + 9n^3$, the time complexity of the standard SVD algorithm, it is easy to see that as long as $\frac{7m}{4s} + 2sr + n < m$, i.e., $r < \frac{m-n}{2s} - \frac{7m}{8s^2} = O(m/s)$, the new algorithm will lead to some savings in computational time. Basically, it says that if D_i has a smaller number of non-zero rows than X_i, then the new algorithm will reduce the computational time of SVD. We note that this is always true if $m > sn$. This analysis suggests we choose $s < m/n$.

As implied by (2), if the partitioning in Algorithm 1 is done on columns, then the leading term of $T_{parallel}$ will be $4m^2sr$. Further, if $r \approx n/s$, then the proposed algorithm will not create much saving in computation. For this reason, we consider only the partitioning on rows.

2.2.2 Serial implementation

It is interesting to point out that even in serial implementations, Algorithm 1 can still benefit from the split-and-

merge strategy. In this case, the time complexity of the new algorithm is

$$\begin{aligned} T_{serial} &= \left[s(14\frac{m}{s}n^2+8n^3)\right]+[4s^2r^2n+8srn^2+9n^3] \\ &\quad +[srn+m^2(2n-1)] \\ &= (2m^2+4s^2r^2+sr)n+(14m+8sr)n^2 \\ &\quad +(8s+9)n^3-m^2, \end{aligned} \tag{3}$$

where the first bracket $[\cdot]$ accounts for the total time complexity of SVD(X_1), ..., SVD(X_s). Since $r \le \min\{m/s, n\}$ is always true, we have

$$\begin{aligned} T_{serial} &\le -m^2+2m^2n+(s+14m)n^2+(4s^2+16s+9)n^3 \\ &\le 2m^2n+14mn^2+(4s^2+17s+9)n^3. \end{aligned}$$

Hence, if the following inequality is true,

$$4s^2+17s+6\frac{m}{n}-2\left(\frac{m}{n}\right)^2<0, \tag{4}$$

then $T_{serial} < 4m^2n+8mn^2+9n^3$. For example, if $m/n=10$, then (4) can be satisfied for $2 \le s \le 4$; and if $m/n=100$, then (4) can be satisfied for $2 \le s \le 67$. Therefore, with an appropriate choice of s, the new algorithm provides a free improvement in computational time (without any requirements to the computer hardware) over the standard SVD algorithm.

2.3 SVD for general large-scale matrices

In the scenario that n is large and m is small, we can run Algorithm 1 on X^T. In the scenario that both n and m are large, we can partition X in both rows and columns into $s \times k$ submatrices; that is,

$$X=\begin{pmatrix} X_{11} & X_{12} & \cdots & X_{1k} \\ X_{21} & X_{22} & \cdots & X_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ X_{s1} & X_{s2} & \cdots & X_{sk} \end{pmatrix}, \tag{5}$$

where X_{ij}'s are of about the same size. Then we have the following algorithm for SVD of general large-scale matrices.

Algorithm 2. *(Split-and-Merge SVD)*

(a) Partition X in (5). Let $X_j=(X_{1j}^T,\ldots,X_{sj}^T)^T$ denote the jth column of the block matrix.
(b) Apply Algorithm 1 to each column X_j and get the SVD $X_j=U_jD_jV_j^T$ for $j=1,\ldots,k$. Let $\tilde{V}=diag(V_1,\cdots,V_k)$ denote a block diagonal matrix.
(c) Let $Z=(U_1D_1,\ldots,U_kD_k)$ be the combined eigenmatrix and get the SVD $Z=U_zD_zV_z^T$.
(d) Output U_z, D_z and $\tilde{V}V_z$ as the three components of SVD of X.

Let n_j denote the column number of X_j, and let r_j denote the rank of D_j. If r_j is much smaller than n_j for each j, then the new algorithm can lead to a substantial saving in computational time compared to the standard SVD algorithm, as the standard SVD algorithm is time sensitive to the column number. To see this more clearly, suppose that we can evenly partition X into $s \times k$ submatrices, and each column component X_j has the same rank r_X, i.e., $r_1=r_2=\cdots=r_s=r_X$ and each sub-component X_{ij} also has the same rank r, i.e., $r_{11}=r_{12}=\cdots=r_{sk}=r$.

