Machine Learning, 56, 9–33, 2004

Clustering Large Graphs via the Singular Value Decomposition*

P. DRINEAS[†] drinep@cs.rpi.edu
Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180

A. FRIEZE[‡] alan@random.math.cmu.edu
Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213

R. KANNAN[§] kannan@cs.yale.edu
Computer Science Department, Yale University, New Haven, CT 06520

S. VEMPALA[¶] vempala@math.mit.edu
Department of Mathematics, M.I.T., Cambridge, MA 02139

V. VINAY vinay@csa.iisc.ernet.in
Indian Institute of Science, Bangalore, India

Editors: Nina Mishra and Rajeev Motwani

Abstract. We consider the problem of partitioning a set of m points in the n-dimensional Euclidean space into k clusters (usually m and n are variable, while k is fixed), so as to minimize the sum of squared distances between each point and its cluster center. This formulation is usually the objective of the k-means clustering algorithm (Kanungo et al. (2000)). We prove that this problem in NP-hard even for $k = 2$, and we consider a continuous relaxation of this discrete problem: find the k-dimensional subspace V that minimizes the sum of squared distances to V of the m points. This relaxation can be solved by computing the Singular Value Decomposition (SVD) of the $m \times n$ matrix A that represents the m points; this solution can be used to get a 2-approximation algorithm for the original problem. We then argue that in fact the relaxation provides a generalized clustering which is useful in its own right.

Finally, we show that the SVD of a random submatrix—chosen according to a suitable probability distribution—of a given matrix provides an approximation to the SVD of the whole matrix, thus yielding a very fast randomized algorithm. We expect this algorithm to be the main contribution of this paper, since it can be applied to problems of very large size which typically arise in modern applications.

Keywords: Singular Value Decomposition, randomized algorithms, k-means clustering

[*] A preliminary version of this work appeared in the 1999 ACM-SIAM Symposium on Discrete Algorithms.

[†] This work was done while the author was a graduate student at Yale University and was supported by NSF Grant CCR-9820850.

[‡] Supported in part by NSF Grant CCR-9530974.

[§] Supported in part by NSF Grant CCR-9820850.

[¶] Supported in part by NSF Grant CCR-6895000 and a Sloan Foundation Fellowship.

1. Introduction

In this paper we address the problem of clustering the rows of a given $m \times n$ matrix A—i.e., the problem of dividing up the set of rows into k clusters where each cluster has "similar" rows. Our notion of similarity of two rows (to be discussed in detail below) will be a function of the length of the vector difference of the two rows. So, equivalently, we may view the problem geometrically—i.e., we are given m points in the n-dimensional Euclidean space and we wish to divide them up into k clusters, where each cluster contains points which are "close to each other". This problem includes as a special case the problem of clustering the vertices of a (directed or undirected) graph, where the matrix is just the adjacency matrix of the graph. Here the dissimilarity of two vertices depends on the number of neighbors that are *not* in common.

There are many notions of similarity and many notions of what a "good" clustering is in the literature. In general, clustering problems turn out to be NP-hard; in some cases, there are polynomial-time approximation algorithms. Our aim here is to deal with very large matrices (with more than 10^5 rows and columns and more than 10^6 non-zero entries), where a polynomial time bound on the algorithm is not useful in practice. Formally, we deal with the case where m and n vary and k (the number of clusters) is fixed; we seek linear time algorithms (with small constants) to cluster such data sets.

We will argue that the basic Singular Value Decomposition (SVD) of matrices provides us with an excellent tool. We will first show that SVD helps us approximately solve the clustering problem described in the abstract (Section 3); unfortunately, the running time of this algorithm is a polynomial of high degree. However, we then argue that the SVD *itself* directly solves the relaxation of the clustering problem, as described in the abstract, and that it gives us what we call a "generalized clustering", where each point will belong to a cluster with a certain "intensity" and clusters are not necessarily disjoint. Using basic Linear Algebra, we show some natural properties of such generalized clusterings (Section 4).

Finally, we develop a linear time randomized algorithm for approximate SVD, and thus for approximate "generalized clusterings", which makes the procedure feasible for the very large matrices (Section 5). Our algorithm is inspired by the work of Frieze, Kannan, and Vempala (1998) and essentially approximates the top few left[1] singular vectors (as well as the corresponding singular values) of a matrix A. We expect this algorithm to be useful in a variety of settings (e.g. data clustering and information retrieval (Berry & Linoff, 1997; Jambu & Lebeaux, 1983), property testing of graphs (Goldreich, Goldwasser, & Ron, 1998), image processing (Andrews & Patterson, 1976a, 1976b; Huang & Narendra, 1974), etc.).

1.1. *The discrete clustering problem*

Consider the following clustering problem, whose formulation is usually the objective of the k-means clustering algorithm: we are given m points $\mathcal{A} = \{A_{(1)}, A_{(2)}, \dots A_{(m)}\}$ in an n-dimensional Euclidean space and a positive integer k, where k will be considered to be fixed as m and n vary. The problem is to find k points $\mathcal{B} = \{B_{(1)}, B_{(2)}, \dots, B_{(k)}\}$, also in

n-dimensional space, such that

$$f_{\mathcal{A}}(\mathcal{B}) = \sum_{i=1}^{m} \big(\mathbf{dist}^2\big(A_{(i)}, \mathcal{B}\big)\big)$$

is minimized. Here $\mathbf{dist}(A_{(i)}, \mathcal{B})$ is the (Euclidean) distance of $A_{(i)}$ to its nearest point in $\mathcal{B}$. Thus, in this problem we wish to minimize the sum of squared distances to the nearest "cluster center". This measure of clustering quality is also called the squared error distortion (Jain & Dubes, 1988; Gersho & Gray, 1991) and comes under the category of variance-based clustering (we will highlight the connection presently). We call this the "Discrete Clustering Problem" (DCP), to contrast it from an analogous continuous problem. The DCP is NP-hard even for $k = 2$ (via a reduction from minimum bisection, see Section 3.1).

Note that a solution to the DCP defines k clusters S_j, $j = 1, \dots k$. The cluster center $B^{(j)}$ will be the centroid of the points in S_j. To prove this notice that for any set of points $\mathcal{S} = \{X_{(1)}, X_{(2)}, \dots, X_{(r)}\}$ and any point B we have

$$\sum_{i=1}^{r} \big\|X^{(i)} - B\big\|^2 = \sum_{i=1}^{r} \big\|X^{(i)} - \bar{X}\big\|^2 + r\|B - \bar{X}\|^2, \tag{1}$$

where $\|X^{(i)} - B\|$ denotes the Euclidean distance between points $X^{(i)}$ and B (which, of course, is equal to the 2-norm of the vector $X^{(i)} - B$), and $\bar{X}$ is the centroid

$$\bar{X} = \frac{1}{r}\big(X^{(1)} + X^{(2)} + \cdots + X^{(r)}\big)$$

of $\mathcal{S}$. Notice that (1) may be restated as

$$\sum_{i=1}^{r}\sum_{j=1}^{n} \big(X_j^{(i)} - B_j\big)^2 = \sum_{i=1}^{r}\sum_{j=1}^{n} \left(X_j^{(i)} - r^{-1}\sum_{k=1}^{r} X_j^{(k)}\right)^2 + r\sum_{j=1}^{n}\left(B_j - r^{-1}\sum_{k=1}^{r} X_j^{(k)}\right)^2,$$

where B_j denotes the jth element of the vector B (and similarly for the $X^{(i)}$). The proof of the above formula amounts to straightforward but somewhat tedious algebraic manipulations. Thus, the DCP is the problem of partitioning a set of points into clusters so that the *sum of the variances of the clusters* is minimized.

We define a relaxation which we call the "Continuous Clustering Problem" (CCP): find the subspace V of $\mathcal{R}^n$, of dimension at most k, which minimizes

$$g_{\mathcal{A}}(V) = \sum_{i=1}^{m} \mathbf{dist}^2\big(A_{(i)}, V\big).$$

It can be shown that the optimal value of DCP is an upper bound for the optimal value of the CCP. Indeed for any set $\mathcal{B}$ of k points,

$$f_{\mathcal{A}}(\mathcal{B}) \geq g_{\mathcal{A}}(V_{\mathcal{B}}) \tag{2}$$

where $V_{\mathcal{B}}$ is the subspace generated by the points in $\mathcal{B}$.

