Wednesday, February 17, 2016

Problem Set 2

Due: Wednesday, February 24, 2016 – 7 pm
Dropbox Outside Stata G5

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. Write each problem in a separate sheet and write down your name on top of every sheet.
5. **No bibles. This includes solutions posted to problems in previous years.**

1 Rank estimation from a sample

Suppose we have an *unsorted* list of distinct numbers $x_1, x_2, \ldots, x_n$. We want to randomly subsample $t < n$ items from this list (drawing with replacement) and from that subsample we want to return some element x such that $rank(x)$ is approximately equal to k for a given k. By *rank* we mean the rank of x in the original list. The largest of $x_1, \ldots, x_n$ has rank 1, the second largest has rank 2, etc. and we're trying to find something with rank $\approx k$.

Describe a simple strategy for choosing a candidate x from the subsample with rank approximately equal to k. How large do we need to set t (in big-O notation) so that, with probability $(1 - \delta)$, your strategy returns an x with $(1 - \epsilon)k \leq rank(x) \leq (1 + \epsilon)k$?

Hint: This should not require any complex counting arguments or combination/permutation calculations.

2 Distinct elements with deletion

In class we saw how to estimate the number of distinct elements in a data stream using the Flajolet-Martin algorithm, which required $O(\frac{1}{\epsilon^2})$ registers.

Consider the following alternative formulation of the distinct elements problem: given an N dimensional vector $\mathbf{x}$, we want to process a stream of arbitrary increments to entries in $\mathbf{x}$. In other words, if we see a number $i \in 1, \ldots, N$ in the stream, we update entry $x_i \Leftarrow x_i + 1$.

Our goal is to estimate $\|\mathbf{x}\|_0$, which measures the number of non-zero entries in $\mathbf{x}$. With $\mathbf{x}$ viewed as a histogram that maintains counts for N potential elements, $\|\mathbf{x}\|_0$ is exactly the number of distinct elements processed.

In this problem we will develop an alternative algorithm for estimating $\|\mathbf{x}\|_0$ that can also handle *decrements* to entries in $\mathbf{x}$. Specifically, instead of the stream containing just indices i, it contains pairs $(i,+)$ and $(i,-)$. On receiving $(i,+)$ $\mathbf{x}$ should update so that $x_i \Leftarrow x_i + 1$ and on receiving $(i,-)$ $\mathbf{x}$ should update so that $x_i \Leftarrow x_i - 1$. For this problem we will assume that, at the end of our stream, each $x_i \geq 0$ (i.e. for a specific index we can't receive more decrements than increments).

Let's start with a simpler problem. For a given value T, let's design an algorithm that succeeds with probability $(1-\delta)$, outputing LOW if $T < \frac{1}{2}\|\mathbf{x}\|_0$ and HIGH if $T > 2\|\mathbf{x}\|_0$.

(a) As in class, assume we have access to a completely random hash function $h(\cdot)$ that maps each i to a random point in $[0,1]$. We maintain the estimator $s = \sum_{i:h(i)<\frac{1}{2T}} x_i$ as we receive increment and decrement updates. Show that, at the end of our stream:

  (i) If $T < \frac{1}{2}\|x\|_0$, $Pr_h[s = 0] < 1/e \approx .37$

  (ii) If $T > 2\|x\|_0$, $Pr_h[s = 0] > .5$

(b) Using this fact, show how to use $O(\log 1/\delta)$ independent random hash functions, and corresponding individual estimators $s_1, s_2, \ldots, s_{O(\log 1/\delta)}$, to output LOW if $T < \frac{1}{2}\|\mathbf{x}\|_0$ and HIGH if $T > 2\|\mathbf{x}\|_0$. If neither event occurs you can output either LOW or HIGH. Your algorithm should succeed with probability $(1-\delta)$.

(c) Using $O(\log N)$ repetitions of your algorithm for the above decision problem (with δ set appropriately), show how to obtain an estimate F for $\|\mathbf{x}\|_0$ such that $\frac{1}{4}\|\mathbf{x}\|_0 \leq F \leq 4\|\mathbf{x}\|_0$.

The algorithm described is an example of a *linear sketch*, a term that refers to the fact that each estimator s stores exactly the same value at the end of the stream, no matter the order or number of increments and decrements. s only depends on the final value of $\mathbf{x}$ and our random hash function $h(\cdot)$.

3 Better point query?

In class we learned about the Count-Min sketch. We saw that it could be used to estimate the frequency of any item in our stream up to an additive error ϵn, where n is the total number of elements streamed in.

Using the notation from Problem 2, this is equivalent to estimating the value of any entry x_i in $\mathbf{x}$ to within additive error $\epsilon\|\mathbf{x}\|_1$. $\|\mathbf{x}\|_1$ increases by 1 every time a new element is streamed in, which is why $\|\mathbf{x}\|_1 = n$.

In this problem, we'll analyze an alternative algorithm that requires a bit more space, but can estimate the value of x_i to within error $\epsilon\|\mathbf{x}\|_2$, which is often *much better* in practice.

(a) Briefly describe a scenario when an error of $\epsilon\|\mathbf{x}\|_2$ could be much tighter than an error of $\epsilon\|\mathbf{x}\|_1$ for estimating an items frequency.