Hence, following from (2), the time complexity for step (b) in the parallel case can be written as

$$\begin{aligned} T_{step(b)} &= k\Big[(2m^2+4s^2r^2+sr)\frac{n}{k}+(14\frac{m}{s}+8sr)\left(\frac{n}{k}\right)^2 \\ &\quad +17\left(\frac{n}{k}\right)^3-m^2\Big] \\ &= (2m^2+4s^2r^2+sr)n+(14\frac{m}{s}+8sr)\frac{n^2}{k} \\ &\quad +17\frac{n^3}{k^2}-km^2. \end{aligned}$$

Once again, the matrix Z contains only kr_X significant non-zero column vectors and the time complexity of solving the SVD problem is $4m^2kr_X+8mk^2{r_X}^2+9k^3{r_X}^3$. Consequently, we have

$$T_{step(c)}=mkr_X+4m^2kr_X+8mk^2{r_X}^2+9k^3{r_X}^3.$$

By adding the time complexity for calculating $\tilde{V}V_z$, one can finally get

$$T_{total}=T_{step(b)}+T_{step(c)}+n^2\left(2\frac{n}{k}-1\right).$$

If we only consider terms of order three, we have $T_{total}-T_{step(c)} \sim 2m^2n+\frac{14}{sk}mn^2+\frac{17+2k}{k^2}n^3$, provided that r is small. Compared with the time complexity of the standard SVD algorithm, $4m^2n+8mn^2+9n^3$, it is easy to see that a small r_X will produce a smaller $T_{step(c)}$ and, consequently, a smaller T_{total}. To be more precise, if $2kr_X<n$, the leading term of T_{total} will be smaller than $4m^2n$ and Algorithm 2 will potentially lead to some saving in computational time. In fact, as explained in Section 3.1, this is often the case in practical applications of SVD.

3. TWO APPLICATIONS OF THE PROPOSED ALGORITHM

In this section, we discuss two applications of the proposed algorithm, feature screening and online eigenlearning.

3.1 Feature screening

In practice, instead of the exact SVD, one often seeks for an approximation of X such that

$$\|X-\hat{X}\|_F^2 \le \epsilon\|X\|_F^2, \tag{6}$$

where $\hat{X}$ denotes the approximation, $\|A\|_F$ denotes the Frobenius norm of a matrix A, and $\epsilon>0$ is called the approximation error. It follows from (1) that for some $0<\tilde{r}<r$,

$$\hat{X} = \sum_{i=1}^{\tilde{r}} \sum_{i=1}^{\tilde{r}} d_i u_i v_i^T, \tag{7}$$

satisfies (6) if

$$\sum_{i=\tilde{r}+1}^{r} d_i^2 \le \epsilon \sum_{i=1}^{r} d_i^2. \tag{8}$$

That is, $\hat{X}$ can be obtained via (7) by choosing the first $\tilde{r}$ eigenvalues such that (8) is satisfied.

For the block matrix $X = (X_1, \ldots, X_k)$ and the approximator $\hat{X} = (\hat{X}_1, \ldots, \hat{X}_k)$, if $|X_i - \hat{X}_i\|_F^2 \le \epsilon \|X_i\|_F^2$ is satisfied for all $i = 1, \ldots, k$, then we have

$$\begin{aligned} \|(X_1, \ldots, X_k) - (\hat{X}_1, \ldots, \hat{X}_k)\|_F^2 &= \sum_{i=1}^{k} \|X_i - \hat{X}_i\|_F^2 \\ &\le \epsilon \sum_{i=1}^{k} \|X_i\|_F^2 = \epsilon \|X\|_F^2. \end{aligned} \tag{9}$$

If we call each pair of the eigenvectors, (u_i, v_i), a feature of X, then (9) provides a theoretical foundation for feature screening. The noise features, i.e., those corresponding to small eigenvalues, can be screened out from each subset data X_i. With feature screening, $\tilde{r}_i$, the number of selected features from X_i, can be much smaller than n_i, and thus the combined eigen-matrix Z in Algorithm 2 can have a much smaller column number than n. As a result, this reduces the time complexity of SVD. We note that for the spiked eigenvalue model (Johnstone, 2001), where a small number of population eigenvalues are substantially larger than the rest, the computational time of SVD can be substantially reduced by feature screening.

