CS 226 / 6.889 Sketching Algorithms for Big Data — Fall 2017

Problem Set 1

Due: 11:59pm, Monday, October 2nd
Submit to: `sketchingbigdata-f17-assignments@seas.harvard.edu`

See homework policy at `http://www.sketchingbigdata.org/fall17/syllabus/`

Preamble: As you probably noticed, this pset is long. It does not mean it is difficult. The length is mostly due to the fact that the problems introduce additional material and concepts that (we believe) are interesting and shed new light on the material seen in the lecture. If you have any questions about the new material, please contact the TA or the professors.

Problem 1: More on incoherent matrices. Recall from the JL lower bound lecture that an ε-incoherent matrix $\Pi \in \mathbb{R}^{m\times n}$ is such that each column Π^i of Π has unit ℓ_2 norm (i.e. $\sum_{j=1}^m (\Pi^i)_j^2 = 1$), and the dot products $\langle \Pi^i, \Pi^j\rangle$ are all at most ε in magnitude for $i \neq j$.

(a) (3 points) In Lecture 5 we stated that an $n\times n$ real symmetric "ε-approximate identity" matrix A (with 1's on the diagonal and $\pm\varepsilon$ on the off-diagonal entries) has rank $\Omega(n)$ if $\varepsilon < 1/\sqrt{n}$. Let $A^{\oplus k} \in \mathbb{R}^{n\times n}$ be defined such that $(A^{\oplus k})_{i,j} = (A_{i,j})^k$. Show that if $r := rank(A)$, then $rank(A^{\oplus k}) \le \binom{r+k}{k}$. Deduce a dimension lower bound of $m = \Omega(\min\{n, \varepsilon^{-2}\log n/\log(1/\varepsilon)\})$ for $(1+\varepsilon)$-dimensionality reduction on the simplex.

(b) (2 points) In class we observed that any $(1+\varepsilon)$-distortion embedding of the simplex $X = \{0, e_1, \ldots, e_n\} \subset \mathbb{R}^n$ gives an $O(\varepsilon)$-incoherent matrix. Conversely, any ε-incoherent matrix Π gives a $(1+\varepsilon)$-distortion embedding of the simplex (map 0 to 0, and e_i to Π^i). Noga Alon's lower bound showed that any ε-incoherent matrix with m rows needs $m \gtrsim \min\{n, \varepsilon^{-2}\lg n/\lg(1/\varepsilon)\}$. This construction gives a hard point set of size $n+1$ in dimension n for any n (or said differently, a hard point set of size n in dimension $d = n-1$). Show how to use Alon's approach to show that for any $d < n \le 2^{O(\varepsilon^2 d)}$, there is a hard point set such that any $(1+\varepsilon)$-distortion embedding into m-dimensional Euclidean space requires $m \gtrsim \min\{n, \varepsilon^{-2}\lg n/\lg(1/\varepsilon)\}$.

One way to construct an incoherent matrix is from a *code*. In particular, if $\mathcal{C} = \{C_1, \ldots, C_N\}$ is a collection of N vectors each in $[q]^t$ for some positive integers q, t (where $[q]$ denotes $\{1, \ldots, q\}$), then define $\alpha \in (0,1)$ as the maximum fraction of the t coordinates for which any distinct C_i, C_j agree. Given such a code, one can construct an ε-incoherent matrix with $m = qt$, $n = N$, and $\varepsilon = \alpha$ as in Figure 1. Note the dot product between two different codewords is exactly the fraction of coordinates on which they agree.

We would like to show how to construct a code that gives small $m = qt$.

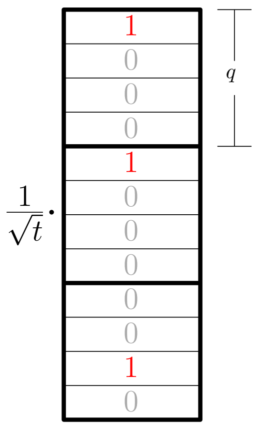

Figure 1: Each codeword gives one column of the incoherent matrix. Here $q = 4, t = 3$ and the codeword is $C_i = (1, 1, 3)$. The vector is $m = qt$ dimensional with the coordinates broken up into t blocks each of size q. A 1 is placed in the jth position in the location specified by $(C_i)_j$. The entire vector is normalized by $1/\sqrt{t}$ to have unit norm.