We'll analyze the following procedure:

- For a small value q to be set later, choose a random hash function $h(\cdot)$ that maps every $i \in \{1, \dots, N\}$ to $\{1, \dots, q\}$[1]. Choose another random hash function $g(\cdot)$ that maps every $i \in \{1, \dots, N\}$ to $\{-1, 1\}$. Allocate space for q counters $C_1, \dots, C_q$.
- When $Increment(x_i)$ is called, set $C_{h(i)} \Leftarrow C_{h(i)} + g(i)$.
- When $Estimate(x_i)$ is called, return $y_i = g(i)C_{h(i)}$.

(b) Show that $\mathbb{E}[y_i] = x_i$. In other words, show that our estimate for x_i is correct in expectation.

(c) Show that $Var[y_i] = \mathbb{E}[y_i^2] - \mathbb{E}[y_i]^2 \leq \frac{\|\mathbf{x}\|_2^2}{q}$. **Warning**: This part is a bit more challenging!

(d) Apply Chebyshev's inequality to show that $\mathbb{P}(|x_i - y_i| \geq \epsilon\|x\|_2) \leq \frac{1}{\epsilon^2 q}$.

(e) What value of q would we need to ensure that we obtain ϵ error with probability 9/10? How many counters do we need to store in comparison to Count-Min?

4 Johnson-Lindenstrauss for k-means clustering

In this problem we analyze a common algorithmic application of the Johnson-Lindenstrauss lemma.

Consider the k-means clustering problem. Given points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$, the goal is partition the points into k disjoint sets (clusters) $C_1, \dots, C_k$ so as to minimize the cost:

$$Cost(\mathbf{x}_1, \dots, \mathbf{x}_n, C_1, \dots, C_k) = \sum_{i=1}^{k} \sum_{j \in C_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|_2^2,$$

where $\boldsymbol{\mu}_i = \frac{1}{|C_i|}\sum_{j \in C_i} \mathbf{x}_j$ is the mean of the points in cluster C_i. In other words, we want to find k clusters that have as little internal variance as possible.

(a) Prove that:

$$Cost(\mathbf{x}_1, \dots, \mathbf{x}_n, C_1, \dots, C_k) = \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{j,\ell \in C_i, j<\ell} \|\mathbf{x}_j - \mathbf{x}_\ell\|_2^2.$$

[1] You can assume full randomess for now, although it is not required.

(b) Suppose we embed $\mathbf{x}_1, \ldots, \mathbf{x}_n$ into $O(\log n/\epsilon^2)$ dimensional vectors $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$ using the Johnson-Lindenstrauss construction from class. Recall that, with high probability, the Johnson-Lindenstrauss lemma ensures that:

$$(1-\epsilon)\|\mathbf{x}_j - \mathbf{x}_k\|_2^2 \leq \|\tilde{\mathbf{x}}_j - \tilde{\mathbf{x}}_k\|_2^2 \leq (1+\epsilon)\|\mathbf{x}_j - \mathbf{x}_k\|_2^2,$$

for all j, k. Leverage this fact and part (a) to show that, for *all clusterings simultaneously*:

$$\begin{aligned} (1-\epsilon)Cost(\mathbf{x}_1, \ldots, \mathbf{x}_n, C_1, \ldots, C_k) \\ \leq Cost(\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n, C_1, \ldots, C_k) \\ \leq (1+\epsilon)Cost(\mathbf{x}_1, \ldots, \mathbf{x}_n, C_1, \ldots, C_k), \end{aligned}$$

with high probability.

(c) Suppose we find a set of clusters $\tilde{C}_1, \ldots, \tilde{C}_k$ such that:

$$Cost(\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n, \tilde{C}_1, \ldots, \tilde{C}_k) \leq \gamma \cdot Cost(\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n, \tilde{C}_1^*, \ldots, \tilde{C}_k^*),$$

where $\tilde{C}_1^*, \ldots, \tilde{C}_k^*$ is the optimal partition for $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$ – i.e. we find a γ approximation to the optimal clustering for the dimensionality reduced points. Show that, for $\epsilon \leq \frac{1}{2}$,

$$Cost(\mathbf{x}_1, \ldots, \mathbf{x}_n, \tilde{C}_1, \ldots, \tilde{C}_k) \leq (1+O(\epsilon))\gamma \cdot Cost(\mathbf{x}_1, \ldots, \mathbf{x}_n, C_1^*, \ldots, C_k^*),$$

where $C_1^*, \ldots, C_k^*$ is the optimal partition for $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

In other words, we can compute an approximate clustering for our (possibly high dimensional) original points using just the dimensionality reduced data. This approach will speed up algorithms for solving k-means whenever $\log n/\epsilon^2 < d$.

(d) (**Open Research Question**) In k-means clustering, k is typically much smaller than n. It can be shown that Johnson-Lindenstrauss embedding to just $O(\log k/\epsilon^2)$ dimensions suffices for obtaining a $(9+\epsilon)\gamma$ approximation. Can you beat the $(9+\epsilon)$ factor approximation loss, or show that it is optimal when reducing to $O(\log k)$ dimensions instead of $O(\log n)$?