Data Science Through a Geometric Lens	Fall 2019
Lecture 04 — October 9, 2019	
Prof. Cyrus Rashtchian	*Topics: AMS Sketch and Majority Algorithm*

Overview. Last time we saw the better distinct elements, without infinite precision hash functions. The main idea was really about geometric bucketing of the stream.

Today we will talk about two things. First, we discuss estimating ℓ_2 norm of a data stream. We introduce two methods for estimating ℓ_2 norm: a) Alon-Matias-Szegedy (AMS) algorithm [1] b) Johnson-Lindenstrauss (JL) lemma [2]. In this lecture we cover the AMS algorithm, which often appears in textbooks and notes because it is a very elegant and simple algorithm. In the next lecture, we take a different view on ℓ_2 and cover the JL lemma, which is a very powerful tool which is widely used in designing big data algorithms. Today we will also talk about Heavy Hitters and Majority algorithm.

1 Review: Data Stream Model

To recap, the *data stream model* considers the situation where we continuously receive a stream of data, do computation about it and answer queries about the data seen so far. We want to deal with massive data sets, so we assume that we only see the data in this order. Typically, we cannot access the data that has already passed through. The tricky/interesting aspect of this model is that we can not remember all the data. But still we want to do something meaningful, and answer the queries within some accuracy using limited memory.

More precisely, our storage is sub-linear in the size m seen so far and the universe size n. Usually our algorithm is randomized, and we only guarantee success with probability $1-\delta$. Also, the answer of the query is usually an approximation of the actual value, so our answer is an up to $1 \pm \varepsilon$ multiplicative approximation of the real answer of the query

Vector and Norm Interpretation. If the stream consists of integers from $[n]$, then we can represent their frequencies as a vector $x = (x_1, x_2, \ldots, x_n)$ where x_i equals the number of times that element i was in the stream. If were using $O(n \log(m))$ bits of storage (because after m stream elements, the value of x_i is at most $\log m$). Then we could just store this vector and increment x_i when we see i in the stream.

Notice that the distinct element problem is just determining $\|x\|_0$, and the counting problem is to determine $\|x\|_1$. Today we will be interested in $\|x\|_2$.

We can also handle multiplicities/weights. If we see element i along with a count c, then the can represent the update as a pair (i, c), which corresponds to the operation $x_i \leftarrow x_i + c$. Although we do not cover this, it is also interesting to allow the weights to be negative ($c < 0$), where we decrement the items. This makes many algorithms more complicated.

Motivation For Estimating ℓ_2 Norm. The point is that $\|x\|_2$ is an important statistic for vector x. If we think of x as a empirical distribution for data samples appeared in the stream, then $\|x\|_2$ corresponds to the second order moment of x which reveal the spickyness of x in some sense. Since $\|x\|_1 = n$ is fixed, $\|x\|$ takes its minimum value when every $x_i = n/m$ which corresponds to the uniform distribution. Similarly, $\|x\|_2$ gets bigger when x_i is further form uniform. Estimating the ℓ_2 norm of the data is originated in database application (estimating the join size, because joins end up multiplying the number of the elements from each side). So it's actually used and implemented in real life!

2 AMS Algorithm

The basic idea for AMS algorithm is to get an unbiased estimator of $\|x\|_2^2$ by linear sketching and try to prove concentration by bounding the variance the estimator. We will do AMS, AMS+, and AMS++ as usual. The punchline is that we can output an estimate $\widetilde{Z}$ of $\|x\|_2^2$ that a $1+\varepsilon$ approximation with probability $1-\delta$. Formally written we get that

$$\Pr\left(\left|\widetilde{Z} - \|x\|_2^2\right| \geq \varepsilon \|x\|_2^2 \right) \leq \delta.$$

The total space will be

$$O\left(\frac{\log n}{\varepsilon^2} \cdot \log(1/\delta)\right).$$

2.1 Description of the AMS algorithm

1. Choose $Y_1, Y_2, \ldots, Y_n$ i.i.d. random variables with $\Pr(Y_i = 1) = \Pr(Y_i = -1) = 0.5$
2. Initialize $Z \leftarrow 0$
3. For each update (i, c) to the stream (a.k.a. element i with count c) do:

$$Z \leftarrow Z + c \cdot Y_i$$

4. Output Z^2

The main idea: We are using the same random variable Y_i when we see element i. In other words, for each increment "$+c$" to x_i for seeing the ith element, we are multiplying by the same Y_i value. In other words, we have that

$$Z = \sum_{i=1}^{n} x_i \cdot Y_i.$$

Since Z is linear in x, when we update x by (i, c), we only need to increment Z by cY_i. Then our estimator for $\|x\|_2^2$ is Z^2. This is a very simple with very clever idea. Next we will see the analysis of the correctness of the algorithm.

