This article has been accepted for publication at the 2013 IEEE 13th International Conference on Data Mining

Distributed Column Subset Selection on MapReduce

Ahmed K. Farahat Ahmed Elgohary Ali Ghodsi Mohamed S. Kamel
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
Email: {afarahat, aelgohary, aghodsib, mkamel}@uwaterloo.ca

***Abstract*—Given a very large data set distributed over a cluster of several nodes, this paper addresses the problem of selecting a few data instances that best represent the entire data set. The solution to this problem is of a crucial importance in the big data era as it enables data analysts to understand the insights of the data and explore its hidden structure. The selected instances can also be used for data preprocessing tasks such as learning a low-dimensional embedding of the data points or computing a low-rank approximation of the corresponding matrix. The paper first formulates the problem as the selection of a few representative columns from a matrix whose columns are massively distributed, and it then proposes a MapReduce algorithm for selecting those representatives. The algorithm first learns a concise representation of all columns using random projection, and it then solves a generalized column subset selection problem at each machine in which a subset of columns are selected from the sub-matrix on that machine such that the reconstruction error of the concise representation is minimized. The paper then demonstrates the effectiveness and efficiency of the proposed algorithm through an empirical evaluation on benchmark data sets.**

***Keywords*-Column Subset Selection; Greedy Algorithms; Distributed Computing; Big Data; MapReduce;**

I. Introduction

Recent years have witnessed the rise of the big data era in computing and storage systems. With the great advances in information and communication technology, hundreds of petabytes of data are generated, transferred, processed and stored every day. The availability of this overwhelming amount of structured and unstructured data creates an acute need to develop fast and accurate algorithms to discover useful information that is hidden in the big data. One of the crucial problems in the big data era is the ability to represent the data and its underlying information in a succinct format.

Although different algorithms for clustering and dimension reduction can be used to summarize big data, these algorithms tend to learn representatives whose meanings are difficult to interpret. For instance, the traditional clustering algorithms such as k-means [1] tend to produce centroids which encode information about thousands of data instances. The meanings of these centroids are hard to interpret. Even clustering methods that use data instances as prototypes, such as k-medoid [2], learn only one representative for each cluster, which is usually not enough to capture the insights of the data instances in that cluster. In addition, using medoids as representatives implicitly assumes that the data points are distributed as clusters and that the number of those clusters are known ahead of time. This assumption is not true for many data sets. On the other hand, traditional dimension reduction algorithms such as Latent Semantic Analysis (LSA) [3] tend to learn a few latent concepts in the feature space. Each of these concepts is represented by a dense vector which combines thousands of features with positive and negative weights. This makes it difficult for the data analyst to understand the meaning of these concepts. Even if the goal of representative selection is to learn a low-dimensional embedding of data instances, learning dimensions whose meanings are easy to interpret allows the understanding of the results of the data mining algorithms, such as understanding the meanings of data clusters in the low-dimensional space.

The acute need to summarize big data to a format that appeals to data analysts motivates the development of different algorithms to directly select a few representative data instances and/or features. This problem can be generally formulated as the selection of a subset of columns from a data matrix, which is formally known as the Column Subset Selection (CSS) problem [4], [5], [6]. Although many algorithms have been proposed for tackling the CSS problem, most of these algorithms focus on randomly selecting a subset of columns with the goal of using these columns to obtain a low-rank approximation of the data matrix. In this case, these algorithms tend to select a relatively large number of columns. When the goal is to select a very few columns to be directly presented to a data analyst or indirectly used to interpret the results of other algorithms, the randomized CSS methods are not going to produce a meaningful subset of columns. On the other hand, deterministic algorithms for CSS, although more accurate, do not scale to work on big matrices with massively distributed columns.

This paper addresses the aforementioned problem by presenting a fast and accurate algorithm for selecting a very few columns from a big data matrix with massively distributed columns. The algorithm starts by learning a concise representation of the data matrix using random projection. Each machine then independently solves a generalized column subset selection problem in which a subset of columns is selected from the current sub-matrix such that the reconstruction error of the concise representation is minimized. A further selection step is then applied to

 DOI 10.1109/ICDM.2013.155

the columns selected at different machines to select the required number of columns. The proposed algorithm is designed to be executed efficiently over massive amounts of data stored on a cluster of several commodity nodes. In such settings of infrastructure, ensuring the scalability and the fault tolerance of data processing jobs is not a trivial task. In order to alleviate these problems, MapReduce [7] was introduced to simplify large-scale data analytics over a distributed environment of commodity machines. Currently, MapReduce (and its open source implementation Hadoop [8]) is considered the most successful and widely-used framework for managing big data processing jobs. The approach proposed in this paper considers the different aspects of developing MapReduce-efficient algorithms.

The contributions of the paper can be summarized as follows:

- The paper proposes an algorithm for distributed Column Subset Selection (CSS) which first learns a concise representation of the data matrix and then selects columns from distributed sub-matrices that approximate this concise representation.
- To facilitate CSS from different sub-matrices, a fast and accurate algorithm for generalized CSS is proposed. This algorithm greedily selects a subset of columns from a source matrix which approximates the columns of a target matrix.
- A MapReduce-efficient algorithm is proposed for learning a concise representation using random projection. The paper also presents a MapReduce algorithm for distributed CSS which only requires two passes over the data with a very low communication overhead.
- Large-scale experiments have been conducted on benchmark data sets in which different methods for CSS are compared.

The rest of the paper is organized as follows. Section II describes the notations used throughout the paper. Section III gives a brief background on the CSS problem. Section IV describes a centralized greedy algorithm for CSS, which is the core of the distributed algorithm presented in this paper. Section V gives a necessary background on the framework of MapReduce. The proposed MapReduce algorithm for distributed CSS is described in details in Section VI. Section VII reviews the state-of-the-art CSS methods and their applicability to distributed data. In Section VIII, an empirical evaluation of the proposed method is described. Finally, Section IX concludes the paper.

II. Notations

The following notations are used throughout the paper unless otherwise indicated. Scalars are denoted by small letters (e.g., m, n), sets are denoted in script letters (e.g., $\mathcal{S}$, $\mathcal{R}$), vectors are denoted by small bold italic letters (e.g., $\boldsymbol{f}$, $\boldsymbol{g}$), and matrices are denoted by capital letters (e.g., A, B). The subscript (i) indicates that the variable corresponds to the i-th block of data in the distributed environment. In addition, the following notations are used:

For a set $\mathcal{S}$:

$\lvert\mathcal{S}\rvert$	the cardinality of the set.

