15-859 Algorithms for Big Data — Spring 2025

Problem Set 3

Due: Thursday, March 20, before class

Please see the following link for collaboration and other homework policies:
http://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/teaching/15851-spring25/grading.pdf

Problem 1: Geometric Mean Estimator for ℓ_1-Estimation (20 points)

In class we studied a sketch for ℓ_1-estimation based on Cauchy random variables. Recall that if S is a matrix of i.i.d. Cauchy random variables with $k = O(\epsilon^{-2})$ rows and n columns, then for any fixed vector $x \in \mathbb{R}^n$, if E is the median of entries in the list $(|(Sx)_1|, |(Sx)_2|, \ldots, |(Sx)_k|)$, then $E \in (1 \pm \epsilon)\|x\|_1$ with probability at least 9/10, for appropriate choice of constant in the big-Oh notation defining k.

In this problem we will consider another estimator F called the *geometric mean estimator*, which is based on the same sketch Sx (for a fixed $x \in \mathbb{R}^n$) with $k = O(\epsilon^{-2})$ rows. For this estimator, we partition the rows of Sx into $k/3$ groups $G_1, \ldots, G_{k/3}$, each of size 3, where we assume k is a multiple of 3. In the i-th group, if $(Sx)_a, (Sx)_b, (Sx)_c$ are the three coordinates in that group, then we set

$$F_i = |(Sx)_a \cdot (Sx)_b \cdot (Sx)_c|^{1/3}.$$

1. (10 points) Show that the expectation of F_i exists and is equal to $C \cdot \|x\|_1$, where $C > 0$ is a scalar that does not depend on x.

   HINT: You will need to use the density function of a Cauchy random variable from lecture.

2. (5 points) Show that the variance of F_i is bounded by $O(\|x\|_1^2)$.

   HINT: You can again use the density function of a Cauchy random variable from lecture.

3. (5 points) We define our overall estimator F as follows:

$$F = \frac{3}{Ck} \sum_{i=1}^{k/3} F_i.$$

   Conclude that for appropriate $k = O(\epsilon^{-2})$, that with probability at least 9/10, we have $F \in (1 \pm \epsilon)\|x\|_1$. Recall that this is for any fixed vector $x \in \mathbb{R}^n$.

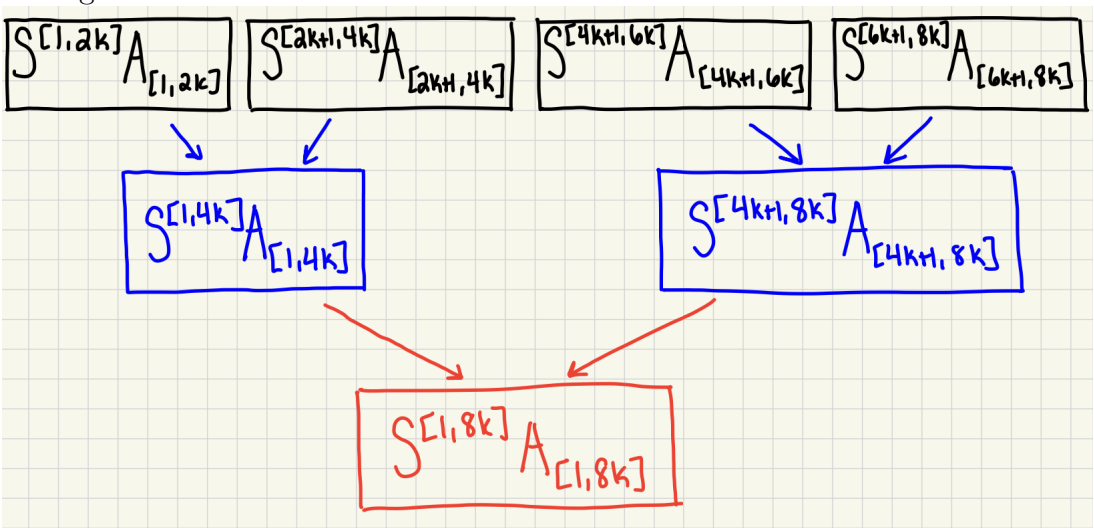
Problem 2: Online Leverage Score Sampling plus Merge and Reduce (30 points)

We consider the data stream model, where we see each row of an $n \times d$ matrix A one at a time. We assume the entries of A are each $O(\log n)$ bit integers and that $n > d$.