Analysis. First we show that show that this is an unbiased estimator.

Claim 1. $\mathbb{E}(Z^2) = \|x\|_2^2$

Proof. We have just argued that $Z = \sum_{i=1}^{n} x_i \cdot Y_i$. Then we can expand Z^2 as follows:

$$Z^2 = \left(\sum_{i=1}^{n} x_i \cdot Y_i\right)^2 = \sum_{i=1}^{n} (x_i \cdot Y_i)^2 + \sum_{i \neq j} x_i x_j \cdot Y_i Y_j.$$

Since $Y_i = \pm 1$, we have that $Y_i^2 = 1$. So the first term is just $\|x\|_2$, and therefore, by linearity of expectation,

$$\mathbb{E}[Z^2] = \|x\|_2 + \sum_{i \neq j} x_i x_j \cdot \mathbb{E}[Y_i Y_j].$$

Now, the Y_i are independent random variables (or at least pairwise independent). So the second term is actually zero because

$$\mathbb{E}[Y_i Y_j] = \mathbb{E}[Y_i]\,\mathbb{E}[Y_j] = 0 \cdot 0 = 0.$$

We conclude that $\mathbb{E}[Z^2] = \|x\|_2$ as desired. □

Next we bound the variance.

Claim 2. $\mathrm{Var}(Z^2) \leq 2\|x\|_2^4$.

Proof. We start with the definition of variance for Z^2.

$$\mathrm{Var}(Z^2) = \mathbb{E}(Z^4) - \mathbb{E}(Z^2)^2 = \mathbb{E}(Z^4) - \|x\|_2^4$$

We decompose $\mathbb{E}(Z^4)$ as

$$\mathbb{E}(Z^4) = \sum_{i,j,k,l} \mathbb{E}(x_i x_j x_k x_l Y_i Y_j Y_k Y_l) = \sum_{i,j,k,l} x_i x_j x_k x_l \,\mathbb{E}(Y_i Y_j Y_k Y_l)$$

Notice $\mathbb{E}(Y_i Y_j Y_k Y_l)$ is 0 if there is one index only appear once in i, j, k, l so we only need to consider the case where every distinct index appears at least twice. Then there are two cases such that $\mathbb{E}(Y_i Y_j Y_k Y_l) = 1$:

- There are two distinct pairs in (i, j, k, l) each occurring twice
- All four of i, j, k, l are identical.

So we have

$$\mathbb{E}(Z^4) \;=\; \frac{1}{2}\binom{4}{2} \cdot \sum_{i \neq j} x_i^2 x_j^2 + \sum_i x_i^4 \;=\; 3\sum_{i \neq j} x_i^2 x_j^2 + \sum_i x_i^4$$

Then we can bound $\mathbb{E}(Z^4)$ by

$$\mathbb{E}(Z^4) \;=\; 3\sum_{i \neq j} x_i^2 x_j^2 + \sum_i x_i^4 \;=\; 2\sum_{i \neq j} x_i^2 x_j^2 + \|x\|_2^4 \;\leq\; 3\|x\|_2^4$$

Putting this together we have $\mathrm{Var}(Z^2) = \mathbb{E}(Z^4) - \mathbb{E}(Z^2)^2 \leq 3\|x\|_2^4 - \|x\|_2^4 = 2\|x\|_2^4$. □

We have just established that

$$\mathrm{Var}(Z^4) \leq 2\|x\|_2^4$$

To motivate taking many estimates (AMS+), we can try to use Chebyshev's inequality and see why it isn't very good

$$\Pr\left(\, |\, \mathbb{E}(Z^2) - \|x\|_2^2| \geq \sqrt{2}c\|x\|_2^2 \, \right) \leq 1/c^2$$

We can observe that this bound is often too loose to be informative for approximating $\|x\|_2^2$. For example, if we choose $c = 3$ (corresponding to error probability $\delta = 1/9$), then we have

$$\Pr(|\, \mathbb{E}(Z^2) - \|x\|_2^2| \leq 3\sqrt{2}\|x\|_2^2) \leq 1/9$$

However we know that $E(Z^2) \geq 0$, so the lower bound it gives is even worse than the trivial bound (that is, we always know that $|\, \mathbb{E}(Z^2) - \|x\|_2^2| \leq \|x\|_2^2$ without Chebyshev).

To improve the error bound, we repeat this estimator s times independently, also known as AMS+, and then we will take the median-of-means AMS++ to get good error probability.