3.2 Online eigen-learning

One important application of the proposed algorithm is online eigen-analysis. Suppose that the data come in a stream, that is, the total number of observations m is fixed but the number of variables n increase. Let $X_{1:t}$ denote the data collected up to time t, and let X_{t+1} denote the data collected at time $t+1$. Let $X_{1:t} = U_{1:t} D_{1:t} V_{1:t}^T$ denote the SVD of $X_{1:t}$. Then the SVD of $X_{1:t+1}$ can be obtained recursively in the following algorithm.

Algorithm 3. *(Online Eigen-Learning)*

(a) Find the SVD $X_{t+1} = U_{t+1} D_{t+1} V_{t+1}^T$, and let $\tilde{V} = diag(V_{1:t}, V_{t+1})$ be a diagonal block matrix.
(b) Let $W = (U_{1:t} D_{1:t}, U_{t+1} D_{t+1})$ and find the SVD $W = U_w D_w V_w^T$.
(c) Output $U_{1:t+1} = U_w$, $D_{1:t+1} = D_w$ and $V_{1:t+1} = \tilde{V} V_w$ as the three components of SVD of $X_{1:t+1}$.

For this algorithm, the feature screening can also be applied in step (b) to reduce the column number of W. This algorithm can have many applications for stream type data, e.g., the spatio-temporal weather data studied in Onorati et al. (2013), the computer system data studied in Idé and Kashima (2004), and the mobile communication data studied in Akoglu and Faloutsos (2010). For these studies, the online eigen-learning algorithm can improve the computational efficiency of the involved eigen-analysis substantially.

4. NUMERICAL EXPERIMENTS

4.1 A simulated example

We generated a dataset from a population with three groups. The dataset consists of $m = 100$ samples and $n = 500{,}000$ variables. Among the 100 samples, 50 are from group 1, 30 are from group 2, and 20 are from group 3. The samples in each group were simulated by first creating its mean vector μ_g $(g = 1, 2, 3)$, for which each element was drawn randomly with replacement from the set $\{-0.3, 0, 0.3\}$, and then drawing independently from the multivariate Gaussian distribution MVN$(\mu_g, 4I_n)$, where I_n denotes an $n \times n$ identity matrix. A similar example has been used in Lee *et al.* (2010) for demonstrating the behavior of principal component (PC) score under the high dimensional setting.

For this dataset, since n is much greater than m, we actually worked on X^T for finding the decomposition. We implemented Algorithm 1 in both parallel and serial. The parallel means that step (b) is run in parallel, and the serial means step (b) is run in serial. The parallel version was implemented on a multicore computer (high-end Dell Precision T7610 workstation, 2.7 GHz processor) by calling the package *parallel* in *R*. The serial version was implemented on the same computer in *R*, but with only a single core being used. For comparison, we also applied the standard SVD in R to the dataset. In the standard SVD, only a single core was used. Table 1 shows the elapsed time used by the three algorithms. As expected, the proposed algorithm can be much faster than the standard SVD algorithm in its parallel implementation, and it is also faster than the standard SVD algorithm in its serial implementation due to the benefit of the split-and-merge strategy. It is remarkable that when $s = 20$, the serial implementation can be 20% faster than the standard SVD algorithm.

4.2 SVD for recommender systems

The recommender system is a crucial tool in E-commerce on the web, which applies data analysis techniques to help customers find which products they would like to purchase. The past research, see e.g., Berry et al. (1995), Gupta and Goldberg (1999), and Sarwar et al. (2000), suggested that the SVD-based approach can produce good predictive accuracy. Let X denote a product×customer matrix, where the number of customers is often very large. The SVD-based approach is to find a low rank matrix $\hat{X} = U_k D_k V_k^T$ that is the closest approximation to X, where k denotes the rank of the approximation matrix. Deerwester et al. (1990) and Berry et al. (1995) pointed out that the low rank approxi-

Table 1. Elapsed time used by the standard, parallel and serial algorithms for decomposition of the simulated dataset: s denotes the number of submatrices in the partition, and is also the number of cores used in the parallel implementation. The mean and standard deviation (SD) of the elapsed time (in seconds on a high-end Dell Precision T7610 workstation, 2.7 GHz processor) are calculated by averaging over five independent runs on the same dataset