It will follow from standard Linear Algebra that the continuous clustering problem can be *exactly* solved in polynomial time, since the optimal subspace can be read off from the Singular Value Decomposition (SVD) of the matrix A containing $A_{(1)}, A_{(2)}, \ldots, A_{(m)}$ as its rows. One can now attempt to solve DCP as follows: first solve CCP to find a k-dimensional subspace V; then project the problem to V and solve the discrete clustering problem in the k-dimensional space (we emphasize that k is now fixed). In Section 3.2 we will show that the k-dimensional problem can be solved exactly in polynomial time (actually, the running time is exponential in k, but k is a fixed constant) and this will give us a 2-approximation algorithm for DCP.

1.2. Generalized clustering

In Section 4, we will argue that the optimal subspace that is returned from the "Continuous Clustering Problem" (CCP) yields a "generalized clustering" of the matrix A. A "generalized clustering" differs in two respects from a normal clustering: *first*, each cluster instead of being a subset of the rows of A (or equivalently an m-vector whose entries are all 0 or 1), is an m-vector of reals where the ith component gives the "intensity" with which the ith point belongs to the cluster. *Second*, the requirement that the clusters be disjoint in the discrete clustering is replaced by a requirement that the vectors corresponding to the different clusters be orthogonal. We will argue that this notion of clustering is quite natural and that it has certain desirable features not allowed by discrete clustering; for example, it allows having overlapping clusters.

1.3. A fast Monte Carlo algorithm for singular value decomposition

Given an $m \times n$ matrix A, we develop a linear time randomized algorithm that approximates a few of the top singular vectors and singular values of A (see Section 2 for background). This algorithm renders the computation of singular values and singular vectors feasible for the very large matrices in modern applications.

Recall that for any $m \times n$ matrix X,

$$\|X\|_F^2 = \sum_{i,j} A_{ij}^2 \quad \text{and} \quad \|X\|_2 = \max_{x \in \mathcal{R}^n : \|x\|=1} \|Xx\|.$$

Our goal is to find an approximation P to A, such that the rank of P is at most k, satisfying (with high probability)

$$\|A - P\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2 \tag{3}$$

$$\|A - P\|_2^2 \leq \|A - A_k\|_2^2 + \epsilon \|A\|_F^2 \tag{4}$$

where A_k is the "optimal" rank k approximation to A and $\epsilon > 0$ a given error parameter. More specifically, A_k is a matrix of rank k such that for all matrices D of rank at most k (see Section 2),

$$\|A - A_k\|_F \leq \|A - D\|_F \quad \text{and} \quad \|A - A_k\|_2 \leq \|A - D\|_2.$$

Thus, the matrix P is *almost* the best rank k approximation to A in the sense described above. The matrix P returned by our algorithm is equal to the product $HH^T A$, where H is an $m \times k$ matrix containing approximations to the top k left singular vectors of A. We remind the reader that A_k can be written as $A_k = U_k U_k^T A$, where U_k is an $m \times k$ matrix containing the exact top k left singular vectors of A. Thus, our algorithm approximates A_k by approximating the matrix U_k by H, or, equivalently, by approximating the top k left singular vectors of A.

There are many algorithms that either exactly compute the SVD of a matrix in $O(mn^2 + m^2n)$ time (an excellent reference is Golub & Van Loan, 1989) or iteratively approximate a few of the top singular vectors and the corresponding singular values (e.g. Lanczos methods). We should note here that Lanczos methods are iterative techniques which—given enough running time—converge to the exact solution (except for some special cases); however, the speed of convergence depends on various factors. We will not elaborate on this topic here; instead we refer the reader to Parlett (1997) and the references therein.

In this paper, we propose a simple, randomized SVD algorithm: instead of computing the SVD of the entire matrix, pick a subset of its rows or columns, scale them appropriately and compute the SVD of this smaller matrix. We will prove that this process returns efficient approximations to the singular values and singular vectors of the original matrix; more specifically, by picking rows, we approximate *right* singular vectors, while, by picking columns, we approximate *left* singular vectors.

Our algorithm will make two passes through the entire matrix, sample columns (or rows) with probabilities proportional to the square of their lengths and then run in time

$$O\left(\frac{k^2}{\epsilon^4}m + \frac{k^3}{\epsilon^6}\right),$$

to satisfy both (3) and (4). If we are *only* interested in satisfying (4), the running time of the algorithm is significantly smaller:

$$O\left(\frac{1}{\epsilon^4}m + \frac{1}{\epsilon^6}\right).$$

Thus, there is no dependency on k. Also, if the matrix is sparse, we can replace m in the running times by m', where m' is the maximum number of non-zero entries in a column of A. Alternatively, our result might be viewed as showing that instead of computing the SVD of the entire matrix A, it is sufficient to compute the SVD of a matrix consisting of $O(k/\epsilon^2)$ randomly picked columns of A, after appropriate scaling. The columns must be picked with probability proportional to their length squared. We should note again that $O(k/\epsilon^2)$ columns suffice in order to satisfy both (3) and (4); if we are *only* interested in satisfying (4), $O(1/\epsilon^2)$ columns are enough.

Our algorithm is directly motivated by the work of Frieze, Kannan, and Vempala (1998) which also presents an algorithm that achieves (3), with running time $O(k^{12}/\epsilon^9)$ and returns a "*description*" to the left[2] singular vectors, namely, it describes the left singular vectors as a matrix-vector product. Thus, while it is theoretically very interesting that the algorithm of Frieze, Kannan, and Vempala (1998) has a running time which does not grow with m or n, the dependence on k and ϵ might be too large in practice. Also, explicitly computing the left singular vectors would make the running time linear in m.

Clearly an approximation of the form (3) or (4) is only useful if A has a good approximation of "small" rank k and further m and n are large (so exact algorithms are not feasible.) There are many examples of situations where these conditions prevail (i.e., information retrieval applications). As an example, our algorithm could be used to perform Latent Semantic Indexing (LSI) (Berry & Linoff, 1997; Papadimitriou et al., 2000; Azar et al., 2001); this is a general technique for analyzing a collection of documents which are assumed to be related (for example, they are all documents dealing with a particular subject). Suppose there are m documents and n terms which occur in the documents. The model hypothesizes that, because of relationships among the documents, there is a small number—say k—of main (unknown) topics which describe the documents. The first aim of the technique is to find a set of k topics which best describe the documents; a topic is modelled as an n-vector of non-negative reals. The interpretation is that the jth component of a topic vector denotes the frequency with which the jth term occurs in a discussion of the topic. With this model at hand, it can be argued that the k best topics are the top k singular vectors of the so-called "document-term" matrix, which is an $m \times n$ matrix A with A_{ij} being the frequency of the jth term in the ith document.

2. Linear algebra background

Any $m \times n$ matrix A can be expressed as

$$A = \sum_{t=1}^{r} \sigma_t(A) u^{(t)} v^{(t)^T},$$

where r is the rank of A, $\sigma_1(A) \geq \sigma_2(A) \geq \cdots \geq \sigma_r(A) > 0$ are its singular values and $u^{(t)} \in \mathcal{R}^m$, $v^{(t)} \in \mathcal{R}^n$, $t = 1, \ldots, r$ are its left and right singular vectors respectively. The $u^{(t)}$'s and the $v^{(t)}$'s are orthonormal sets of vectors; namely, $u^{(i)^T} u^{(j)}$ is one if $i = j$ and zero otherwise. We also remind the reader that

$$\|A\|_F^2 = \sum_{i,j} A_{ij}^2 = \sum_{i=1}^{r} \sigma_i^2(A)$$

$$\|A\|_2 = \max_{x \in \mathcal{R}^n : \|x\|=1} \|Ax\| = \max_{x \in \mathcal{R}^m : \|x\|=1} \|x^T A\| = \sigma_1(A)$$

In matrix notation, SVD is defined as $A = U\Sigma V^T$ where U and V are orthogonal (thus $U^T U = I$ and $V^T V = I$) matrices of dimensions $m \times r$ and $n \times r$ respectively, containing the left and right singular vectors of A. $\Sigma = \mathbf{diag}(\sigma_1(A), \ldots, \sigma_r(A))$ is an $r \times r$ diagonal matrix containing the singular values of A.