For a vector $\boldsymbol{x} \in \mathbb{R}^m$:

$\boldsymbol{x}_i$	i-th element of $\boldsymbol{x}$.
$\lVert\boldsymbol{x}\rVert$	the Euclidean norm (ℓ_2-norm) of $\boldsymbol{x}$.

For a matrix $A \in \mathbb{R}^{m \times n}$:

A_{ij}	(i,j)-th entry of A.
$A_{i:}$	i-th row of A.
$A_{:j}$	j-th column of A.
$A_{:\mathcal{S}}$	the sub-matrix of A which consists of the set $\mathcal{S}$ of columns.
A^T	the transpose of A.
$\lVert A\rVert_F$	the Frobenius norm of A: $\lVert A\rVert_F = \sqrt{\Sigma_{i,j} A_{ij}^2}$.
$\tilde{A}$	a low rank approximation of A.
$\tilde{A}_{\mathcal{S}}$	a rank-l approximation of A based on the set $\mathcal{S}$ of columns, where $\lvert\mathcal{S}\rvert = l$.

III. Column Subset Selection (CSS)

The Column Subset Selection (CSS) problem can be generally defined as the selection of the most representative columns of a data matrix [4], [5], [6]. The CSS problem generalizes the problem of selecting representative data instances as well as the unsupervised feature selection problem. Both are crucial tasks, that can be directly used for data analysis or as pre-processing steps for developing fast and accurate algorithms in data mining and machine learning.

Although different criteria for column subset selection can be defined, a common criterion that has been used in much recent work measures the discrepancy between the original matrix and the approximate matrix reconstructed from the subset of selected columns [9], [10], [11], [12], [13], [4], [5], [6], [14]. Most of the recent work either develops CSS algorithms that directly optimize this criterion or uses this criterion to assess the quality of the proposed CSS algorithms. In the present work, the CSS problem is formally defined as

Problem 1: **(Column Subset Selection)** Given an $m \times n$ matrix A and an integer l, find a subset of columns $\mathcal{L}$ such that $\lvert\mathcal{L}\rvert = l$ and

$$\mathcal{L} = \arg\min_{\mathcal{S}} \lVert A - P^{(\mathcal{S})} A\rVert_F^2,$$

where $P^{(\mathcal{S})}$ is an $m \times m$ projection matrix which projects the columns of A onto the span of the candidate columns $A_{:\mathcal{S}}$.

The criterion $\mathbf{F}(\mathcal{S}) = \lVert A - P^{(\mathcal{S})} A\rVert_F^2$ represents the sum of squared errors between the original data matrix A and its rank-l column-based approximation (where $l = \lvert\mathcal{S}\rvert$),

$$\tilde{A}_{\mathcal{S}} = P^{(\mathcal{S})} A\,. \tag{1}$$

 DOI 10.1109/ICDM.2013.155

In other words, the criterion $\mathbf{F}(\mathcal{S})$ calculates the Frobenius norm of the residual matrix $E = A - \tilde{A}_{\mathcal{S}}$. Other types of matrix norms can also be used to quantify the reconstruction error. Some of the recent work on the CSS problem [4], [5], [6] derives theoretical bounds for both the Frobenius and spectral norms of the residual matrix. The present work, however, focuses on developing algorithms that minimize the Frobenius norm of the residual matrix.

The projection matrix $P^{(\mathcal{S})}$ can be calculated as

$$P^{(\mathcal{S})} = A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}} \right)^{-1} A_{:\mathcal{S}}^T , \tag{2}$$

where $A_{:\mathcal{S}}$ is the sub-matrix of A which consists of the columns corresponding to $\mathcal{S}$. It should be noted that if $\mathcal{S}$ is known, the term $\left(A_{:\mathcal{S}}^T A_{:\mathcal{S}} \right)^{-1} A_{:\mathcal{S}}^T A$ is the closed-form solution of least-squares problem $T^* = \arg\min_T \|A - A_{:\mathcal{S}} T\|_F^2$.

The set of selected columns (i.e., data instances or features) can be directly presented to a data analyst to learn about the insights of the data, or they can be used to preprocess the data for further analysis. For instance, the selected columns can be used to obtain a low-dimensional representation of all columns into the subspace of selected ones. This representation can be obtained by calculating an orthogonal basis for the selected columns Q and then embedding all columns of A into the subspace of Q as $W = Q^T A$. The selected columns can also be used to calculate a column-based low-rank approximation of A [12]. Moreover, the leading singular values and vectors of the low-dimensional embedding W can be used to approximate those of the data matrix.

IV. Greedy CSS

The column subset selection criterion presented in Section III measures the reconstruction error of a data matrix based on the subset of selected columns. The minimization of this criterion is a combinatorial optimization problem whose optimal solution can be obtained in $O\left(n^l mnl\right)$ [5]. This section briefly describes a deterministic greedy algorithm for optimizing this criterion, which extends the greedy method for unsupervised feature selection recently proposed by Farahat et al. [15], [16]. A brief description of this method is included in this section for completeness. The reader is referred to [16] for the proofs of the different formulas presented in this section.

The greedy CSS [16] is based the following recursive formula for the CSS criterion.

Theorem 1: Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,

$$\mathbf{F}(\mathcal{S}) = \mathbf{F}(\mathcal{P}) - \|\tilde{E}_{\mathcal{R}}\|_F^2 ,$$

where $E = A - P^{(\mathcal{P})} A$, and $\tilde{E}_{\mathcal{R}}$ is the low-rank approximation of E based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns.

Proof: See [16, Theorem 2]. ■

The term $\|\tilde{E}_{\mathcal{R}}\|_F^2$ represents the decrease in reconstruction error achieved by adding the subset $\mathcal{R}$ of columns to $\mathcal{P}$. This recursive formula allows the development of an efficient greedy algorithm that approximates the optimal solution of the column subset selection problem. At iteration t, the goal is to find column p such that

$$p = \arg\min_i \quad \mathbf{F}(\mathcal{S} \cup \{i\}) , \tag{3}$$

where $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

Let G be an $n \times n$ matrix which represents the inner-products over the columns of the residual matrix E, i.e., $G = E^T E$. The greedy selection problem can be simplified to (See [16, Section 6])

Problem 2: **(Greedy Column Subset Selection)** At iteration t, find column p such that

$$p = \arg\max_i \quad \frac{\|G_{:i}\|^2}{G_{ii}}$$

where $G = E^T E$, $E = A - \tilde{A}_{\mathcal{S}}$ and $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

For iteration t, define $\boldsymbol{\delta} = G_{:p}$ and $\boldsymbol{\omega} = G_{:p} / \sqrt{G_{pp}} = \boldsymbol{\delta} / \sqrt{\boldsymbol{\delta}_p}$. The vector $\boldsymbol{\delta}^{(t)}$ can be calculated in terms of A and previous $\boldsymbol{\omega}$'s as

$$\boldsymbol{\delta}^{(t)} = A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)} . \tag{4}$$

The numerator and denominator of the selection criterion at each iteration can be calculated in an efficient manner without explicitly calculating E or G using the following theorem.