Algorithm	Standard	parallel				Serial			
s	1	5	10	20	40	5	10	20	40
Mean(s)	10.58	2.94	2.14	2.05	1.42	10.01	8.92	8.02	8.89
SD	0.02	0.03	0.005	0.005	0.02	0.01	0.01	0.11	0.27

mation is better than the original data due to the filtering out of the small singular values that introduce "noise" in the product-customer relationship. As demonstrated in Sarwar et al. (2000), the SVD-based approach produced results that are better than a traditional collaborative filtering algorithm most of the time when applied to a Movie dataset.

However, as pointed out by Sarwar et al. (2002), the SVD-based approach suffer one serious limitation that makes them less suitable for large-scale data; the matrix decomposition step is computationally very expensive and is a major stumbling block towards achieving high scalability. To overcome this bottleneck, some approximate SVD algorithms, e.g., incremental SVD algorithm, have been used.

To demonstrate the advantage of the proposed algorithm, we applied the proposed algorithm to a MovieLens dataset with 10,677 movies and 71,567 customers. The dataset is downloaded at http://grouplens.org/datasets/movielens/. Figure 1 shows the eigenvalues of a sub-dataset with 894 customers. It indicates that the data contains only a very few significant features. Therefore, we applied the feature screening method described in Section 3.1 to this dataset. In our implementation of the feature screening method, the split-and-merge SVD algorithm was run in four steps as follows.

- Step 1: Partition the product×customers matrix by columns into 80 submatrices. Run SVD in parallel for each submatrix and approximate each submatrix using a low rank matrix with the approximation error $\epsilon_1 = 0.2$. This step leads to a combined eigen-matrix of size $10{,}677 \times 15{,}089$. On average, 188.6 features were selected for each submatrix.
- Step 2: Partition the eigen-matrix obtained in step 1 by columns into 40 submatrices. Run SVD in parallel for each submatrix and approximate each submatrix using a low rank matrix with the approximation error $\epsilon_2 = 0.2$. This step leads to a combined eigen-matrix of size $10{,}677 \times 5{,}470$.
- Step 3: Partition the eigen-matrix obtained in step 2 by columns into 20 submatrices. Run SVD in parallel for each submatrix and approximate each submatrix using a low rank matrix with the approximation error $\epsilon_3 = 0.2$. This step leads to a combined eigen-matrix of size $10{,}677 \times 1{,}595$.
- Step 4: Run SVD for the eigen-matrix obtained in step 3 and obtain a low rank approximator of X.

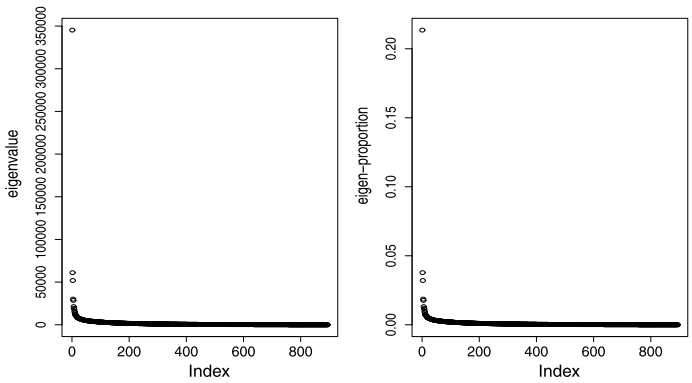

Figure 1. Left: eigenvalues of a sub-dataset with 894 customers. Right: eigen-proportions corresponds to the eigenvalues shown in the left plot, where the eigen-proportion is defined as $d_i^2 / \sum_{i=1}^{r} d_i^2$, and d_i is the ith singular value of the sub-dataset.

The algorithm cost 193.7 seconds (elapsed time) on the Dell Precision T7610 workstation. The resulting matrix approximator has an approximation error $\epsilon = 0.49$. As shown in Figure 1, this approximation accuracy is still acceptable. Note that if we run Algorithm 1 directly with the approximation error $\epsilon = 0.49$, it will lead to a combined eigen-matrix of size over 10,677×3,000, and the standard SVD algorithm for such a matrix is still quite time consuming. For comparison, we have also run the standard SVD algorithm for the original data on the same computer. It cost 4,779.8 seconds (elapsed time), about 24.7 times longer than the proposed algorithm.