If we define $A_k = \sum_{t=1}^{k} \sigma_t u^{(t)} v^{(t)^T}$, then A_k is the best rank k approximation to A with respect to the 2-norm and the Frobenius norm. Thus, for any matrix D of rank at most k, $\|A - A_k\|_2 \leq \|A - D\|_2$ and $\|A - A_k\|_F \leq \|A - D\|_F$. A matrix A has a "good" rank k approximation if $A - A_k$ is small with respect to the 2-norm and the Frobenius norm. It is well known that

$$\|A - A_k\|_F^2 = \sum_{t=k+1}^{r} \sigma_t^2(A) \quad \text{and} \quad \|A - A_k\|_2 = \sigma_{k+1}(A).$$

From basic Linear Algebra, $A_k = U_k \Sigma_k V_k^T = A V_k V_k^T = U_k U_k^T A$, where U_k and V_k are sub-matrices of U and V, containing only the top k left or right singular vectors of A respectively; for a detailed treatment of Singular Value Decomposition see Golub and Van Loan, 1989. Also, $\mathbf{Tr}(A)$ denotes the sum of the diagonal elements of A; it is well-known that $\|A\|_F^2 = \mathbf{Tr}(AA^T)$ for any $m \times n$ matrix A.

In the following, $A_{(i)}$ denotes the ith row of A as a row vector and $A^{(i)}$ denotes the ith column of A as a column vector. The length of a column (or row) vector will be denoted by $\|A^{(i)}\|$ or $(\|A_{(i)}\|)$ and is equal to the square root of the sum of the squares of its elements.

3. Discrete clustering

3.1. Complexity

Theorem 1. *DCP is NP-hard for $k \geq 2$.*

Proof: We will prove the NP-hardness for $k = 2$ via a reduction from minimum bisection, the problem of partitioning a graph into two equal-sized parts so as to minimize the number of edges going between the two parts. The proof for $k > 2$ is similar, via a reduction from the minimum k-section problem.

Let $G = (V, E)$ be the given graph with n vertices $1, \ldots, n$, with n even. Let $d(i)$ be the degree of the i'th vertex. We will map each vertex of the graph to a point with $|E| + |V|$ coordinates. There will be one coordinate for each edge and one coordinate for each vertex. The vector X^i for a vertex i is defined as $X^i(e) = 1$ if e is adjacent to i and 0 if e is not adjacent to i; in addition $X^i(i) = M$ and $X^i(j) = 0$ for all $j \neq i$. The parameter M will be set to be n^3.

Consider a partition into two parts P, Q with p and q vertices respectively. Let B_p and B_q be the cluster centers. Consider the DCP value for the partition P, Q on just the last n coordinates, we have $B_p(i) = M/p$ and $B_q(i) = M/q$ and so the DCP value on these coordinates is

$$pM^2\left(1-\frac{1}{p}\right)^2+qM^2\left(1-\frac{1}{q}\right)^2=nM^2+M^2\left(\frac{1}{p}+\frac{1}{q}\right)-4M^2.$$

If $p=q=n/2$, then this is $(n-4+4/n)M^2$. On the other hand, if $p\neq q$, then it is at least

$$\left(n-4+\frac{4n}{n^2-4}\right)M^2$$

and so the increase in the DCP value on the last n coordinates is at least $16M^2/n(n^2-4)$. The first $|E|$ coordinates contribute at most $4n^2$ to the DCP. So if we have $M\geq n^3$ (say), then the optimal solution will always choose $p=q$, since any gain by not having $p=q$ is subsumed by the loss in the DCP value over the last n coordinates.

Now, the sum of pairwise squared distances within a cluster can be rewritten as follows:

$$\begin{aligned}\sum_{i,j\in P}|X^i-X^j|^2&=2p\sum_{i\in P}|X^i|^2-2\left(\sum_{i\in P}X^i\right)^2\\&=2p\sum_{i\in P}|X^i-B_p|^2\end{aligned}$$

Therefore,

$$p\sum_{i\in P}|X^i-B_p|^2+q\sum_{i\in Q}|X^i-B_q|^2=\sum_{i<j\in P}|X^i-X^j|^2+\sum_{i<j\in Q}|X^i-X^j|^2$$

If $p=q$, then this exactly $n/2$ times the value of the DCP. The RHS, namely the sum of pairwise distances within clusters, can be evaluated separately for the coordinates corresponding to edges and those corresponding to vertices. For the former, for a pair i, j in the same cluster it is $d(i)+d(j)$ if (i,j) is not an edge and $d(i)+d(j)-2$ if (i,j) is an edge. Therefore the total is

$$\begin{aligned}&\sum_{(i,j)\in E,i<j\in P}d(i)+d(j)-2+\sum_{(i,j)\notin E,i<j\in P}d(i)+d(j)+\sum_{i<j\in P}2M^2\\&\quad+\sum_{(i,j)\in E,i<j\in Q}d(i)+d(j)-2+\sum_{(i,j)\notin E,i<j\in Q}d(i)+d(j)+\sum_{i<j\in Q}2M^2\\&=(p-1)\sum_{i\in P}d(i)+(q-1)\sum_{i\in Q}d(i)-2|E|+2|E(P,Q)|\\&\quad+M^2(p(p-1)+q(q-1))\end{aligned}$$

Note that if $p=q$ then this only depends on $|E(P,Q)|$, i.e., the size of the minimum bisection. But then the minimum DCP solution is also a minimum bisection for the original graph. □

3.2. A 2-approximation algorithm for DCP

We first show how to solve DCP in $O(m^{k^2d/2})$ time when the input $\mathcal{A}$ is a subset of $\mathcal{R}^d$; here k and d are considered to be fixed. Each set $\mathcal{B}$ of "cluster centers" defines a Voronoi diagram where cell $\mathcal{C}_i = \{X \in \mathcal{R}^d \colon \|X - B_{(i)}\| \leq \|X - B_{(j)}\| \text{ for } j \neq i\}$ consists of those points whose closest point in $\mathcal{B}$ is $B_{(i)}$. Each cell is a polyhedron and the total number of faces in $C_1, \ldots, C_k$ is no more than $\binom{k}{2}$ (since each face is the set of points equidistant from two points of $\mathcal{B}$). It is not too difficult to see that, without loss of generality, we can move the boundary hyperplanes of the optimal Voronoi diagram, without any face passing through a point of $\mathcal{A}$, so that each face contains at least d points of $\mathcal{A}$.

Assume that the points of $\mathcal{A}$ are in general position and $0 \notin \mathcal{A}$ (a simple perturbation argument deals with the general case); this means that each face now contains d affinely independent points of $\mathcal{A}$. We have lost the information about which side of each face to place these points and so we must try all possibilities for each face. This leads to the following enumerative procedure for solving DCP:

Algorithm for DCP in d dimensions

Input: positive integers k, d and a set of m points $\mathcal{A} \subseteq \mathcal{R}^d$.
Output: a set $\mathcal{B}$ with k points from $\mathcal{R}^d$ such that $f_{\mathcal{A}}(\mathcal{B})$ is minimized.

1. Enumerate all $\sum_{t=k}^{\binom{k}{2}} \binom{\binom{m}{d}}{t} = O(m^{dk^2/2})$ sets of $k \leq t \leq \binom{k}{2}$ hyperplanes, each of which contains d affinely independent points of $\mathcal{A}$.
2. Check that the arrangement defined by these hyperplanes has exactly k cells.
3. Make one of 2^{td} choices as to which cell to assign each point of $\mathcal{A}$ which is lying on a hyperplane.
4. This defines a unique partition of $\mathcal{A}$. Find the centroid of each set in the partition and compute $f_{\mathcal{A}}$.

We now examine how CCP can be solved by Linear Algebra in polynomial time, for any values of m, n and k. Indeed, let V be a k-dimensional subspace of $\mathcal{R}^n$ and $\bar{A}_{(1)}, \ldots, \bar{A}_{(m)}$ be the orthogonal projections of $A_{(1)}, \ldots, A_{(m)}$ onto V. Let $\bar{\mathcal{A}}$ be the $m \times n$ matrix with rows $\bar{A}_{(1)}, \ldots, \bar{A}_{(m)}$. Thus $\bar{\mathcal{A}}$ has rank at most k and

$$\|A - \bar{A}\|_F^2 = \sum_{i=1}^{m} \left\|A_{(i)} - \bar{A}_{(i)}\right\|^2 = \sum_{i=1}^{m} \mathbf{dist}^2\left(A_{(i)}, V\right).$$

Thus to solve CCP, all we have to do is compute the top k right singular vectors of A, since it is known that these minimize $\|A - \bar{A}\|_F^2$ over all rank k matrices; let V_{SVD} denote the subspace spanned by the top k right singular vectors of A.