Theorem 2: Let $\boldsymbol{f}_i = \|G_{:i}\|^2$ and $\boldsymbol{g}_i = G_{ii}$ be the numerator and denominator of the criterion function for column i respectively, $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$, and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$. Then,

$$\begin{aligned}
\boldsymbol{f}^{(t)} &= \Big(\boldsymbol{f} - 2 \Big(\boldsymbol{\omega} \circ \Big(A^T A \boldsymbol{\omega} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\omega}^{(r)^T} \boldsymbol{\omega} \right) \boldsymbol{\omega}^{(r)} \Big) \Big) \\
&\quad + \|\boldsymbol{\omega}\|^2 (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \Big)^{(t-1)} , \\
\boldsymbol{g}^{(t)} &= \Big(\boldsymbol{g} - (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \Big)^{(t-1)} .
\end{aligned}$$

where $\circ$ represents the Hadamard product operator.

Proof: See [16, Theorem 4]. ■

Algorithm 1 shows the complete greedy CSS algorithm. The distributed CSS algorithm presented in this paper introduces a generalized variant of the greedy CSS algorithm in which a subset of columns is selected from a source matrix such that the reconstruction error of a target matrix is minimized. The distributed CSS method uses the greedy generalized CSS algorithm as the core method for selecting columns at different machines as well as in the final selection step.

 DOI 10.1109/ICDM.2013.155

Algorithm 1 Greedy Column Subset Selection

Input: Data matrix A, Number of columns l
Output: Selected subset of columns $\mathcal{S}$

1: Initialize $\mathcal{S} = \{\ \}$
2: Initialize $\boldsymbol{f}_i^{(0)} = \|A^T A_{:i}\|^2$, $\boldsymbol{g}_i^{(0)} = A_{:i}^T A_{:i}$ for $i = 1...n$
3: Repeat $t = 1 \rightarrow l$:
4: $\quad p = \arg\max_i \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{p\}$
5: $\quad \boldsymbol{\delta}^{(t)} = A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)}$
6: $\quad \boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$
7: $\quad$ Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s (Theorem 2)

V. MapReduce Paradigm

MapReduce [7] was presented as a programming model to simplify large-scale data analytics over a distributed environment of commodity machines. The rationale behind MapReduce is to impose a set of constraints on data access at each individual machine and communication between different machines to ensure both the scalability and fault-tolerance of the analytical tasks. Currently, MapReduce is considered the *de-facto* solution for many data analytics tasks over large distributed clusters [17], [18].

A MapReduce job is executed in two phases of user-defined data transformation functions, namely, *map* and *reduce* phases. The input data is split into physical blocks distributed among the nodes. Each block is viewed as a list of *key-value* pairs. In the first phase, the *key-value* pairs of each input block b are processed by a single *map* function running independently on the node where the block b is stored. The *key-value* pairs are provided one-by-one to the *map* function. The output of the *map* function is another set of intermediate *key-value* pairs. The values associated with the same *key* across all nodes are grouped together and provided as an input to the *reduce* function in the second phase. Different groups of values are processed in parallel on different machines. The output of each *reduce* function is a third set of *key-value* pairs and collectively considered the output of the job. It is important to note that the set of the intermediate *key-value* pairs is moved across the network between the nodes which incurs significant additional execution time when much data are to be moved. For complex analytical tasks, multiple jobs are typically chained together [17] and/or many rounds of the same job are executed on the input data set [18].

In addition to the programming model constraints, Karloff et al. [19] defined a set of computational constraints that ensure the scalability and the efficiency of MapReduce-based analytical tasks. These computational constraints limit the used memory size at each machine, the output size of both the *map* and *reduce* functions and the number of rounds used to complete a certain tasks.

The MapReduce algorithms presented in this paper adhere to both the programming model constraints and the computational constraints. The proposed algorithm aims also at minimizing the overall running time of the distributed column subset selection task to facilitate interactive data analytics.

VI. Distributed CSS on MapReduce

This section describes a MapReduce algorithm for the distributed column subset selection problem. Given a big data matrix A whose columns are distributed across different machines, the goal is to select a subset of columns $\mathcal{S}$ from A such that the CSS criterion $\mathbf{F}(\mathcal{S})$ is minimized.

One naïve approach to perform distributed column subset selection is to select different subsets of columns from the sub-matrices stored on different machines. The selected subsets are then sent to a centralized machine where an additional selection step is optionally performed to filter out irrelevant or redundant columns. Let $A_{(i)}$ be the sub-matrix stored at machine i, the naïve approach optimizes the following function.

$$\sum_{i=1}^{c} \left\| A_{(i)} - P^{(\mathcal{L}_{(i)})} A_{(i)} \right\|_F^2 , \tag{5}$$

where $\mathcal{L}_{(i)}$ is the set of columns selected from $A_{(i)}$ and c is the number of physical blocks of data. The resulting set of columns is the union of the sets selected from different sub-matrices: $\mathcal{L} = \cup_{i=1}^{c} \mathcal{L}_{(i)}$. The set $\mathcal{L}$ can further be reduced by invoking another selection process in which a smaller subset of columns is selected from $A_{:\mathcal{L}}$.

The naïve approach, however simple, is prone to missing relevant columns. This is because the selection at each machine is based on approximating a local sub-matrix, and accordingly there is no way to determine whether the selected columns are globally relevant or not. For instance, suppose the extreme case where all the truly representative columns happen to be loaded on a single machine. In this case, the algorithm will select a less-than-required number of columns from that machine and many irrelevant columns from other machines.

In order to alleviate this problem, the different machines have to select columns that best approximate a common representation of the data matrix. To achieve that, the proposed algorithm first learns a concise representation of the span of the big data matrix. This concise representation is relatively small and it can be sent over to all machines. After that each machine can select columns from its sub-matrix that approximate this concise representation. The proposed algorithm uses random projection to learn this concise representation, and proposes a generalized Column Subset Selection (CSS) method to select columns from different machines. The details of the proposed methods are explained in the rest of this section.