(c) (3 points) Show that for any N and $0 < \alpha < 1/2$, such codes exist with $q = O(\alpha^{-1}), t = O(\alpha^{-1} \log N)$ by picking random C_i (hint: use the Chernoff bound). Note this gives an incoherent matrix with N columns and $m = qt = O(\varepsilon^{-2} \log N)$ rows, matching JL.

It turns out one can do better when ε is sufficiently small. Consider the finite field $\mathbb{F}_q$ and consider all polynomials $p_1, \ldots, p_N \in \mathbb{F}_q[x]$ of degree at most d where $N = q^{d+1}$. Define the *Reed-Solomon code* $C_1, \ldots, C_N$ as follows: $t = q$ where the jth entry of C_i is the evaluation of p_i on the jth element of $\mathbb{F}_q$ (so C_i is the evaluation table of p_i).

(d) (6 points) Recall we need $N \geq n$ to ensure there are enough C_i's to form our matrix Π. Show how to choose d, q so $N \geq n$ and $\alpha \leq \varepsilon$, and show what this gives (in big-Oh notation) for $m = qt$ being the number of rows over the incoherent matrix Π we obtain.

(e) (4 points) How small does ε need to be as a function of n for the codes from part (d) to give smaller m than the random codes from part (c)? If the turning point is ε_T, you should provide an answer ε'_T such that $\log(1/\varepsilon'_T) = \Theta(\log(1/\varepsilon_T))$.

(f) (5 points) Suppose one has an ε-incoherent matrix Π. Show how ℓ_1-point queries to x being updated in the turnstile streaming model can be answered solely given $y = \Pi x$. (Note then that part (d) implies a *deterministic* ℓ_1-point query algorithm with low space in turnstile streams.) **Hint:** it may help to remember that $\Pi x = \sum_i x_i \Pi^i$.

OPEN PROBLEM: As mentioned above, an incoherent matrix with $m = O(\varepsilon^{-2} \log n)$ exists. The solution to this problem provides another, incomparable bound. The lower bound is $m = \Omega(\min\{n, \varepsilon^{-2}(\log n)/\log(1/\varepsilon)\}$ [1, Section 9]. Can the gap between upper and lower bounds be closed? It is conceivable a better upper bound could be achieved by discovering a better code construction.

Problem 2: Counting distinct elements with deletions. In Lecture 2 we showed how to estimate the number of distinct elements in a stream in $poly(\varepsilon^{-1}\lg n)$ bits of space with $2/3$ success probability, where all integers in the stream are in $[n]$. As mentioned in Lecture 3, this problem can be re-cast in the following way: there is a vector $x \in \mathbb{R}^n$ that starts as 0, and each integer i in the stream causes the change $x_i \leftarrow x_i + 1$. We would then like to estimate $\|x\|_0 := |\{i : x_i \neq 0\}|$. What if the updates in the stream are of the form (i, Δ) for $\Delta \in \{-1, 1\}$, and now such an update causes the change $x_i \leftarrow x_i + \Delta$?

(a) (2 points) What about our non-idealized algorithm from Lecture 2 breaks down now that negative updates are also allowed?

(b) (8 points) Show how to alter the non-idealized algorithm to obtain space $poly(\varepsilon^{-1}\lg(nm))$ for this modified problem, where m is the length of the stream. **Hint:** consider using techniques from Lecture 7 to fix the issue in part (a).

Problem 3: Reducing randomness via Nisan's Generator. In Lecture 4 we have seen a collection of algorithms for estimating the ℓ_p norm of the n-dimensional vector x induced by the stream, for $p \in (0, 2]$. The idea was to calculate a "linear sketch" $\Pi x = [Z_1 \ldots Z_k]$, where R was an $k \times n$ random matrix, with i.i.d. entries r_{ij} selected from a p-stable distribution. After calculating Πx, the algorithm outputs

$$\text{median}[|Z_1|, \cdots, |Z_k|]/C(p)$$

as an estimator of $\|x\|_p$, where $C(p)$ denotes some scaling factor that depends only on p. To make life simple, in this problem we will focus on the decision version of the algorithm, which checks whether

$$\text{median}[|Z_1|, \cdots, |Z_k|]/C(p) \quad \geq \quad T \tag{1}$$

for some threshold T. For concreteness, in what follows we focus on $p = 2$, in which case the entries of Π can be selected from Gaussian distribution $\mathcal{N}(0, 1)$. We also assume that the entries of x always remain integers from $\{-M \ldots M\}$ for some $M = n^{O(1)}$, i.e., they have values polynomial in the dimension n.