We now show that combining the above two ideas gives a 2-approximation algorithm for DCP. Let $\bar{\mathcal{A}} = \{\bar{A}_{(1)}, \ldots, \bar{A}_{(m)}\}$ be the projection of $\mathcal{A}$ onto the subspace V_{SVD} above. Let

$\bar{\mathcal{B}} = \{\bar{B}_{(1)}, \dots, \bar{B}_{(k)}\}$ be the optimal solution to DCP with input $\bar{\mathcal{A}}$. We emphasize that DCP will run on a k-dimensional subspace, where k is fixed.

```
Algorithm for general DCP

1. Compute $V_{\mathrm{SVD}}$.
2. Solve DCP on the set of $m$ points $\bar{\mathcal{A}} \subseteq \mathcal{R}^k$ (thus $d = k$), in order to obtain $\bar{\mathcal{B}}$, a set
   of $k$ centroids in $\mathcal{R}^k$.
3. Output $\bar{\mathcal{B}}$.
```

Notice that the running time of the second step is $O(m^{k^3/2})$.

Lemma 1. *The above algorithm returns a* 2*-approximation for DCP.*

Proof: It follows from (2) that the optimal value $Z_{\mathcal{A}}^{\mathrm{DCP}}$ of the DCP satisfies

$$Z_{\mathcal{A}}^{\mathrm{DCP}} \geq \sum_{i=1}^{m} \left\| A_{(i)} - \bar{A}_{(i)} \right\|^2. \tag{5}$$

Observe that if $\hat{\mathcal{B}} = \{\hat{B}^{(1)}, \dots, \hat{B}^{(k)}\}$ is an optimal solution to the DCP and $\bar{\mathcal{B}}$ consists of the projection of the points in $\hat{\mathcal{B}}$ onto V

$$Z_{\mathcal{A}}^{\mathrm{DCP}} = \sum_{i=1}^{m} \mathbf{dist}^2\left(A_{(i)}, \hat{\mathcal{B}}\right) \geq \sum_{i=1}^{m} \mathbf{dist}^2\left(\bar{A}^{(i)}, \bar{\mathcal{B}}\right).$$

Combining this with (5) we get

$$\begin{aligned} 2Z_{\mathcal{A}}^{\mathrm{DCP}} &\geq \sum_{i=1}^{m} \left(\left\| A_{(i)} - \bar{A}_{(i)} \right\|^2 + \mathbf{dist}^2\left(\bar{A}_{(i)}, \bar{\mathcal{B}}\right) \right) \\ &= \sum_{i=1}^{m} \mathbf{dist}^2\left(A_{(i)}, \bar{\mathcal{B}}\right) \\ &= f_{\mathcal{A}}(\bar{\mathcal{B}}), \end{aligned}$$

proving that we do indeed get a 2-approximation. □

4. Generalized clusters and SVD

In this section, we will argue that there is a natural way of generalizing clusters which leads to singular vectors. To do this, we introduce a typical motivation: suppose we wish to analyze the structure of a large portion of the web. Consider the underlying directed graph with one vertex per URL and an edge from vertex i to vertex j if there is a hypertext link from i to j. It turns out to be quite useful to cluster the vertices of this graph. Obviously, very large graphs can arise in this application and traditional heuristics (even polynomial time ones) are not good enough.

Given this directed graph $G(V, E)$, we wish to divide the vertex set into "clusters" of "similar" vertices. Since all our information is in the graph, we define two vertices to be "similar" if they share a lot of common neighbors. In general, we assume that all the relevant information is captured by the adjacency matrix A of the graph G; we do not dwell on the "modelling" part, indeed we assume that the translation of the real problem into the graph/matrix has already been done for us.

We now examine how "similarity" may be precisely defined. For this purpose, it is useful to think of the web example, where an edge from i to j means that i thinks of j as important. So, intuitively, similar vertices "reinforce" each others opinions of which web pages are important.

Quite often by "clustering", one means the partition of the node set into subsets of similar nodes. A partition though is too strict, since it is quite common to have overlapping clusters. Also, in traditional clustering, a cluster is a subset of the node set; essentially, the characteristic vector of a cluster is a 0-1 vector. Again, this is too strict, since different nodes may belong to a cluster with different "intensities". For example, if $N(v)$ (the set of neighbors of v) is large and there are many nodes u such that $N(u)$ is a subset of $N(v)$, then a good cluster—intuitively—would include v and many of the u 's (for reinforcement), *but*—again intuitively—v is more important in the cluster than the u's.

Given an $m \times n$ matrix A, we define a cluster x as an m-vector of reals. So, $x(u)$ is the "intensity" with which u belongs to x. We are also interested in assigning a "weight" (or importance) to the cluster x: a crucial quantity in this regard is the vector $x^T A$ because $(x^T A)_i$ is the "frequency" of occurrence of node i in the neighborhood of the cluster x. So, high values of $|(x^T A)_i|$ mean high reinforcement. Thus,

$$\sum_{i=1}^{n}(x^T A)_i^2 = \|x^T A\|_2^2$$

is a measure of the importance of the cluster represented by vector x. We also note that if x is scaled by some constant λ, so is every component of $x^T A$. We now make the following definition:

Definition 1. A **cluster** of the matrix A is an m-vector x with $\|x\| = 1$. The **weight of the cluster** x (denoted by $W(x)$) is $\|x^T A\|$ (the Euclidean length of the vector).

We now argue the reasoning behind using Euclidean lengths in the above definition. While we cannot exhaustively discuss all other possible measures, we look at two other obvious norms: l_∞ (the maximal element of the vector in absolute value) and l_1 (the sum of the absolute values of the elements in the vector). The following examples illustrate the advantage of Euclidean norm over these two and carry over for many other norms.

In the definition, we used the Euclidean norm for both x and $x^T A$. First suppose we used instead the l_1 norm for x. Then, if there are k nodes of G in the same neighborhood, putting $x_i = 1$ for one of them and zero for the others *or* putting $x_i = 1/k$ for each of them, returns the same value for $x^T A$ and so the same weight. However, we prefer larger clusters (thus larger values of $\|x^T A\|$) since they guarantee greater reinforcement. It can be shown that if we restrict to $\|x\| = 1$, then we would choose the larger cluster. Similarly if the l_∞ is used for x, then we shall always have $x_i = 1$ for all i being the maximum weight cluster, which obviously is not always a good choice. It can also be shown that if the l_∞ norm is used for $\|x^T A\|$, then x will be based only on the highest in-degree node which is not always desirable either. A similar example can be provided for the case when the l_1 norm is used for $x^T A$.

Having defined the weight of a cluster, we next want to describe a decomposition process that successively removes the maximum weight cluster from the graph. Let u be the maximum weight cluster and v be any other cluster. We can express v as $v = \lambda u + w$ where λ is a scalar and w is **orthogonal** to u; then, it is known from Linear Algebra that $w^T A$ is also **orthogonal** to $u^T A$, thus

$$\|v^T A\|^2 = \lambda^2 \|u^T A\|^2 + \|w^T A\|^2.$$

It is obvious that as λ grows the weight of v grows as well. Thus, to get a "good" clustering, we can not merely require v to be different from u, since it may be arbitrarily close to u. This observation leads us to the correct requirement; namely that v is required to be orthogonal to u. In our generalized clustering, the orthogonality requirement replaces the traditional disjointness requirement. We now make a second definition.

Definition 2. An optimal clustering of A is a set of orthonormal vectors $x^{(1)}, x^{(2)}, \ldots$ so that $x^{(i)}$ is a maximum weight cluster of A subject to being **orthogonal** to $x^{(1)}, \ldots x^{(i-1)}$.

It can now be shown (directly from Linear Algebra) that corresponding to "removing" the first k clusters is the operation of subtracting the $m \times n$ matrix

$$\sum_{t=1}^{k} x^{(t)} x^{(t)^T} A$$

from A. So if

$$R^{(k)} = A - \sum_{t=1}^{k} x^{(t)} x^{(t)^T} A,$$

$R^{(k)}$ defines a "residual" graph after removing the first k clusters; more specifically, it represents a weighted graph with edge weights. The intuition is that if the first few clusters are of large weight, then the residual matrix will have small norm. We can quantify this using basic Linear Algebra and noting that $\sum_{t=1}^{k} x^{(t)} x^{(t)^T} A$ is a matrix of rank k:

Lemma 2. *$R^{(k)}$ has the least Frobenius norm* **and** *the least 2-norm among all matrices of the form $A - D$, where the rank of D is at most k.*

So, the optimal clustering makes the "error" matrix $R^{(k)}$ as small as possible in two natural ways; the optimal clusters $x^{(1)}, \ldots, x^{(k)}$ are essentially the left singular vectors of A and they may be computed through the Singular Value Decomposition. We note here that we defined the weights of clusters by looking at the out-degrees of the nodes of the graph; symmetrically, we may look at the in-degrees. Linear Algebra and elementary properties of the Singular Value Decomposition tell us that an optimal clustering with respect to in-degrees yields also an optimal clustering with respect to out-degrees and vice versa.