 DOI 10.1109/ICDM.2013.155

A. Random Projection

The first step of the proposed algorithm is to learn a concise representation B for a distributed data matrix A. In the proposed approach, a random projection method is employed. Random projection [20][21][22] is a well-known technique for dealing with the curse-of-the-dimensionality problem. Let Ω be a random projection matrix of size $n \times r$, and given a data matrix X of size $m \times n$, the random projection can be calculated as $Y = X\Omega$. It has been shown that applying random projection Ω to X preserves the pairwise distances between vectors in the row space of X with a high probability [20]:

$$\begin{aligned}(1-\epsilon)\left\|X_{i:}-X_{j:}\right\| &\leq \left\|X_{i:}\Omega - X_{j:}\Omega\right\| \\ &\leq (1+\epsilon)\left\|X_{i:}-X_{j:}\right\| ,\end{aligned} \tag{6}$$

where ϵ is an arbitrarily small factor.

Since the CSS criterion $\mathbf{F}(\mathcal{S})$ measures the reconstruction error between the big data matrix A and its low-rank approximation $P^{(\mathcal{S})}A$, it essentially measures the sum of the distances between the original rows and their approximations. This means that when applying random projection to both A and $P^{(\mathcal{S})}A$, the reconstruction error of the original data matrix A will be approximately equal to that of $A\Omega$ when both are approximated using the subset of selected columns:

$$\|A - P^{(\mathcal{S})}A\|_F^2 \approx \|A\Omega - P^{(\mathcal{S})}A\Omega\|_F^2 . \tag{7}$$

So, instead of optimizing $\|A - P^{(\mathcal{S})}A\|_F^2$, the distributed CSS can approximately optimize $\|A\Omega - P^{(\mathcal{S})}A\Omega\|_F^2$.

Let $B = A\Omega$, the distributed column subset selection problem can be formally defined as

Problem 3: **(Distributed Column Subset Selection)** Given an $m \times n_{(i)}$ sub-matrix $A_{(i)}$ which is stored at node i and an integer $l_{(i)}$, find a subset of columns $\mathcal{L}_{(i)}$ such that $|\mathcal{L}_{(i)}| = l_{(i)}$ and

$$\mathcal{L}_{(i)} = \arg\min_{\mathcal{S}} \|B - P^{(\mathcal{S})}B\|_F^2,$$

where $B = A\Omega$, Ω is an $n \times r$ random projection matrix, $\mathcal{S}$ is the set of the indices of the candidate columns and $\mathcal{L}_{(i)}$ is the set of the indices of the selected columns from $A_{(i)}$.

A key observation here is that random projection matrices whose entries are sampled $i.i.d$ from some univariate distribution Ψ can be exploited to compute random projection on MapReduce in a very efficient manner. Examples of such matrices are Gaussian random matrices [20], uniform random sign (± 1) matrices [21], and sparse random sign matrices [22].

In order to implement random projection on MapReduce, the data matrix A is distributed in a column-wise fashion and viewed as pairs of $\langle i, A_{:i}\rangle$ where $A_{:i}$ is the i-th column of A. Recall that $B = A\Omega$ can be rewritten as

$$B = \sum_{i=1}^{n} A_{:i}\Omega_{i:} \tag{8}$$

Algorithm 2 Fast Random Projection on MapReduce

Input: Data matrix A, Univariate distribution Ψ, Number of dimensions r

Output: Concise representation $B = A\Omega$, $\Omega_{ij} \sim \Psi \; \forall i, j$

```
map:
    B̄ = [0]_{m×r}
    foreach ⟨i, A_{:i}⟩
        Generate v = [v_1, v_2, ...v_r], v_j ~ Ψ
        B̄ = B̄ + A_{:i} v
    for j = 1 to m
        emit ⟨j, B̄_{j:}⟩
reduce:
    foreach ⟨j, [[B̄_(1)]_{j:}, [B̄_(2)]_{j:}, ..., [B̄_(c)]_{j:}]⟩
        B_{j:} = Σ_{i=1}^{c} [B̄_(i)]_{j:}
        emit ⟨j, B_{j:}⟩
```

and since the *map* function is provided one column of A at a time, one does not need to worry about pre-computing the full matrix Ω. In fact, for each input column $A_{:i}$, a new vector $\Omega_{i:}$ needs to be sampled from Ψ. So, each input column generates a matrix of size $m \times r$ which means that $O(nmr)$ data should be moved across the network to sum the generated n matrices at m independent $reducers$ each summing a row $B_{j:}$ to obtain B. To minimize that network cost, an *in-memory* summation can be carried out over the generated $m \times r$ matrices at each mapper. This can be done incrementally after processing each column of A. That optimization reduces the network cost to $O(cmr)$, where c is the number of physical blocks of the matrix[1]. Algorithm 2 outlines the proposed random projection algorithm. The term $emit$ is used to refer to outputting new $\langle key, value\rangle$ pairs from a mapper or a reducer.

B. Generalized CSS

This section presents the generalized column subset selection algorithm which will be used to perform the selection of columns at different machines. While Problem 1 is concerned with the selection of a subset of columns from a data matrix which best represent other columns of the same matrix, Problem 3 selects a subset of columns from a source matrix which best represent the columns of a different target matrix. The objective function of Problem 3 represents the reconstruction error of the target matrix B based on the selected columns from the source matrix. and the term $P^{(\mathcal{S})} = A_{:\mathcal{S}}\left(A_{:\mathcal{S}}^T A_{:\mathcal{S}}\right)^{-1} A_{:\mathcal{S}}^T$ is the projection matrix which projects the columns of B onto the subspace of the columns selected from A.

In order to optimize this new criterion, a greedy algorithm can be introduced. Let $\bar{\mathbf{F}}(\mathcal{S}) = \left\|B - P^{(\mathcal{S})}B\right\|_F^2$ be the

[1]The *in-memory* summation can also be replaced by a MapReduce *combiner* [7].

 DOI 10.1109/ICDM.2013.155

distributed CSS criterion, the following theorem derives a recursive formula for $\bar{\mathbf{F}}(\mathcal{S})$.

Theorem 3: Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,

$$\bar{\mathbf{F}}(\mathcal{S}) = \bar{\mathbf{F}}(\mathcal{P}) - \left\|\tilde{F}_{\mathcal{R}}\right\|_F^2 ,$$

where $F = B - P^{(\mathcal{P})}B$, and $\tilde{F}_{\mathcal{R}}$ is the low-rank approximation of F based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns of $E = A - P^{(\mathcal{P})}A$.