A coreset for subspace approximation, given an $n \times d$ matrix A with $n > d$, is a weighted subset of rows of A, denoted $S \cdot A$, so that $\|SAx\|_2 = (1 \pm \epsilon)\|Ax\|_2$ simultaneously for all x. In this problem, we will construct a coreset for A in the data stream model. We will introduce the merge-and-reduce framework, and then improve it using online leverage score sampling.

It turns out that deterministic coresets S exist with only $k = C \cdot d/\epsilon^2$ rows of S, for a certain constant $C > 0$, and they can be constructed in $O(kd(\log n))$ bits of space. These are due to Batson, Spielman, and Srivastava, and they are called BSS-sparsifiers. We will assume the existence of such coresets in what follows.

Consider the following merge-and-reduce algorithm. We store the first $2k$ rows in the stream, denoted $A_{[1,2k]}$. We then compute a BSS-sparsifier $S^{[1,2k]}A_{[1,2k]}$ to reduce the number of rows to k. We superscript $S^{[1,2k]}$ to keep track of which rows the BSS-sparsifier has been applied to. We then store the next $2k$ rows $A_{[2k+1,4k]}$ in the stream. We then compute a BSS-sparsifier $S^{[2k+1,4k]}A_{[2k+1,4k]}$ in the stream. We then *merge* $S^{[1,2k]}A_{[1,2k]}$ and $S^{[2k+1,4k]}A_{[2k+1,4k]}$ by computing a BSS-sparsifier of their concatenation. We will call this $S^{[1,4k]}A_{[1,4k]}$. We then store the next $2k$ rows $A_{[4k+1,6k]}$ and compute a BSS-sparsifier $S^{[4k+1,6k]}A^{[4k+1,6k]}$. We then store the next $2k$ rows $A_{[6k+1,8k]}$ and compute a BSS-sparsifier $S^{[6k+1,8k]}A^{[6k+1,8k]}$. We then merge $S^{[4k+1,6k]}A^{[4k+1,6k]}$ and $S^{[6k+1,8k]}A^{[6k+1,8k]}$, obtaining $S^{[4k+1,8k]}A^{[4k+1,8k]}$. Now we additionally merge $S^{[1,4k]}A_{[1,4k]}$ and $S^{[4k+1,8k]}A^{[4k+1,8k]}$, obtaining $S^{[1,8k]}A_{[1,8k]}$, and so on. In this way we build a binary tree of height $\Theta(\log(n/2k))$ of coresets, and at any point during the stream we only store one coreset at each depth. This algorithm is called the merge-and-reduce algorithm.



1. (10 points) Although BSS-sparsifiers are deterministic, and thus will always be correct, when one merges coresets the error in the multiplicative approximation grows. Suppose

that we use the same value for the approximation factor for all coresets, and therefore the value of k for all coresets at all nodes in the binary tree is the same. What value of k should we choose to guarantee that the final coreset at the root of the binary tree is a $(1 \pm \epsilon)$-approximation, and what is the overall memory required of the streaming algorithm for that value of k? Asymptotic notation is fine. You can assume that $k < n^{0.9}$ if it makes your calculations easier.