We should note here that these aspects of clustering in the context of the web graph, as well as the introduction of the SVD technique to cluster such graphs was pioneered in Kleinberg (1998) (see also Gibson, Kleinberg, & Raghavan, 1998). Kleinberg (1998) argues—and we do not reproduce the exact argument here—that given the adjacency matrix of a large subgraph of the web graph, where nodes in the graph correspond to web pages that were returned from a search engine as results of a specific query, it is desirable to find the top few singular vectors of that matrix. Roughly, the reason that we are interested in the top few singular vectors is that they correspond to different meanings of the query. Since A is large (in the examples of Kleinberg (1998), in the hundreds of thousands), Kleinberg (1998) judiciously chooses a small submatrix of A and computes only the singular vectors of this submatrix. In the next section, we prove that choosing a submatrix according to a simple probability distribution, returns good approximations to the top few singular vectors and the corresponding singular values.

5. The randomized SVD algorithm

Given an $m \times n$ matrix A we seek to approximate its top k left singular values/vectors. Intuitively, our algorithm picks c columns of A, scales them appropriately, forms an $m \times c$ matrix C and computes its left singular vectors. If A is an objects-features matrix (the (i, j)th entry of A denotes the importance of feature j for object i), our algorithm may be seen as picking a few features (coordinates) with respect to a certain probability distribution, dropping the remaining features, and then doing Principal Component Analysis on the selected features.

Suppose $p_i,\ i = 1, \ldots, n$ are nonnegative reals summing to 1 and satisfying

$$p_i = \left\|A^{(i)}\right\|^2 \big/ \|A\|_F^2. \tag{6}$$

```
**Fast SVD** Algorithm
**Input:** $m \times n$ matrix $A$, integers $c \leq n$, $k \leq c$, $\{p_i\}_{i=1}^{n}$.
**Output:** $m \times k$ matrix $H$, $\lambda_1, \ldots, \lambda_k \in \mathcal{R}^+$.

1. **for** $t = 1$ **to** $c$
     – Pick an integer from $\{1 \ldots n\}$, where $\mathbf{Pr}(\text{pick } i) = p_i$.
     – Include $A^{(i)}/\sqrt{cp_i}$ as a column of $C$.
2. Compute the top $k$ left singular vectors of $C$ (denoted by $h^{(1)}, h^{(2)}, \ldots, h^{(k)}$).
3. Return $H$, a matrix whose columns are the $h^{(t)}$, and $\lambda_1 = \sigma_1(C), \ldots, \lambda_k = \sigma_k(C)$
   (our approximations to the top $k$ singular values of $A$).
```

We should note here that our results also hold—with some small loss in accuracy—if the p_i are nonnegative reals, summing to 1 and satisfying

$$p_i \geq \beta \left\| A^{(i)} \right\|^2 / \|A\|_F^2, \tag{7}$$

where $\beta \leq 1$ is a positive constant, allowing us some flexibility on sampling the columns of A. For simplicity of presentation, we shall focus on the former case, since the analysis of the latter case is essentially the same. We also note that a sampler which samples the columns with probabilities proportional to their lengths squared is simple to construct after one pass through the matrix A (see Section 5.1).

In the above, $\sigma_t(C)$ are the singular values of C. From elementary Linear Algebra we know that $\sigma_t^2(C)$ are the singular values of CC^T *or* C^TC. We should note here that computing the top k left singular vectors of C (step 2) may be easily done in time $O(mc^2)$. For the sake of completeness, we briefly outline this process: compute C^TC ($O(mc^2)$ time) and its singular value decomposition ($O(c^3)$ time). Say that

$$C^TC = \sum_{t=1}^{c} \sigma_t^2(C) w^{(t)} w^{(t)^T}.$$

Here $w^{(t)}$, $t = 1, \ldots, c$ are the right (and left) singular vectors of C^TC. Then, from elementary Linear Algebra, the $w^{(t)}$ are the right singular vectors of C; thus, the $h^{(t)} = Cw^{(t)}/\sigma_t(C), t = 1 \ldots k$ are the left singular vectors of C and they can be computed in $O(mck)$ time.

The algorithm is simple and intuitive; the only part that requires further attention is the sampling process. We emphasize here that the probability distribution described in (6) is not chosen arbitrarily. More specifically, if $p_i = \|A^{(i)}\|^2/\|A\|_F^2$, we will prove (see Lemma 6) that, among all possible probability distributions, this particular one minimizes the expectation of $\|AA^T - CC^T\|_F$, a quantity that will be crucial in proving the error bounds of the above algorithm. Intuitively, the left singular vectors of CC^T are close approximations to the left singular vectors of AA^T; thus, as $\|AA^T - CC^T\|_F$ decreases, the accuracy of our approximations increases. As we shall see, $\|AA^T - CC^T\|_F$ decreases inversely proportional to the number of columns in our sample.

5.1. Implementation details and running time

An important property of our algorithm is that it can be easily implemented. Its "heart" is an SVD computation of a $c \times c$ matrix (C^TC). Any fast algorithm computing the top k right singular vectors of such a matrix could be used to speed up our algorithm. One should be cautious though; since c is usually of $O(k)$, we might end up seeking approximations to almost all singular vectors of C^TC. It is well-known that in this case full SVD of C^TC is much more efficient than iterative approximation techniques; for a detailed treatment of such issues see Parlett, 1997.

Let's assume that the matrix is presented in a particular general form—which we call the **sparse unordered** representation, in which (only) the non-zero entries are presented as triples (i, j, A_{ij}) in any order. This is suited to applications where multiple agents may write in parts of the matrix to a central database, and we cannot make assumptions about the rules for write-conflict resolution. One example of this may be the "load" matrix, where each of many routers writes into a central database a log of the messages it routed during a day in the form of triples (source, destination, number of bytes). We shall prove that we can decide which columns to include in our sample in one pass through the matrix, using $O(c)$ RAM space. The following two lemmas show how this may be done.

Lemma 3. *Suppose $a_1, a_2, \ldots, a_n$ are n non-negative reals which are read once in this order. Then with $O(c)$ additional storage, we can pick i.i.d. samples $i_1, i_2, \ldots i_c \in \{1, 2, \ldots n\}$ such that*

$$\mathbf{Pr}(i_t = i) = \frac{a_i}{\sum_{j=1}^{n} a_j}.$$

Proof: We argue that we can pick i_1. The others can be done by running c independent copies of this process. To pick i_1, suppose we have read $a_1, a_2, \ldots, a_\ell$ so far and have a sample i_1 such that, for some fixed value $1 \leq i \leq \ell$,

$$\mathbf{Pr}(i_1 = i) = \frac{a_i}{\sum_{j=1}^{\ell} a_j}.$$

We also keep the running sum $\sum_{j=1}^{\ell} a_j$. On reading $a_{\ell+1}$, we just replace the current i_1 with $\ell + 1$ with probability

$$\frac{a_{\ell+1}}{\sum_{j=1}^{\ell+1} a_j}.$$

It can be shown by induction that this works. □

Lemma 4. *In one pass, we can pick i.i.d. samples $i_1, i_2, \ldots, i_c$ drawn according to probabilities p_i satisfying $p_i = \|A^{(i)}\|^2 / \|A\|_F^2$.*

Proof: To pick i_1, just pick (using Lemma 3) an entry (i, j) with probability proportional to its square and take $i_1 = j$. The other i_t are also picked by running c independent experiments simultaneously. Obviously, the overall probability of picking column j in one trial is

$$\sum_{i=1}^{m} \frac{A_{ij}^2}{\|A\|_F^2} = \frac{\|A^{(j)}\|^2}{\|A\|_F^2}$$

□

In the second pass, we pick out the entries of the columns of matrix C that we decided to keep; note that we know the scaling factors since we know the probabilities with which we pick each column. We now analyze the running time requirements of the algorithm. We remind the reader that the algorithm works on matrices in sparse representation, where the matrix is presented as a set of triples (i, j, A_{ij}) with at most one triple for each (i, j). So, the zero entries need not be given; some zero entries may be presented.

Theorem 2. *After the preprocessing step, the running time of the algorithm is* $O(c^2m + c^3)$.