The analysis in the lecture skipped over the following two issues:

- Discretization: given that the algorithm space is measured in bits, we need to make sure that each r_{ij} has bounded precision. Dealing with this issue is straightforward, as we can modify the random variables so that their values fall into an interval $[-C\sqrt{\log n} \ldots C\sqrt{\log n}]$, and are multiples of of $1/n^C$, for some $C = O(1)$. The analysis of the modified algorithm remains essentially unchanged, modulo minor increase in the approximation error and failure probability. In what follows we assume that r_{ij} are already generated in this way, and therefore need only $b = O(\log n)$ bits of representation.

- Pseudo-randomness: even if $r'_{ij}s$ are discrete, we cannot afford to store all of them in memory, as this would require knb bits of storage. Instead, it was mentioned that they can be generated "on the fly" using a pseudorandom generator, i.e., there is an efficiently computable mapping $G : \{0,1\}^L \to \{\{0,1\}^b\}^{nk}$ such that we can set $r_{ij} = G(v)_{ij}$, where v is a "random seed" selected from $\{0,1\}^L$ uniformly at random. Formalizing and optimizing this step is the focus of this problem.

We will use the pseudo-random generator for bounded space due to Nisan [2]. Consider a class of (S,b)-*automata* Q, that have 2^S states and read sequences of symbols from $\{0,1\}^b$, i.e., operate over an alphabet of size 2^b. Such automata are defined by:

- a transition function $Q(s,a)$, which descibe the state the automaton moves to from state s after reading a,
- an initial state $start$, and
- a set of accepting states Acc.

Such automata can model any deterministic computation device that processes a sequence u of symbols from $\{0,1\}^b$ in space S. We use $Q(u)$ to denote the state reached by the automaton after reading u, starting from $start$.

Nisan's generator G has the following wonderful properties. Suppose that the automaton is applied to sequences of length R. Then:

- The seed length L of G is equal to $O(S \log R)$, assuming $b = O(S)$.
- It ϵ-*fools* any (S,b)-automaton Q , i.e.,

$$|\underset{u\in(\{0,1\}^b)^R}{\mathbb{P}}[Q(u) \in Acc] - \underset{v\in\{0,1\}^L}{\mathbb{P}}[Q(G(v)) \in Acc]| \le \epsilon$$

  for $\epsilon = 2^{-\Omega(S)}$.

Note that in the above definition, the input to Q consists of (pseudo)-random bits, which are "tested" by Q. Nisan's generator is designed to ϵ-fool all such tests, despite generating randomness from a relatively small truly random seed.

To use Nisan's generator in our streaming algorithm, we need to model the algorithm as a finite automaton reading the random entries of the matrix Π and producing some decision in the end. Then we use the properties of the generator to argue that replacing truly random Π by a pseudorandomly generated version does not (significantly) alter the behavior of the algorithm.

The computation specified in Equation (1) can be performed by an automaton Q that reads the entries $r_{1,1}, \ldots, r_{1,n}, r_{2,1}, \ldots, r_{2,n}, \ldots r_{k,n}$ of Π (i.e., in the row-wise order), computes the vector $[Z_1, \cdots, Z_k]$, evaluates the median and accepts if the result is at least T. (Note that Q is *parameterized* by the vector x, i.e., x is *not* an input to Q!).

Finally, we are ready to state the problems:

(a) (3 points) Observe that Q can be implemented so that $S = O(k \log n)$. Calculate the length of the seed L required to $2^{-\Omega(S)}$-fool such automata Q.

(b) (12 points) Show a better implementation of Q that requires only $S = O(\log k + \log n)$. Calculate the length of the seed L required to $2^{-\Omega(S)}$-fool such automata Q.

Problem 4: (1 point) How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. (sum of time spent solving problem and typing up your solution)

References

[1] Noga Alon. Problems and results in extremal combinatorics–I *Discrete Mathematics*, 273(1-3): 31–53, 2003.

[2] Noam Nisan. Pseudorandom Generators for Space-Bounded Computation. *Combinatorica*, 12(4):449-461, 1992.