COMS E6998-9: Algorithmic Techniques for Massive Data — Sep 17, 2015

Lecture 4 – CountSketch and High Frequencies

Instructor: *Alex Andoni* — Scribes: *Jeffrey Martin*

1 From CountMin to CountSketch

1.1 Recap

During the previous lecture we learned that the 2nd moment is defined as $F_2 = \sum_i f_i^2$, and that it can be probabilistically approximated by the Tug of War algorithm.

We also learned that the "heavy hitters" – elements f_i with $f_i > \phi F_1$ for some ϕ – can be identified up to multiplicative error ϵ in ϕ by CountMin. CountMin achieves this bound by running the Heavy Hitters algorithm repeatedly and taking the minimum of observed estimator for each f_i. Due to the upward bias in Heavy Hitters estimators, the minimum observed estimator for f_i must exceed $\phi(1+\epsilon)$ for f_i to be reported; else CountMin's probabilistic guarantees won't hold. Analogously, the minimum observed f_i must be less than $\phi(1-\epsilon)$ for f_i to be reported as not being a heavy hitter.

In more detail, the algorithm[1] is:

```
Initialize(r, L):
    array S[L][w]
    L hash functions h_1,...,h_L, into {1,...,w}

Process(int i):
    for (j = 0; j < L; ++j)
        S[j][h_j(i)] += 1

Estimator:
    foreach i in PossibleIP:
        f̂_i = median_j(S[j][h_j(i)]
```

Total space required is $O(\frac{\log n}{\epsilon\phi})$.

2 CountMin Linearity

Suppose two different frequency estimates f' and f'' are computed over the same range $[n]$ using the same hash h_j. In this case, the following linearity property holds:

$$\text{CountMin}(f' + f'') = \text{CountMin}(f') + \text{CountMin}(f'')$$

[1] verbatim from the pervious lecture's notes

This follows immediately from the fact that each item in each stream gets counted onc, and both streams count item i in the same bucket $h_j(i)$: adding up f' and f'' amounts to adding their buckets individually and thus adding up the frequencies of each i.

3 CountSketch

3.1 P-norms

(This section is here due to its timing in the lecture. It appears somewhat later in the lecture slides.) The p-norm of a vector x, denoted by $||x||_p$ is defined as:

$$||x||_p = \left(\sum (x_i)^p\right)^{1/p}$$

There are two special cases deserving of attention. The 0-norm just counts the number of non-zero elements in x, because x_i^p is always either zero (if x_i is zero) or one (otherwise). The $\infty - norm$ plucks out the largest element of x, the intution being that in the limit as $p \to \infty$, the p-norm converges on $\max_{i \in x} i$.

3.2 Generalizations of CountMin

While CountMin is linear for $f' + f''$ in the cases observed so far, there are generalizations that it is less effective for. For instance, if both positive and negative frequencies are allowed, our reliance on selecting the minimum value becomes problematic. We also need to redefine "heavy hitters", for instance by using absolute values of frequencies, like in this criterion: $|f_i| \geq \phi \sum_j |f_j|$.

3.3 CountSketch: the Algorithm

One improvement to CountMin is to focus not on estimating f_i itself but instead estimate f_i^2. This can be accomplished by applying Tug of War within each bucket. As we saw previously, Tug of War estimates F_2, so applying Tug of War to all estimators of f_i gotten from repeated instance of Heavy Hitters will yield an estimator for f_i^2. At that point, the median trick may be applied.

The resulting algorithm is this:

```
Initialize(L, w):
    array S[L][w]
    L hash functions h_1,...,h_L, into {1,...,w}
    L hash functions r_1,...,r_L, into {-1,1}

Process(int i, real δ_i):
    for (j = 0; j < L; ++j)
        S[j][h_j(i)] += r_j(i)δ_l

Estimator:
    foreach i in PossibleIP:
        f̂_i = median_j(S[j][h_j(i)]
```

4 From CountSketch to Compressed Sensing

4.1 k-sparse Approximation

A large piece of data like an image often must be compressed to be stored. One approach to compression is to take the Fourier transform of the data (which will be a vector) and store only a subset that is hopefully representative enough to make producing an approximation to the original dataset possible. This motivates the following definition:

Definition 1. *Given a vector $f \in \mathbb{R}^n$, a k-sparse approximation to it is $f^* \in \mathbb{R}^k$ such that $||f^* - f||$ is minimized.*

For any vector $f \in \mathbb{R}^n$, its k-sparse approximation is f^* consisting of its k largest (by absolute value) elmeents.

4.2 Compressed Sensing

Definition 2. *Compressed sensing is the following problem: for some vector $f \in \mathbb{R}^n$, provide a matrix S such that an approximation to the k-sparse approximation to f may be computed from Sf*

The problem may be solved trivially by choosing $S = I_n$ (the $n \times n$ identity matrix), but the accomplishment is in achieving a good space/accuracy tradeoff. The following theorem achieves such a tradeoff:

Theorem 3. *Using $O(k \log n)$ space, the k-sparse approximation may be approximated with the following error guarantee by $\hat{f}$:*

$$||f^* - f|| \leq \min_{k-sparse \hat{f}} ||\hat{f} - f||$$

Superior results may be achieved by adaptively updating S, but this was not elaborated on during the lecture.