Proof: Using the recursive formula for the low-rank approximation of A: $\tilde{A}_{\mathcal{S}} = \tilde{A}_{\mathcal{P}} + \tilde{E}_{\mathcal{R}}$, and multiplying both sides with Ω gives

$$\tilde{A}_{\mathcal{S}}\Omega = \tilde{A}_{\mathcal{P}}\Omega + \tilde{E}_{\mathcal{R}}\Omega .$$

Low-rank approximations can be written in terms of projection matrices as

$$P^{(\mathcal{S})}A\Omega = P^{(\mathcal{P})}A\Omega + R^{(\mathcal{R})}E\Omega .$$

Using $B = A\Omega$,

$$P^{(\mathcal{S})}B = P^{(\mathcal{P})}B + R^{(\mathcal{R})}E\Omega .$$

Let $F = E\Omega$. The matrix F is the residual after approximating B using the set $\mathcal{P}$ of columns

$$F = E\Omega = \left(A - P^{(\mathcal{P})}A\right)\Omega = A\Omega - P^{(\mathcal{P})}A\Omega = B - P^{(\mathcal{P})}B.$$

This means that

$$P^{(\mathcal{S})}B = P^{(\mathcal{P})}B + R^{(\mathcal{R})}F$$

Substituting in $\bar{\mathbf{F}}(\mathcal{S}) = \left\|B - P^{(\mathcal{S})}B\right\|_F^2$ gives

$$\bar{\mathbf{F}}(\mathcal{S}) = \left\|B - P^{(\mathcal{P})}B - R^{(\mathcal{R})}F\right\|_F^2$$

Using $F = B - P^{(\mathcal{P})}B$ gives

$$\bar{\mathbf{F}}(\mathcal{S}) = \left\|F - R^{(\mathcal{R})}F\right\|_F^2$$

Using the relation between Frobenius norm and trace,

$$\begin{aligned}\bar{\mathbf{F}}(\mathcal{S}) &= \text{trace}\left(\left(F - R^{(\mathcal{R})}F\right)^T\left(F - R^{(\mathcal{R})}F\right)\right)\\ &= \text{trace}\left(F^TF - 2F^TR^{(\mathcal{R})}F + F^TR^{(\mathcal{R})}R^{(\mathcal{R})}F\right)\\ &= \text{trace}\left(F^TF - F^TR^{(\mathcal{R})}F\right) = \|F\|_F^2 - \left\|R^{(\mathcal{R})}F\right\|_F^2\end{aligned}$$

Using $\bar{\mathbf{F}}(\mathcal{P}) = \|F\|_F^2$ and $\tilde{F}_{\mathcal{R}} = R^{(\mathcal{R})}F$ proves the theorem. ■

Using the recursive formula for $\bar{\mathbf{F}}(\mathcal{S} \cup \{i\})$ allows the development of a greedy algorithm which at iteration t optimizes

$$p = \arg\min_i \bar{\mathbf{F}}(\mathcal{S} \cup \{i\}) = \arg\max_i \left\|\tilde{F}_{\{i\}}\right\|_F^2 \qquad (9)$$

Algorithm 3 Greedy Generalized Column Subset Selection

Input: Source matrix A, Target matrix B, Number of columns l

Output: Selected subset of columns $\mathcal{S}$

1: Initialize $\boldsymbol{f}_i^{(0)} = \|B^TA_{:i}\|^2$, $\boldsymbol{g}_i^{(0)} = A_{:i}^TA_{:i}$ for $i = 1...n$
2: Repeat $t = 1 \to l$:
3: $\quad p = \arg\max_i \boldsymbol{f}_i^{(t)}/\boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{p\}$
4: $\quad \boldsymbol{\delta}^{(t)} = A^TA_{:p} - \sum_{r=1}^{t-1}\boldsymbol{\omega}_p^{(r)}\boldsymbol{\omega}^{(r)}$
5: $\quad \boldsymbol{\gamma}^{(t)} = B^TA_{:p} - \sum_{r=1}^{t-1}\boldsymbol{\omega}_p^{(r)}\boldsymbol{\upsilon}^{(r)}$
6: $\quad \boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)}/\sqrt{\boldsymbol{\delta}_p^{(t)}}$, $\boldsymbol{\upsilon}^{(t)} = \boldsymbol{\gamma}^{(t)}/\sqrt{\boldsymbol{\delta}_p^{(t)}}$
7: $\quad$ Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s (Theorem 4)

Let $G = E^TE$ and $H = F^TE$, the objective function of this optimization problem can be simplified as follows.

$$\begin{aligned}\left\|\tilde{F}_{\{i\}}\right\|_F^2 &= \left\|E_{:i}\left(E_{:i}^TE_{:i}\right)^{-1}E_{:i}^TF\right\|_F^2\\ &= \text{trace}\left(F^TE_{:i}\left(E_{:i}^TE_{:i}\right)^{-1}E_{:i}^TF\right)\\ &= \frac{\left\|F^TE_{:i}\right\|^2}{E_{:i}^TE_{:i}} = \frac{\|H_{:i}\|^2}{G_{ii}} .\end{aligned} \qquad (10)$$

This allows the definition of the following generalized CSS problem.

Problem 4: **(Greedy Generalized CSS)** At iteration t, find column p such that

$$p = \arg\max_i \frac{\|H_{:i}\|^2}{G_{ii}}$$

where $H = F^TE$, $G = E^TE$, $F = B - P^{(\mathcal{S})}B$, $E = A - P^{(\mathcal{S})}A$ and $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

For iteration t, define $\boldsymbol{\gamma} = H_{:p}$ and $\boldsymbol{\upsilon} = H_{:p}/\sqrt{G_{pp}} = \boldsymbol{\gamma}/\sqrt{\boldsymbol{\delta}_p}$. The vector $\boldsymbol{\gamma}^{(t)}$ can be calculated in terms of A, B and previous $\boldsymbol{\omega}$'s and $\boldsymbol{\upsilon}$'s as $\boldsymbol{\gamma}^{(t)} = B^TA_{:p} - \sum_{r=1}^{t-1}\boldsymbol{\omega}_p^{(r)}\boldsymbol{\upsilon}^{(r)}$.

Similarly, the numerator and denominator of the selection criterion at each iteration can be calculated in an efficient manner using the following theorem.