5. DISCUSSION

In this paper, we have proposed a split-and-merge approach for singular value decomposition for large scale matrices. The proposed approach can be implemented in both distributed and serial machines. In either case, it can lead to significant savings in computational time. We have also discussed two applications of the proposed approach, feature screening and online eigen-analysis. We expect that the proposed approach can be applied for big data problems more often.

For feature screening, one important problem is signal detection, i.e., identifying the features (u_i, v_i)'s that represent

true signals. Signal detection is a long-standing problem in principal component analysis, and the most of existing approaches are based on eigen-proportions. Recently, for the case that X is a random matrix with each element normally distributed, Nadakuditi and Edelman (2008) proposed a hypothesis testing approach based on the results on the asymptotic distribution of eigenvalues. Liang (2007) proposed an approach based on a test for the pattern of eigenvectors. A further research in this direction is of interest.

In principal component analysis, it is often of interest to predict principal component scores for new observations from training samples. For a matrix X of variable×sample, the principal component scores are defined as the projection X^TU, where U is the u-eigenvector of X. Lee et al. (2010) demonstrated that naive approaches to principal component score prediction could be substantially biased toward 0 in the analysis of large matrices for which the number of variables was much larger than the number of samples. The online eigen-learning algorithm provides a natural way to address the bias problem: We suggest to replace the u-eigenvector of the training data by that of the full data (including training and testing) in calculating the predicted principal component scores. The online eigen-learning algorithm is very efficient for updating the u-eigenvector based on the test data. Moreover, it follows from Theorem 2 of Lee et al. (2014) that the principal component scores obtained in this way are consistent.

We note that parallel SVD algorithms have been developed in the literature, see e.g., Berry *et al.* (2005) for an overview. The major difference between the split-and-merge SVD algorithm and the existing parallel SVD algorithms is that the former possesses an embarrassingly parallel structure, which makes it particularly suitable for extremely large-scale matrices. As discussed in Section 3.1, the different submatrices X_i's can even be stored in different computers, and feature screening can be done for each submatrix separately. In addition, the split-and-merge SVD algorithm is implementable in serial, which makes it particularly suitable for an online analysis of stream data. The existing parallel SVD algorithms do not possess such attractive features.

Finally, we note that a split-and-merge strategy-based SVD algorithm has also been developed in the literature, see Tzeng (2013) for the detail of the algorithm. The major differences between Tzeng's algorithm and our algorithm are that the former works through a PCA procedure and its effectiveness depends on many factors, such as the estimated rank of the matrix and the manner of data splitting. When the estimated rank is smaller than the true rank of the matrix, Tzeng's algorithm can only produce an approximate SVD. If the data is not split appropriately, e.g., the subset data is not drawn randomly from the whole dataset or its size is too small, the resulting SVD is also approximate, see Tzeng et al. (2008) for an illustrative example. In addition, the efficiency of Tzeng's algorithm depends on the rank of the matrix. If the rank $r \approx \min(\sqrt{m}, \sqrt{n})$, then Tzeng's algorithm will have almost the same computational complexity as the original SVD algorithm. Compared to Tzeng's algorithm, our algorithm is accurate; it always produces an exact SVD regardless the rank of the matrix and the manner of data splitting. Also, the efficiency of our algorithm is less dependent on the true rank of the matrix. As implied by (2) and (3), our algorithm can be more efficient than the original SVD algorithm for a wide range of choices of data splitting. On the computational complexity side, Tzeng's algorithm can be better than ours only when $r \ll \min(m, n)$. In this case, it can gain some computational efficiency via the PCA procedure which works on a covariance matrix of size $O(r)$ for each subset data.

ACKNOWLEDGEMENTS

The authors thank the editor, associate editor and two referees for their comments which have led to significant improvement of this paper. Liang's research was partially supported by the National Science Foundation grants DMS-15052926 and DMS-15060903.

Received 20 November 2014

REFERENCES

Akoglu, L., and Faloutsos, C. (2010). Event detection in time series of mobile communication graphs. In *Proceedings of the Army Science Conference*, pp. 18–25.