Proof: The scaling of the columns prior to including them in C needs cm operations. Computing C^TC takes $O(c^2m)$ time and computing its SVD $O(c^3)$ time. Finally, we need to compute H, which can be done in $O(mck)$ operations. Thus, the overall running time (excluding the preprocessing step) is $O(c^2m + c^3 + cmk)$, and since $k \leq c$ the result follows. □

5.2. Theoretical analysis

Our analysis will guarantee that $\|A - P\|_F$ is at most $\|A - A_k\|_F$ plus some additional error, which is inversely proportional to the number of columns that we included in our sample. (Similarly, our analysis will guarantee that $\|A - P\|_2$ is at most $\|A - A_k\|_2$ plus some additional error, which is inversely proportional to the number of columns that we included in our sample.) As the "quality" of H improves, H and U_k span *almost* the same space and P is *almost* the optimal rank k approximation to A. We remind the reader that we use A_k to denote the "optimal" rank k approximation to A, and U_k to denote the $m \times k$ matrix whose columns are the top k left singular vectors of A.

Theorem 3. *If* $P = HH^TA$ *is a rank (at most)* k *approximation to* A, *constructed using the algorithm of Section* 5, *then, for any* $c \leq n$ *and* $\delta > 0$,

$$\|A - P\|_F^2 \leq \|A - A_k\|_F^2 + 2\big(1 + \sqrt{8\ln(2/\delta)}\big)\sqrt{\frac{k}{c}}\|A\|_F^2$$

$$\|A - P\|_2^2 \leq \|A - A_k\|_2^2 + 2\big(1 + \sqrt{8\ln(2/\delta)}\big)\sqrt{\frac{1}{c}}\|A\|_F^2$$

hold with probability at least $1 - \delta$. *We assume that* $p_i = \|A^{(i)}\|^2/\|A\|_F^2$ *and sampling is done with replacement.*

Let us note here that using the probabilities of Eq. (7), would result to the following error bounds:

$$\|A - P\|_F^2 \leq \|A - A_k\|_F^2 + 2(1 + \sqrt{8\beta^{-1}\ln(2/\delta)})\sqrt{\frac{k}{\beta c}}\|A\|_F^2$$

$$\|A - P\|_2^2 \leq \|A - A_k\|_2^2 + 2(1 + \sqrt{8\beta^{-1}\ln(2/\delta)})\sqrt{\frac{1}{\beta c}}\|A\|_F^2.$$

Proof: Denote by $h^{(t)}$, $t = 1, \ldots, k$ the top k left singular vectors of C and by $\sigma_t(C)$ the corresponding singular values. Let H denote an $m \times k$ matrix whose columns are the $h^{(t)}$; since the $h^{(t)}$ are orthonormal, $H^T H = I$ and

$$\begin{aligned}\|A - HH^T A\|_F^2 &= \mathbf{Tr}((A^T - A^T HH^T)(A - HH^T A))\\ &= \mathbf{Tr}(A^T A) - \mathbf{Tr}(A^T HH^T A)\\ &= \|A\|_F^2 - \|H^T A\|_F^2\\ &= \|A\|_F^2 - \sum_{t=1}^{k} \left\|A^T h^{(t)}\right\|^2. \end{aligned} \tag{8}$$

Writing AA^T and CC^T both in a coordinate system with $h^{(1)}, \ldots, h^{(k)}$ as the top k coordinate vectors, we see that $h^{(t)^T}(AA^T - CC^T)h^{(t)}$ is the (t, t) entry of $AA^T - CC^T$. So we have

$$\sum_{t=1}^{k} \left(h^{(t)^T}(AA^T - CC^T)h^{(t)}\right)^2 \leq \|AA^T - CC^T\|_F^2$$

or, equivalently (since $C^T h^{(t)} = \sigma_t(C)h^{(t)}$)

$$\sum_{t=1}^{k} \left(\left\|A^T h^{(t)}\right\|^2 - \sigma_t^2(C)\right)^2 \leq \|AA^T - CC^T\|_F^2$$

and, by an application of the Cauchy-Schwartz inequality,

$$\sum_{t=1}^{k} \left(\left\|A^T h^{(t)}\right\|^2 - \sigma_t^2(C)\right) \geq -\sqrt{k}\|AA^T - CC^T\|_F. \tag{9}$$

We now state the well-known Hoffman-Wielandt inequality (see e.g. Golub & Van Loan, 1989). Given symmetric matrices X and Y (of the same dimensions),

$$\sum_{t=1}^{k} (\sigma_t(X) - \sigma_t(Y))^2 \leq \|X - Y\|_F^2,$$

where $\sigma_t(X)$ and $\sigma_t(Y)$ denote the tth singular values of X and Y respectively. The lemma holds for any $k \leq \min\{\text{rank}(X), \text{rank}(Y)\}$. Applying this inequality to the symmetric matrices AA^T and CC^T, we see that

$$\begin{aligned}\sum_{t=1}^{k} (\sigma_t(CC^T) - \sigma_t(AA^T))^2 &= \sum_{t=1}^{k} \left(\sigma_t^2(C) - \sigma_t^2(A)\right)^2\\ &\leq \|AA^T - CC^T\|_F^2\end{aligned}$$

and, by an application of the Cauchy-Schwartz inequality,

$$\sum_{t=1}^{k} \left(\sigma_t^2(C) - \sigma_t^2(A)\right) \geq -\sqrt{k}\|AA^T - CC^T\|_F. \tag{10}$$

Adding (9) and (10), we get

$$\sum_{t=1}^{k} \left(\left\|A^T h^{(t)}\right\|^2 - \sigma_t^2(A)\right) \geq -2\sqrt{k}\|AA^T - CC^T\|_F. \tag{11}$$

We now state the following lemmas, whose proofs may be found in the appendix.

Lemma 5. *If C is created using the algorithm of Section* 5, *then*, *with probability at least* $1 - \delta$ (*for all* $\delta > 0$),

$$\|AA^T - CC^T\|_F \leq \frac{1 + \sqrt{8\ln(2/\delta)}}{\sqrt{c}}\|A\|_F^2.$$

Lemma 6. *Setting the p_i's as in Eq.* (6) *minimizes the expectation of* $\|AA^T - CC^T\|_F^2$.

Thus, using (11) and Lemma 5, with probability at least $1 - \delta$, for all $\delta > 0$,

$$\sum_{t=1}^{k} \left\|A^T h^{(t)}\right\|^2 \geq \sum_{t=1}^{k} \sigma_t^2(A) - 2(1 + \sqrt{8\ln(2/\delta)})\sqrt{\frac{k}{c}}\|A\|_F^2$$

and the Frobenius norm result of the first statement of the theorem follows by substituting this bound to Eq. (8), since

$$\|A - A_k\|_F^2 = \|A\|_F^2 - \sum_{t=1}^{k} \sigma_t^2(A).$$

In order to prove the statement of the theorem for the 2-norm of the error, let $\mathcal{H}_k = \text{range}(H) = \textbf{span}(h^{(1)}, h^{(2)}, \ldots, h^{(k)})$. Let $\mathcal{H}_{m-k}$ be the orthogonal complement of $\mathcal{H}_k$ in $\mathcal{R}^m$. Then,

$$\|A - HH^T A\|_2 = \max_{x \in \mathcal{R}^m, \|x\|=1} \|x^T(A - HH^T A)\|.$$

But, x can be expressed as $a_1 \cdot y + a_2 \cdot z$, such that $y \in \mathcal{H}_k$, $z \in \mathcal{H}_{m-k}$, $a_1, a_2 \in \mathcal{R}$ and $a_1^2 + a_2^2 = 1$. Thus,

$$\begin{aligned}\max_{x \in \mathcal{R}^m : \|x\|=1} \|x^T(A - HH^T A)\| \leq& \max_{y \in \mathcal{H}_k : \|y\|=1} \|a_1 y^T(A - HH^T A)\| \\ &+ \max_{z \in \mathcal{H}_{m-k} : \|z\|=1} \|a_2 z^T(A - HH^T A)\|\end{aligned}$$

$$
\begin{aligned}
&\leq \max_{y \in \mathcal{H}_k : \|y\|=1} \|y^T(A - HH^T A)\| \\
&\quad + \max_{z \in \mathcal{H}_{m-k} : \|z\|=1} \|z^T(A - HH^T A)\|.
\end{aligned}
$$