Theorem 4: Let $\boldsymbol{f}_i = \|H_{:i}\|^2$ and $\boldsymbol{g}_i = G_{ii}$ be the numerator and denominator of the greedy criterion function for column i respectively, $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$, and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$. Then,

$$\begin{aligned}\boldsymbol{f}^{(t)} &= \Big(\boldsymbol{f} - 2\left(\boldsymbol{\omega} \circ \left(A^TB\boldsymbol{\upsilon} - \Sigma_{r=1}^{t-2}\left(\boldsymbol{\upsilon}^{(r)T}\boldsymbol{\upsilon}\right)\boldsymbol{\omega}^{(r)}\right)\right)\\ &\quad + \|\boldsymbol{\upsilon}\|^2(\boldsymbol{\omega} \circ \boldsymbol{\omega})\Big)^{(t-1)},\\ \boldsymbol{g}^{(t)} &= \left(\boldsymbol{g} - (\boldsymbol{\omega} \circ \boldsymbol{\omega})\right)^{(t-1)},\end{aligned}$$

where $\circ$ represents the Hadamard product operator.

As outlined in Section VI-A, the algorithm's distribution strategy is based on sharing the concise representation of the data B among all mappers. Then, independent $l_{(b)}$ columns

 DOI 10.1109/ICDM.2013.155

Algorithm 4 Distributed CSS on MapReduce

Input: Matrix A of size $m \times n$, Concise representation B, Number of columns l
Output: Selected columns C

map:
  $A_{(b)} = [\ ]$
  foreach $\langle i, A_{:i} \rangle$
    $A_{(b)} = [A_{(b)}\ A_{:i}]$
  $\bar{\mathcal{S}} =$ GeneralizedCSS($A_{(b)}, B, l_{(b)}$)
  foreach j **in** $\bar{\mathcal{S}}$
    emit $\langle 0, [A_{(b)}]_{:j} \rangle$
reduce:
  For all values $\{[A_{(1)}]_{:\bar{\mathcal{S}}_{(1)}}, [A_{(2)}]_{:\bar{\mathcal{S}}_{(2)}}, ..., [A_{(c)}]_{:\bar{\mathcal{S}}_{(c)}}\}$
  $A_{(0)} = \left[[A_{(1)}]_{:\bar{\mathcal{S}}_{(1)}}, [A_{(2)}]_{:\bar{\mathcal{S}}_{(2)}}, ..., [A_{(c)}]_{:\bar{\mathcal{S}}_{(c)}}\right]$
  $\mathcal{S} =$ GeneralizedCSS ($A_{(0)}$, B, l)
  foreach j **in** $\mathcal{S}$
    emit $\langle 0, [A_{(0)}]_{:j} \rangle$

from each mapper are selected using the generalized CSS algorithm. A second phase of selection is run over the $\sum_{b=1}^{c} l_{(b)}$ (where c is the number of input blocks) columns to find the best l columns to represent B. Different ways can be used to set $l_{(b)}$ for each input block b. In the context of this paper, the set of $l_{(b)}$ is assigned uniform values for all blocks (i.e. $l_{(b)} = \lfloor l/c \rfloor \forall b \in 1, 2, ..c$). Other methods are to be considered in future extensions. Algorithm 4 sketches the MapReduce implementation of the distributed CSS algorithm. It should be emphasized that the proposed MapReduce algorithm requires only two passes over the data set and its moves a very few amount of the data across the network.

VII. Related Work

Different approaches have been proposed for selecting a subset of representative columns from a data matrix. This section focuses on briefly describing these approaches and their applicability to massively distributed data matrices. The Column Subset Selection (CSS) methods can be generally categorized into randomized, deterministic and hybrid.

The randomized methods sample a subset of columns from the original matrix using carefully chosen sampling probabilities. Frieze et al. [9] was the first to suggest the idea of randomly sampling l columns from a matrix and using these columns to calculate a rank-k approximation of the matrix (where $l \geq k$). That work of Frieze et al. was followed by different papers [10], [11] that enhanced the algorithm by proposing different sampling probabilities. Drineas et al. [12] proposed a subspace sampling method which samples columns using probabilities proportional to the norms of the rows of the top k right singular vectors of A. Deshpande et al. [13] proposed an adaptive sampling method which updates the sampling probabilities based on the columns selected so far.

Column subset selection with uniform sampling can be easily implemented on MapReduce. For non-uniform sampling, the efficiency of implementing the selection on MapReduce is determined by how easy are the calculations of the sampling probabilities. The calculations of probabilities that depend on calculating the leading singular values and vectors are time-consuming on MapReduce. On the other hand, adaptive sampling methods are computationally very complex as they depend on calculating the residual of the whole data matrix after each iteration.

The second category of methods employs a deterministic algorithm for selecting columns such that some criterion function is minimized. This criterion function usually quantifies the reconstruction error of the data matrix based on the subset of selected columns. The deterministic methods are slower, but more accurate, than the randomized ones. In the area of numerical linear algebra, the column pivoting method exploited by the QR decomposition [23] permutes the columns of the matrix based on their norms to enhance the numerical stability of the QR decomposition algorithm. The first l columns of the permuted matrix can be directly selected as representative columns. Besides methods based on QR decomposition, different recent methods have been proposed for directly selecting a subset of columns from the data matrix. Boutsidis et al. [4] proposed a deterministic column subset selection method which first groups columns into clusters and then selects a subset of columns from each cluster. Çivril and Magdon-Ismail [14] presented a deterministic algorithm which greedily selects columns from the data matrix that best represent the right leading singular values of the matrix. Recently, Boutsidis et al. [6] presented a column subset selection algorithm which first calculates the top-k right singular values of the data matrix (where k is the target rank) and then uses deterministic sparsification methods to select $l \geq k$ columns from the data matrix. Besides, other deterministic algorithms have been proposed for selecting columns based on the volume defined by them and the origin [24], [25].

The deterministic algorithms are more complex to implement on MapReduce. For instance, it is time-consuming to calculate the leading singular values and vectors of a massively distributed matrix or to cluster their columns using k-means. It is also computationally complex to calculate QR decomposition with pivoting. Moreover, the recently proposed algorithms for volume sampling are more complex than other CSS algorithms as well as the one presented in this paper, and they are infeasible for large data sets.

A third category of CSS techniques is the hybrid methods which combine the benefits of both the randomized and deterministic methods. In these methods, a large subset of columns is randomly sampled from the columns of the data matrix and then a deterministic step is employed to reduce

 DOI 10.1109/ICDM.2013.155

Table I
THE PROPERTIES OF THE DATA SETS USED TO EVALUATE THE DISTRIBUTED CSS METHOD.

Data set	Type	# Instances	# Features
RCV1-200K	Documents	193,844	47,236
TinyImages-1M	Images	1 million	1,024

the number of selected columns to the desired rank. For instance, Boutsidis et al. [5] proposed a two-stage hybrid CSS algorithm which first samples $O(l \log l)$ columns based on probabilities calculated using the l-leading right singular vectors, and then employs a deterministic algorithm to select exactly l columns from the columns sampled in the first stage. However, the algorithm depends on calculating the leading l right singular vectors which is time-consuming for large data sets.