Berry, M. W., Dumais, S. T., and O'Brain, G. W. (1995). Using linear algebra for intelligent information retrieval. *SIAM Review*, **37**, 573–595. MR1368388

Berry, M. W., Mezher, D., Philippe, B., and Sameh, A. (2005). Parallel algorithms for the singular value decomposition. In *Handbook of Parallel Computing and Statistics* (eds. E. J. Kontoghiorghes), Chapman & Hall/CRC Press, London, pp. 117–161. MR2282977

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

Deshpande, A., and Vempala, S. (2006). Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization and Combinatorial Optimization* (eds. J. Díaz, K. Jansen, J. D. P. Rolim, and U. Zwick), Lecture Notes in Comput. Sci. 4110, Springer, Berlin, pp. 292–303. MR2305018

Eagle, N., and Pentland, A. S. (2009). Eigenbehaviors: Identifying structure in routine. *Behavioral Ecology and Sociobiology*, **63**, 1057–1066.

Golub, G. H., and Reinsch, C. (1970). Singular value decomposition and least square solution. *Numer. Math.*, **14**, 403–420. MR1553974

Golub, G. H., and Van Loan C. F. (2013). *Matrix Computations* (Fourth Edition). The Johns Hopkins University Press, Baltimore. MR3024913

Gupta, D., and Goldberg, K. (1999). Jester 2.0: A Linear time collaborative filtering algorithm applied to jokes. In *Proc. of the ACM SIGIR*, pp. 291–292.

Holmes, M. P., Gray, A. G., and Isbell, C. L. (2008). Quic-SVD: Fast SVD using cosine trees. In *Advances in Neural Information Processing Systems 21*, pp. 673–680.

Idé, T., and Kashima, H. (2004). Eigenspace-based anomaly detection in computer systems. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, pp. 440–409.

Johnstone, I. M. (2001). On the distribution of the largest eigenvalue in principal component analysis. *Ann. Statist.*, **29**, 295–327. MR1863961

Lee, S., Zou, F., and Wright, F. (2010). Convergence an prediction of principal component scores in high-dimensional settings. *Ann. Statist.*, **38**, 3605–3629. MR2766862

Lee, S., Zou, F., and Wright, F. (2014). Convergence of sample eigenvalues, eigenvectors, and principal component scores for ultra-high dimensional data. *Biometrika*, **101**, 484–490. MR3215362

Liang, F. (2007). Use of SVD-based probit transformation in clustering gene expression profiles. *Computational Statistics & Data Analysis*, **51**, 6355–6366. MR2408599

Nadajuditi, R. R., and Edelman, A. (2008). Sample eigenvalue based detection of high-dimensional signals in white noise using relatively few samples. *IEEE Trans. on Signal Processing*, **56**, 2625–2638. MR1500236

Onorati, R., Sampson, P., and Guttorp, P. (2013). A spatio-temporal model based on the SVD to analyze daily average temperature across the sicily region. *Journal of Environmental Statistics*, **5**, 1–19.

Patterson, N., Price, A. L., and Reich, D. (2006). Population structure and eigenanalysis. *PLoS Genetics*, **2**, e190.

Paul, D. (2007). Asymptotics of sample eigenstructure for a large dimensional spiked covariance model. *Statist. Sinica*, **17**, 1617–1642. MR2399865

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2000). Application of dimensionality reduction in recommender system—a case study. In *ACM Web-Mining for E-Commerce Workshop*, ACM Press.

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*, ACM Press, pp. 27–28.

Tzeng, J. (2013). Split-and-combine singular value decomposition for large-scale matrix. *Journal of Applied Mathematics*, **2013**, article ID 683053. MR3033589

Tzeng, J., Lu, H. H.-S., and Li, W.-H. (2008). Multidimensional scaling for large genomic data sets. *BMC Bioinformatics*, **9**, 179.

Faming Liang
Department of Biostatistics
University of Florida
Gainesville, FL 32611
USA
E-mail address: faliang@ufl.edu

Runmin Shi
Department of Statistics
University of Florida
Gainesville, FL 32611
USA

Qianxing Mo
Department of Medicine and Dan L. Duncan Cancer Center
Baylor College of Medicine
Houston, TX 77030
USA