2. (20 points) In order to improve the memory required of the above scheme, we will combine it with online leverage score sampling. The i-th online leverage score ℓ_i is defined to be

$$\ell_i = \min(a_i^T (A_{i-1}^T A_{i-1} + \lambda I)^{-1} a_i, 1),$$

where A_{i-1} for $i > 1$ denotes the submatrix of A consisting of its first $i-1$ rows, A_0 is the 0 matrix, and $\lambda > 0$ is a parameter. It can be shown that

$$\sum_{i=1}^{n} \ell_i = O(d \log(1 + \|A\|_2^2/\lambda)),$$

which you can take as given[1].

   (a) (3 points) Argue that one can maintain $A_{i-1}^T A_{i-1}$ in a stream exactly and deterministically using only $O(d^2 \log n)$ bits of memory, and thus one can compute ℓ_i deterministically and exactly given the i-th row in the stream, for each $i \in \{1, 2, \ldots, n\}$.

   For the next part, we will need the following fact, which is based on a matrix concentration inequality similar to the matrix Chernoff bound we did in class. It is different than the result in class though since rows are sampled independently and without replacement. You can use this fact as given:

   (**Sampling Without Replacement**) Given an error parameter $0 < \gamma < 1$, let u be a vector of leverage score overestimates, i.e., $\tau_i(A) \leq u_i$ for all i, where $\tau_i(A)$ is the i-th leverage score of A. For each row we define a sampling probability $p_i = \min(1, \gamma^{-2} u_i c \log d)$, where $c > 0$ is a positive constant. Let Sample(u) return a random diagonal matrix T with independently chosen entries: we have $T_{i,i} = 1/\sqrt{p_i}$ with probability p_i and $T_{i,i} = 0$ otherwise. Then with probability at least 9/10, we have

   - T has at most $c\gamma^{-2}(\log d)\|u\|_1$ non-zero entries, and
   - Simultaneously for all x, $\|TAx\|_2 = (1 \pm \gamma)\|Ax\|_2$.

   (b) (5 points) Argue that if one samples each row i of A independently with probability equal to $p_i = \min(\Theta(\epsilon^{-2} \ell_i \log d), 1)$, and rescales a sampled row by $\frac{1}{\sqrt{p_i}}$, obtaining a matrix TA of sampled and rescaled rows of A, then with probability at least 9/10, one has $\|TAx\|_2^2 = (1 \pm \epsilon/3)(\|Ax\|_2^2 \pm \lambda \|x\|_2^2)$, simultaneously for all x.

[1] If you would like to see a proof, see Problem 1 in the 2022 15-859 Homework 2.

HINT: Apply the Sampling Without Replacement Bound. It will help to show that the online leverage score upper bounds the actual leverage score of the matrix $[A; \sqrt{\lambda I}]$. You can also use the fact that for positive semidefinite matrices C and D, if $D \succeq C$ then $C^{-1} \succeq D^{-1}$ in the positive semidefinite ordering discussed in class.

(c) (4 points) Argue that by setting $\lambda = \epsilon \cdot \sigma^2_{\min}(A)$, where $\sigma_{\min}(A)$ is the smallest singular value of A, with probability at least $9/10$, we have that $\|TAx\|_2^2 = (1 \pm O(\epsilon))\|Ax\|_2^2$ simultaneously for all x.

(d) (8 points) Suppose one samples rows of A by their online leverage score, and feeds only the sampled (and rescaled) rows TA into the merge-and-reduce algorithm above. Note that the total stream length that merge-and-reduce is being applied to now is equal to the number of non-zero entries of T. Show how to modify the parameters of the merge-and-reduce algorithm above to obtain an algorithm using $d^2 \cdot \text{poly}(\log\log(\kappa d/\epsilon))(\log n)/\epsilon^2$ bits of memory so that, with probability at least $9/10$, the final coreset output at the root of the tree, denoted STA, satisfies $\|STAx\|_2 = (1 \pm \epsilon)\|Ax\|_2$ simultaneously for all x. Here $\kappa = \|A\|_2^2/\sigma^2_{\min}(A)$ is the condition number of A.