The hybrid algorithms for CSS can be easily implemented on MapReduce if the randomized selection step is MapReduce-efficient and the deterministic selection step can be implemented on a single machine. This is usually true if the number of columns selected by the randomized step is relatively small.

In comparison to other CSS methods, the algorithm proposed in this paper is designed to be MapReduce-efficient. In the distributed selection step, representative columns are selected based on a common representation. The common representation proposed in this work is based on random projection. This is more efficient than the work of Çivril and Magdon-Ismail [14] which selects columns based on the leading singular vectors. In comparison to other deterministic methods, the proposed algorithm is specifically designed to be parallelized which makes it applicable to big data matrices whose columns are massively distributed. On the other hand, the two-step of distributed then centralized selection is similar to that of the hybrid CSS methods. The proposed algorithm however employs a deterministic algorithm at the distributed selection phase which is more accurate than the randomized selection employed by hybrid methods in the first phase.

VIII. EXPERIMENTS

Experiments have been conducted on two big data sets to evaluate the efficiency and effectiveness of the proposed distributed CSS algorithm on MapReduce. The properties of the data sets are described in Table I. The *RCV1-200K* is a subset of the RCV1 data set [26] which has been prepared and used by Chen et al. [27] to evaluate parallel spectral clustering algorithms. The *TinyImages-1M* data set contains 1 million images that were sampled from the 80 million tiny images data set [28] and converted to grayscale.

Similar to previous work on CSS, the different methods are evaluated according to their ability to minimize the reconstruction error of the data matrix based on the subset of selected columns. In order to quantify the reconstruction error across different data sets, a relative accuracy measure is defined as

$$\text{Relative Accuracy} = \frac{\|A - \tilde{A}_{\mathcal{U}}\|_F - \|A - \tilde{A}_{\mathcal{S}}\|_F}{\|A - \tilde{A}_{\mathcal{U}}\|_F - \|A - \tilde{A}_l\|_F} \times 100\% \,,$$

where $\tilde{A}_{\mathcal{U}}$ is the rank-l approximation of the data matrix based on a random subset $\mathcal{U}$ of columns, $\tilde{A}_{\mathcal{S}}$ is the rank-l approximation of the data matrix based on the subset $\mathcal{S}$ of columns and $\tilde{A}_l$ is the best rank-l approximation of the data matrix calculated using the Singular Value Decomposition (SVD). This measure compares different methods relative to the uniform sampling as a baseline with higher values indicating better performance.

The experiments were conducted on Amazon EC2[2] clusters, which consist of 10 instances for the *RCV1-200K* data set and 20 instances for the *TinyImages-1M* data set. Each instance has a 7.5 GB of memory and a two-cores processor. All instances are running Debian 6.0.5 and Hadoop version 1.0.3. The data sets were converted into a binary format in the form of a sequence of *key-value* pairs. Each pair consisted of a column index as the key and a vector of the column entries. That is the standard format used in Mahout[3] for storing distributed matrices.

The distributed CSS method has been compared with different state-of-the-art methods. It should be noted that most of these methods were not designed with the goal of applying them to massively-distributed data, and hence their implementation on MapReduce is not straightforward. However, the designed experiments used the best practices for implementing the different steps of these methods on MapReduce to the best of the authors' knowledge. In specific, the following distributed CSS algorithms were compared.

- **UniNoRep**: is uniform sampling of columns without replacement. This is usually the worst performing method in terms on approximation error and it will be used as a baseline to evaluate methods across different data sets.
- **HybirdUni**, **HybirdCol** and **HybirdSVD**: are different distributed variants of the hybrid CSS algorithm which can be implemented efficiently on MapReduce. In the randomized phase, the three methods use probabilities calculated based on uniform sampling, column norms and the norms of the leading singular vectors' rows, respectively. The number of selected columns in the randomized phase is set to $l \log(l)$. In the deterministic phase, the centralized greedy CSS is employed to select exactly l columns from the randomly sampled columns.
- **DistApproxSVD**: is an extension of the centralized algorithm for sparse approximation of Singular Value Decomposition (SVD) [14]. The distributed CSS algorithm presented in this paper (Algorithm 4) is used

[2]Amazon Elastic Compute Cloud (EC2): http://aws.amazon.com/ec2

[3]Mahout is an Apache project for implementing Machine Learning algorithms on Hadoop. See http://mahout.apache.org/.

 DOI 10.1109/ICDM.2013.155

Table II
THE RUN TIMES AND RELATIVE ACCURACIES OF DIFFERENT CSS METHODS. THE BEST PERFORMING METHOD FOR EACH l IS HIGHLIGHTED IN BOLD, AND THE SECOND BEST METHOD IS UNDERLINED. NEGATIVE MEASURES INDICATE METHODS THAT PERFORM WORSE THAN UNIFORM SAMPLING.

Methods	Run time (minutes)			Relative accuracy (%)		
	$l = 10$	$l = 100$	$l = 500$	$l = 10$	$l = 100$	$l = 500$
RCV1 - 200K						
Uniform - Baseline	0.6	0.6	0.5	0.00	0.00	0.00
Hybird (Uniform)	0.8	0.8	2.9	-2.37	-1.28	4.49
Hybird (Column Norms)	1.6	1.5	3.7	4.54	0.81	6.60
Hybird (SVD-based)	1.3	1.4	3.6	9.00	12.10	18.43
Distributed Approx. SVD	16.6	16.7	18.8	<u>41.50</u>	57.19	63.10
Distributed Greedy CSS (rnd)	5.8	6.2	7.9	**51.76**	<u>61.92</u>	<u>67.75</u>
Distributed Greedy CSS (ssgn)	2.2	2.9	5.1	40.30	**62.41**	**67.91**
Tiny Images - 1M						
Uniform - Baseline	1.3	1.3	1.3	0.00	0.00	0.00
Hybird (Uniform)	1.5	1.7	8.3	19.99	6.85	6.50
Hybird (Column Norms)	3.3	3.4	9.4	17.28	3.57	7.80
Hybird (SVD-based)	52.4	52.5	59.4	3.59	8.57	10.82
Distributed Approx. SVD	71.0	70.8	75.2	**70.02**	**31.05**	**24.49**
Distributed Greedy CSS (ssgn)	22.1	23.6	24.2	<u>67.58</u>	<u>25.18</u>	<u>20.74</u>

to select columns that best approximate the leading singular vectors (by setting $B = U_k\Sigma_k$). The use of the distributed CSS algorithm extends the original algorithm proposed by Çivril and Magdon-Ismail [14] to work on distributed matrices. In order to allow efficient implementation on MapReduce, the number of leading singular vectors is set of 100.