4.3 CountSketch as CS Special Case

Let S be a $k \times n$ matrix in which each row has exactly n/k entries set to 1 and the rest set to 0, and let each column have only a single entry set to 1. Then the $k \times n$ matrix is essentially a hash table with k buckets – the elements are uniformly distributed across buckets (hence n/k per row) and appear in exactly one bucket (hence 1 per column). Then, taking Sf amounts to hashing adding up the frequencies for each of the k buckets. The heavy hitters correspond in this case to the k largest elements, for teh choice of $\phi = 1/2k$.

5 Moments

Definition 4. *The p-th moment of f, denoted by F_p, is $\sum f_i^p$.*

Compressed sensing's performance varies for different moments. In the cases of F_0 and F_2, we can approximate in space only $O(\log n)$ as we saw in the Flajolet-Martin and Tug of War algorithms, respectively. In the case of $p = \infty$, the moment is inapproximable as shown in Lecture 3, although heavy hitters may be determined. For $2 < p < \infty$, we can use Precision Sampling (next section) to approximate using space only $\Theta(n^{1-2/p} \log^2 n)$

6 Precision Sampling

6.1 Estimate a Sum

Given n numbers $a_1, \ldots, a_n \in [0,1]$, how can one estimate $a_1 + \ldots + a_n$ at minimal cost? "Standard sampling" is the obvious solution – randomly pick a subset of size m, take its average, and multiply by n/m to get the estimator $\widetilde{S}$. Unfortunately, to get constant additive error we need a sample of size $\Omega(n)$. Why? The Chebyshev bound tells us that 90% of the time,

$$S - O(n/m) < \widetilde{S} < S + O(n/m)$$

Tightening that bound to any arbitrary $S \pm \epsilon$ requires m be of the same order as n.

6.2 Precision Sampling Framework

An alternative to standard sampling is to allow some error in the values of the a_i rather than risk the representativeness of the subset sampled. This approach is "precision sampling": an algorithm is given access to $\widetilde{a_i}$ such that $a_i - u_i \leq \widetilde{a_i} \leq a_i + u_i$, for predetermined values of u_i. The challenge is then to achieve a good tradeoff between u_i (which are considered costly if small) and the accuracy of the resulting estimator $\widetilde{S}$).

Somewhat more formally, precision sampling can be thought of as a game.

- Adversary: fix $a_1, \ldots, a_n$
- Player: choose $u_1, \ldots, u_n$
- Adversary: fix $\widetilde{a_1}, \ldots, \widetilde{a_n}$ such that $|a_i - \widetilde{a_i}| < u_i$, and provide these to Player
- Player: output estimate $\widetilde{S}$ such that $|\sum a_i - \gamma \widetilde{S}| < 1$, for $\gamma \approx 1$

The costliness of Player's solution is simply computed as $\sum \frac{1}{u_i}$, and the average cost is $\frac{1}{n} \sum \frac{1}{u_i}$

6.3 Precision Sampling Lemma and Algorithm

Lemma 5. *With 90% probability of success and average cost $O(\log n)$, one can get additive error $O(1)$ multiplicative error of 10, i.e.*

$$S/10 - O(1) < \widetilde{S} < S/10 + O(1)$$

The algorithm that achieves this is straightforward:

- Draw each u_i randomly from Exp(1), a probability distribution described by probability density function: $p(x) = e^{-x}$
- Return $\widetilde{S} = \max_i \widetilde{a_i}/u_i$

Proof of accuracy bound:
As per the problem definition, for all i, $|a_i - \widetilde{a_i}| \leq u_i$ from which it follows that $|a_i/u_i - \widetilde{a_i}/u_i| \leq 1$. We further know that $u_1, \ldots, u_n$ are drawn form Exp(1).

The exponential distribution satisfies the following property: for any $\lambda_1, \ldots, \lambda_n > 0$

$$\min_{i \in [n]} \frac{u_i}{\lambda_i}$$

is distributed as

$$\frac{u}{\sum \lambda_i}$$

where u has Exp(1) distribution as well. Taken together, these give

$$\tilde{S} = \max \frac{\widetilde{a_i}}{u_i} = \frac{\sum a_i}{\text{Exp}(1)} \pm 1$$

.

Now note that $Exp(1) \in [1/10, 10]$ with probability $e^{-1/10} - e^{-10} > 0.9$. Hence

$$S/10 - 1 \leq \tilde{S} \leq 10S + 1.$$

Proof of average cost:
Average cost is $\mathbb{E}[1/u_i]$. Because we got each u_i by drawing from Exp(1), this amounts to $\mathbb{E}[1/\text{Exp}(1)]$. That, in turn, is the integral

$$\int_0^\infty \frac{1}{u} e^{-u} du$$

The integral diverges, unfortunately, but can be broken up into coverging and diverging parts

$$\int_0^1 \frac{1}{u} e^{-u} du + \int_1^\infty \frac{1}{u} e^{-u} du$$

The right-hand portion is ≤ 1 because both $\frac{1}{u}$ and e^{-u} are always ≤ 1 on $[1, \infty]$ so this must hold true for the integral of their product. This leaves the left-hand portion which still diverges but may be decomposed into

$$\int_0^{1/n^3} \frac{1}{u} e^{-u} du + \int_{1/n^3}^1 \frac{1}{u} e^{-u} du$$

Once again, the right-hand portion can be bounded, this time by $O(\ln(n^3))$ due to results from analysis. The left-hand portion still diverges, but is consequential only if $u < 1/n^3$ – and that happens with probability $O(1/n^3)$. Thus if we allow for a $O(1/n^3)$ chance of the algorithm failing, we have bounded the cost of the algorithm by $O(\ln(n^3)) + O(1)$, which is itself $O(\log n)$.

The slides continue past this but the lecture did not.