But, for any $y \in \mathcal{H}_k$, $y^T HH^T$ is equal to y. Thus, $\|y^T(A - HH^T A)\| = \|y^T A - y^T A\| = 0$ for all y. Similarly, for any $z \in \mathcal{H}_{m-k}$, $z^T HH^T$ is equal to 0. Thus, we are only seeking a bound for $\max_{z \in \mathcal{H}_{m-k} : \|z\|=1} \|z^T A\|$. To that effect,

$$
\begin{aligned}
\|z^T A\|^2 &= z^T A A^T z = z^T(AA^T - CC^T)z + z^T CC^T z \\
&\leq z^T(AA^T - CC^T)z + \|z^T C\| \\
&\leq \|AA^T - CC^T\|_F + \sigma_{k+1}^2(C).
\end{aligned}
$$

The maximum $\|z^T C\|$ over all unit length $z \in \mathcal{H}_{m-k}$ appears when z is equal to the $(k+1)$ st left singular vector of C. Thus,

$$
\|A - HH^T A\|_2^2 \leq \sigma_{k+1}^2(C) + \|AA^T - CC^T\|_F.
$$

Now, AA^T and CC^T are symmetric matrices and a result of perturbation theory (see e.g. Golub & Van Loan, 1989) states that

$$
|\sigma_{k+1}(AA^T) - \sigma_{k+1}(CC^T)| \leq \|AA^T - CC^T\|_2.
$$

But, using Lemma 5 and the fact that $\|X\|_2 \leq \|X\|_F$ for any matrix X,

$$
\|AA^T - CC^T\|_2 \leq \frac{1 + \sqrt{8\ln(2/\delta)}}{\sqrt{c}} \|A\|_F^2 \tag{12}
$$

holds with probability at least $1 - \delta$. Thus,

$$
|\sigma_{k+1}(AA^T) - \sigma_{k+1}(CC^T)| = \left|\sigma_{k+1}^2(A) - \sigma_{k+1}^2(C)\right| \leq \frac{1 + \sqrt{8\ln(2/\delta)}}{\sqrt{c}} \|A\|_F^2
$$

and the statement of the theorem for the 2-norm of the error follows. □

We conclude with a few words on Theorem 3: the bounds are useful when $\|A - A_k\|_F \ll \|A\|_F$, which is typically the case when spectral techniques are most useful. Also, notice that the error of our approximation is inversely proportional to the number of columns that we include in our sample.

5.3. Doing better than worst-case

Note that even though we prove that picking $c = O(k/\epsilon^2)$ columns of A does the job, it is possible that in an actual problem, the situation may be far from worst-case. In practice, it suffices to pick c rows, where c is at first much smaller than the worst-case bound. Then, we may check whether the resulting approximation $HH^T A$ to A is sufficiently close to A. We can do this in a randomized fashion, namely, sample the entries of $A - HH^T A$ to estimate the sum of squares of this matrix. If this error is not satisfactory, then we may increase c. The details of variance estimates on this procedure are routine.

5.4. Approximating the right singular vectors

We could modify the algorithm to pick rows of A instead of columns and compute approximations to the *right* singular vectors. The bounds in Theorem 3 remain essentially the same (columns become rows and n becomes m). P is now equal to $AH'H'^T$, where H' is an $n \times k$ matrix containing our approximations to the top k *right* singular vectors. The running time of the algorithm is $O(r^2 n + r^3)$, where r is the number of rows that we include in our sample.

6. Recent related work and conclusions

In Ostrovsky and Rabani (2002), the authors presented a polynomial time approximation scheme for the Discrete Clustering Problem, with an exponential dependence on k and the error parameter. We should also note that a closely related problem to DCP is the k-median problem, where again we seek to partition the given points into k clusters; we also seek for each cluster a cluster center or median. The optimal solution minimizes the *sum of distances* of each point to the median of its cluster. For the k-median problem, good approximation algorithms have been hard to come by; most notably (Charikar et al., 2002) gave a constant factor approximation algorithm based on a rounding procedure for a natural linear programming relaxation. The constant has been improved in Jain and Vazirani (1999) and further in Charikar and Guha (1999) using the primal-dual method.

In Achlioptas and McSherry (2001, 2003), Achlioptas and McSherry describe an alternative randomized algorithm for Singular Value Decomposition: given an $m \times n$ matrix A, sample elements of A; create an $m \times n$ matrix $\tilde{A}$ by only keeping the elements of A that are included in the sample, after dividing them by the probability of being sampled. The remaining elements of $\tilde{A}$ are zeroed out; essentially, $\tilde{A}$ is a sparse "sketch" of A. Extending an elegant result of Furedi and Komlos (1981), they prove that $A - \tilde{A}$ is small with respect to the 2-norm. Thus, they argue that the singular vectors and singular values of $\tilde{A}$ closely approximate the corresponding singular vectors and singular values of A. We note that $\tilde{A}$ is an $m \times n$ matrix, thus in order to compute its SVD efficiently one has to employ the Lanczos/Arnoldi techniques. Their error bound with respect to the 2-norm is better than ours (their asymptotic dependency on $1/\epsilon$ is smaller); in Achlioptas and McSherry (2003) they prove that the Frobenius norm bounds are the same for the two algorithms. For a detailed analysis and comparison, we refer the reader to Achlioptas and McSherry (2003).

More recently, Bar-Yossef (2002, 2003) addressed the question of the optimality of our algorithms. In Bar-Yossef (2003), he proves that our algorithm is optimal, with respect to the Frobenius norm bound, up to polynomial factors of $1/\epsilon$. More interestingly, he also proves that if columns of the original matrix are picked uniformly at random (and not with our judiciously chosen sampling probabilities), the error bound of our algorithms *cannot* be achieved.

Perhaps the most interesting open question would be to improve the error bounds of our algorithms by allowing *extra passes* through the input matrices. For example, after computing some initial approximations to the left singular vectors using only a small sample of the columns of A, it would be interesting to design an algorithm that iteratively improves this approximation by accessing A (or parts of A) again. This situation *is not* covered by the lower bounds in Bar-Yossef (2003).

Appendix

Proof of Lemma 5: The matrix C is defined as in Section 5: C contains columns of A after scaling. Thus, CC^T is the sum of c independent random variables and

$$CC^T = \sum_{t=1}^{c} \frac{A^{(i_t)}(A^T)_{(i_t)}}{cp_{i_t}}.$$

We assume that $i_1, \ldots, i_c$ are picked by independent identical trials; in each trial an element from $\{1, 2, \ldots n\}$ is picked according to the probabilities $p_k = \|A^{(k)}\|^2/\|A\|_F^2$, $k = 1, \ldots, n$. Consider the function

$$F(i_1, i_2, \ldots, i_c) = \|AA^T - CC^T\|_F,$$

where $i_1, \ldots i_c$ are independent random variables. We will compute the expectation of F, show that F satisfies a Lipschitz condition and then apply a Martingale inequality to the Doob Martingale associated with F.

Following the lines of Frieze, Kannan, and Vempala (1998) and Drineas and Kannan (2001), we seek to bound

$$\mathbf{E}\left(\sum_{i,j=1}^{m,m} ((AA^T)_{ij} - (CC^T)_{ij})^2\right).$$

Fix attention on one particular i, j. For $t = 1 \ldots c$ define the random variable

$$w_t = \left(\frac{A^{(i_t)}(A^T)_{(i_t)}}{cp_{i_t}}\right)_{ij} = \frac{A_{ii_t}A^T_{i_t j}}{cp_{i_t}}.$$

So, the w_t's are independent random variables. Also, $(CC^T)_{ij} = \sum_{t=1}^{c} w_t$. Thus, its expectation is equal to the sum of the expectations of the w_t's. But,

$$\mathbf{E}(w_t) = \sum_{k=1}^{n} \frac{A_{ik} A_{kj}^T}{c p_k} p_k = \frac{1}{c}(AA^T)_{ij}.$$

So, $\mathbf{E}((CC^T)_{ij}) = \sum_{t=1}^{c} \mathbf{E}(w_t) = (AA^T)_{ij}$. Since $(CC^T)_{ij}$ is the sum of c independent random variables, the variance of $(CC^T)_{ij}$ is the sum of the variances of these variables. But, using $\mathbf{Var}(w_t) = \mathbf{E}(w_t^2) - \mathbf{E}(w_t)^2$, we see that

$$\mathbf{Var}(w_t) = \sum_{k=1}^{n} \frac{A_{ik}^2 (A^T)_{kj}^2}{c^2 p_k} - \frac{1}{c^2}(AA^T)_{ij}^2 \le \sum_{k=1}^{n} \frac{A_{ik}^2 (A^T)_{kj}^2}{c^2 p_k}.$$

Thus,

$$\mathbf{Var}(CC^T)_{ij} \le c \sum_{k=1}^{n} \frac{A_{ik}^2 (A^T)_{kj}^2}{c^2 p_k}.$$