- **DistGreedyCSS**: is the distributed column subset selection method described in Algorithm 4. For all experiments, the dimension of the random projection matrix is set to 100. This makes the size of the concise representation the same as the DistApproxSVD method. Two types of random matrices are used for random projection: (1) a dense Gaussian random matrix (rnd), and (2) a sparse random sign matrix (ssgn).

For the methods that require the calculations of Singular Value Decomposition (SVD), the Stochastic SVD (SSVD) algorithm [29] is used to approximate the leading singular values and vectors of the data matrix. The use of SSVD significantly reduces the run time of the original SVD-based algorithms while achieving comparable accuracy. In the conducted experiments, the SSVD implementation of Mahout was used.

Table II shows the run times and relative accuracies for different CSS methods. It can be observed from the table that for the *RCV1-200K* data set, the DistGreedyCSS methods (with random Gaussian and sparse random sing matrices) outperforms all other methods in terms of relative accuracies. In addition, the run times of both of them are relatively small compared to the DistApproxSVD method which achieves accuracies that are close to the DistGreedyCSS method. Both the DistApproxSVD and DistGreedyCSS methods achieve very good approximation accuracies compared to randomized and hybrid methods. It should also be noted that using a sparse random sign matrix for random projection takes much less time than a dense Gaussian matrix, while achieving comparable approximation accuracies. Based on this observation, the sparse random matrix has been used with the *TinyImages-1M* data set.

For the *TinyImages-1M* data set, although the DistApproxSVD achieves slightly higher approximation accuracies than DistGreedyCSS (with sparse random sign matrix), the DistGreedyCSS selects columns in almost one-third of the time. The reason why the DistApproxSVD outperforms DistGreedyCSS for this data set is that its rank is relatively small (less than 1024). This means that using the leading 100 singular values to represent the concise representation of the data matrix captures most of the information in the matrix and accordingly is more accurate than random projection. The DistGreedyCSS however still selects a very good subset of columns in a relatively small time.

IX. CONCLUSION

This paper proposes an accurate and efficient MapReduce algorithm for selecting a subset of columns from a massively distributed matrix. The algorithm starts by learning a concise representation of the data matrix using random projection. It then selects columns from each sub-matrix that best approximate this concise approximation. A centralized selection step is then performed on the columns selected from different sub-matrices. In order to facilitate the implementation of the proposed method, a novel algorithm for greedy generalized CSS is proposed to perform the selection from different sub-matrices. In addition, the different steps of the algorithms are carefully designed to be MapReduce-efficient. Experiments on big data sets demonstrate the effectiveness and efficiency of the proposed algorithm in comparison to other CSS methods when implemented on distributed data.

REFERENCES

[1] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[2] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," Technische Hogeschool, Delft (Netherlands). Department of Mathematics and Informatics, Tech. Rep., 1987.

[3] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science and Technology*, vol. 41, no. 6, pp. 391–407, 1990.

[4] C. Boutsidis, J. Sun, and N. Anerousis, "Clustered subset selection and its applications on it service metrics," in *Proceedings of the Seventeenth ACM Conference on Information and Knowledge Management (CIKM'08)*, 2008, pp. 599–608.

[5] C. Boutsidis, M. W. Mahoney, and P. Drineas, "An improved approximation algorithm for the column subset selection problem," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, 2009, pp. 968–977.

[6] C. Boutsidis, P. Drineas, and M. Magdon-Ismail, "Near optimal column-based matrix reconstruction," in *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS'11)*, 2011, pp. 305 –314.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[9] A. Frieze, R. Kannan, and S. Vempala, "Fast Monte-Carlo algorithms for finding low-rank approximations," in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS'98)*, 1998, pp. 370 –378.

[10] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering large graphs via the singular value decomposition," *Machine Learning*, vol. 56, no. 1-3, pp. 9–33, 2004.

[11] P. Drineas, R. Kannan, and M. Mahoney, "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 158–183, 2007.

[12] P. Drineas, M. Mahoney, and S. Muthukrishnan, "Subspace sampling and relative-error matrix approximation: Column-based methods," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer Berlin / Heidelberg, 2006, pp. 316–326.

[13] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, "Matrix approximation and projective clustering via volume sampling," *Theory of Computing*, vol. 2, no. 1, pp. 225–247, 2006.

[14] A. Çivril and M. Magdon-Ismail, "Column subset selection via sparse approximation of SVD," *Theoretical Computer Science*, vol. 421, no. 0, pp. 1 – 14, 2012.

[15] A. K. Farahat, A. Ghodsi, and M. S. Kamel, "An efficient greedy method for unsupervised feature selection," in *Proceedings of the Eleventh IEEE International Conference on Data Mining (ICDM'11)*, 2011, pp. 161 –170.

[16] ——, "Efficient greedy feature selection for unsupervised learning," *Knowledge and Information Systems*, vol. 35, no. 2, pp. 285–310, 2013.

[17] T. Elsayed, J. Lin, and D. W. Oard, "Pairwise document similarity in large collections with MapReduce," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers (HLT'08)*, 2008, pp. 265–268.

[18] A. Ene, S. Im, and B. Moseley, "Fast clustering using MapReduce," in *Proceedings of the Seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, 2011, pp. 681–689.

[19] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, 2010, pp. 938–948.

[20] S. Dasgupta and A. Gupta, "An elementary proof of a theorem of Johnson and Lindenstrauss," *Random Structures and Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.

[21] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," *Journal of computer and System Sciences*, vol. 66, no. 4, pp. 671–687, 2003.

[22] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the Twelfth ACM SIGKDD international conference on Knowledge Discovery and Data Mining (KDD'06)*, 2006, pp. 287–296.

[23] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins Univ Pr, 1996.

[24] A. Deshpande and L. Rademacher, "Efficient volume sampling for row/column subset selection," in *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, 2010, pp. 329 –338.

[25] V. Guruswami and A. K. Sinop, "Optimal column-based low-rank matrix reconstruction," in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, 2012, pp. 1207–1214.

[26] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *The Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[27] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Chang, "Parallel spectral clustering in distributed systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 3, pp. 568 –586, 2011.

[28] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 11, pp. 1958–1970, 2008.

[29] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert, "An algorithm for the principal component analysis of large data sets," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2580–2594, 2011.