Using $\mathbf{E}((AA^T - CC^T)_{ij}) = 0$ and substituting the values for p_k,

$$\begin{aligned}
\mathbf{E}\big(\|AA^T - CC^T\|_F^2\big) &= \sum_{i=1,j=1}^{m,m} \mathbf{E}\big((AA^T - CC^T)_{ij}^2\big) \\
&= \sum_{i=1,j=1}^{m,m} \mathbf{Var}((CC^T)_{ij}) \\
&\le \frac{1}{c} \sum_{k=1}^{n} \frac{1}{p_k} \left(\sum_i A_{ik}^2 \right) \left(\sum_j (A^T)_{kj}^2 \right) \\
&= \frac{1}{c} \sum_{k=1}^{n} \frac{1}{p_k} \left\| A^{(k)} \right\|^2 \left\| A^{(k)} \right\|^2 \\
&\le \frac{\|A\|_F^2}{c} \sum_{k=1}^{n} \left\| A^{(k)} \right\| \Bigg)^2 \\
&= \frac{1}{c} \|A\|_F^4.
\end{aligned}$$

Using the fact that for any random variable X, $\mathbf{E}(|X|) \le \sqrt{\mathbf{E}(X^2)}$, we get that

$$\mathbf{E}(\|AA^T - CC^T\|_F) \le \frac{1}{\sqrt{c}} \|A\|_F^2. \tag{13}$$

We now present a Lipschitz constraint for F. Consider changing only one of the i_t to i_t' (keeping the other $c-1$ i_t's the same). Let C' be the new matrix so obtained. Then,

$$\|CC^T - C'C'^T\|_F \leq \frac{1}{c}\left(\frac{\|A^{(i_t)}\|\,\|(A^T)_{(i_t)}\|}{p_{i_t}} + \frac{\|A^{(i_t')}\|\,\|(A^T)_{(i_t')}\|}{p_{i_t'}}\right) \leq \frac{2}{c}\|A\|_F^2.$$

Using the triangle inequality,

$$\begin{aligned}\|CC^T - AA^T\|_F &\leq \|C'C'^T - AA^T\|_F + \|CC^T - C'C'^T\|_F\\ &\leq \|C'C'^T - AA^T\|_F + \frac{2}{c}\|A\|_F^2.\end{aligned}$$

Similarly, we get

$$\|C'C'^T - AA^T\|_F \leq \|CC^T - AA^T\|_F + \frac{2}{c}\|A\|_F^2.$$

Thus, changing one of the i_t does not change F by more than $(2/c)\|A\|_F^2$. Now, using Azuma's inequality (see e.g. McDiarmid, 1989)

$$\begin{aligned}&\mathbf{Pr}\left[|F(i_1 \dots i_c) - \mathbf{E}(F(i_1 \dots i_c))| \leq \lambda\sqrt{c}\frac{2}{\beta c}\|A\|_F^2\right]\\ &= \mathbf{Pr}\left[|\|AA^T - CC^T\|_F - \mathbf{E}(\|AA^T - CC^T\|_F)| \leq \lambda\sqrt{c}\frac{2}{\beta c}\|A\|_F^2\right] \geq 1 - 2e^{-\lambda^2/2}.\end{aligned}$$

Thus, using Eq. (13), for all $\delta > 0$, with probability at least $1-\delta$,

$$\|AA^T - CC^T\|_F \leq \frac{1+\sqrt{8\ln(2/\delta)}}{\sqrt{c}}\|A\|_F^2.$$

Proof of Lemma 6: In proving the above lemma we showed that

$$\mathbf{E}\big(\|AA^T - CC^T\|_F^2\big) = \frac{1}{c}\sum_{k=1}^{n}\frac{1}{p_k}\|A^{(k)}\|^4 - \frac{1}{c}\|AA^T\|_F^2.$$

To prove that our choice of the p_k's minimizes the above expectation among all possible choices of the p_k's, we define the function (observe that $(1/c)\|AA^T\|_F^2$ is independent of the p_k's)

$$f(p_1, \dots p_n) = \frac{1}{c}\sum_{k=1}^{n}\frac{1}{p_k}\|A^{(k)}\|^4.$$

We want to minimize f given that $\sum_{k=1}^{n} p_k = 1$. Using simple calculus (that is substituting $p_n = 1 - \sum_{k=1}^{n-1} p_k$ and solving the system of equations $\frac{\partial f}{\partial p_k} = 0, k = 1, \ldots, n-1$), we get that $p_k = \|A^{(k)}\|^2 / \|A\|_F^2$.

Acknowledgments

We wish to thank Nina Mishra and 3 anonymous reviewers for the careful reading of our paper and for many comments that significantly improved the presentation of our work.

Notes

[1] A simple modification of our algorithm may be used to approximate right singular vectors as well.

[2] Again, the algorithm can be modified to approximate *right* singular vectors.

References

Achlioptas, D., & McSherry, F. (2001). Fast computation of low rank approximations. In *Proceedings of the 33rd Annual Symposium on Theory of Computing* (pp. 337–346).

Achlioptas, D., & McSherry, F. (2003). Fast computation of low rank matrix approximations. *Manuscript*.

Andrews, H. C., & Patterson, C. L. (1976a). Singular value decomposition image coding. *IEEE Trans. on Communications, 4*, 425–432.

Andrews, H. C., & Patterson, C. L. (1976b). Singular value decompositions and digital image processing. *IEEE Trans. ASSP,* 26–53.

Azar, Y., Fiat, A., Karlin, A., McSherry, F., & Saia, J. (2001). Spectral analysis of data. In *Proc. of the 33rd ACM Symposium on Theory of Computing* (pp. 619–626).

Berry, M. J., & Linoff, G. (1997). *Data mining techniques*. John-Wiley.

Bar-Yossef, Z. (2002). The complexity of massive dataset computations. Ph.D. thesis, University of California, Berkeley.

Bar-Yossef, Z. (2003). Sampling lower bounds via information theory. In *Proceedings of the 35th Annual Symposium on Theory of Computing* (pp. 335–344).

Charikar, M., & Guha, S. (1999). Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science* (pp. 378–388).

Charikar, M., Guha, S., Shmoys, D., & Tardos, E. (2002). A constant factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences, 65:1*, 129–149.

Drineas, P., & Kannan, R. (2001). Fast Monte-Carlo algorithms for approximate matrix multiplication. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (pp. 452–459).

Frieze, A., Kannan, R., & Vempala, S. (1998). Fast Monte-Carlo algorithms for finding low rank approximations. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science* (pp. 370–378).

Furedi, Z., & Komlos, J. (1981). The eigenvalues of random symmetric matrices. *Combinatorica, 1*, 233–241.

Gersho, A., & Gray, R. M. (1991). *Vector quantization and signal compression*. Kluwer Academic.

Gibson, D., Kleinberg, J., & Raghavan, P. (1998). Clustering categorical data: An approach based on dynamical systems. *Very Large Data Bases (VLDB),* 311–322.

Goldreich, O., Goldwasser, S., & Ron, D. (1998). Property testing and its connection to learning and approximation. *Journal of the ACM, 5:4*, 653–750.

Golub, G., & Van Loan, C. (1989). *Matrix computations*. Johns Hopkins University Press.

Huang, T., & Narendra, P. (1974). Image restoration by singular value decomposition. *Applied Optics, 14:9*, 2213–2216.

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice Hall.

Jain, K., & Vazirani, V. (1999). Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science* (pp. 2–13).

Jambu, M., & Lebeaux, M.-O. (1983). *Cluster analysis and data analysis*. North Holland.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2000). The analysis of a simple k-means clustering algorithm. In *Symposium on Computational Geometry* (pp. 100–109).

Kleinberg, J. (1998). Authoritative sources in a hyperlinked environment. In *Proceedings of 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 668–677).

McDiarmid, C. J. H. (1989). On the method of bounded differences. In *Surveys in Combinatorics: Invited Papers at the 12th British Combinatorial Conference* (pp. 148–188).

Ostrovsky, R., & Rabani, Y. (2002). Polynomial time approximation schemes for geometric k-clustering. *Journal of the ACM, 49:2*, 139–156.

Papadimitriou, C. H., Raghavan, P., Tamaki, H., & Vempala, S. (2000). Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences, 61:2*, 217–235.

Parlett, B. (1997). *The symmetric eigenvalue problem*. Classics in Applied Mathematics, SIAM.

Received December 27, 2002
Revised February 10, 2004
Accepted February 11, 2004
Final manuscript March 4, 2004