2.2 AMS+

We maintain $Z_1, Z_2, \ldots, Z_s$ where for every j,

$$Z_j = \sum_{i=1}^{n} Y_{ji} x_i.$$

Now we have s times more random variables. That is, Y_{ij} are i.i.d. random variables with the same distribution as above.

Then our estimator for $\|x\|_2^2$ is $Z = (\sum_j Z_j^2)/s$.

Taking taking the mean of s independent estimators does not affect the mean of the estimator (still an unbiased estimator) but it will reduce the variance by a factor of s.

More precisely, to analyze this improved AMS+ estimator, we compute the expectation and variance of the estimator:

$$\mathbb{E}(Z) = \frac{1}{s} \cdot \sum_{j=1}^{s} \mathbb{E}(Z_j^2) = \|x\|_2^2 \qquad \text{and} \qquad \mathrm{Var}(Y) = \frac{1}{s^2} \cdot \sum_{j=1}^{s} \mathrm{Var}(Z_j^2) \leq \frac{2\|x\|_2^4}{s}.$$

Chebyshev's inequality gives

$$\Pr(|\, \mathbb{E}(Z^2) - \|x\|_2^2| \leq c\sqrt{2/s}\|x\|_2^2) \leq 1/c^2$$

If we set $c = \Theta(1)$ and $s = \Theta(1/\varepsilon^2)$, we get a $(1 \pm \epsilon)$ approximation with constant probability!

The space we need for this algorithm is dominated by the space of storing Z_j for all j if we temporarily ignore the space to generate Y_{ji}. Recall that m is the number of stream elements and n is the universe size. For a fixed j, the maximum possible value for Z_j is mn so to store Z_j we need $\log(mn)$. And there are $O(1/\varepsilon^2)$ such Z_j so the total space we need is $O(\log(mn)/\epsilon^2)$ bits. We next do AMS++ to get $1 - \delta$ success probability.

2.3 AMS++

We take the median of $t = O(\log(1/\delta))$ copies of AMS+. From the first two lectures (the analysis is the exact same) we can use a Chernoff bound to get the error probability down to $1 - \delta$. Putting this all together, we have an algorithm with space $st \log n$, or plugging in these values:

$$O\left(\frac{\log n}{\varepsilon^2} \cdot \log(1/\delta)\right),$$

where we assume that $m = O(n^b)$ for some constant b, that is, the length of the stream is pretty much the same as the universe size as far as $O(\log n)$ is concerned.

2.4 Using 4-wise independent hash functions to make it awesome

Now we can consider how to actually generate these Y_i. If we look at the proof of correctness in detail, we can realize that we only need 4-wise independence of Y_i because throughout the analysis when we computing $\mathbb{E}(Z^2)$ and $\mathbb{E}(Z^4)$, the maximum degree of the polynomial in Y_i is 4. If we pick Y_i by 4-wise independent hash function, all the calculations of the expectations will remain the same so the analysis and error bounds still hold. We know that we can generate Y_i from $O(\log n)$ random bits so this is not dominant space consumption in the algorithm.

3 Heavy Hitters

Let x be a super high dimensional vector, where x_i is the count of element i (e.g., the number of times the ith hashtag on Instagram was used last year). If we can't store x_i exactly, we want an approximation. And if we can't store x, we still want to answer queries about the most frequent (a.k.a. the 'heavy' elements).

We have following definitions:

1. ℓ_1 point query: $query(i) = x_i \pm \varepsilon\|x\|_1$
2. ℓ_1 heavy hitters: $query()$ return $L \in [n]$ s.t. :
   (a) $|x_i| > \varepsilon\|x\|_1 \Longrightarrow i \in L$
   (b) $|L| = O(1/\varepsilon)$

The first question is about estimating the count of an item (as long as it is heavy enough). The second question is about finding a heavy item.

Note that the number of ε-heavy hitters, i.e. those i satisfying $|x_i| > \varepsilon\|x\|_1 \to i \in L$, is less than $1/\varepsilon$, so the second requirement is just saying that L should not be more than a constant factor larger than this maximum possible size.

The heavy hitters problem shows up, for example, when we are trying to find frequent items in a data stream. In the turnstile model with deletions, if we interpret updates in one time period T as decrementing frequencies and in some other disjoint time interval T' as increasing frequencies, then note that x during a query will be the difference of two frequencies. Then a heavy hitter corresponds to an index i that *changed* significantly in frequency, and thus turnstile heavy hitters algorithms can also be used to detect large frequency changes.

We will also study the notion of point query with a tail guarantee, and "ε-tail heavy hitters".

1. ℓ_1 point query with a tail guarantee: $query(i) = x_i \pm \varepsilon \|x_{\overline{[1/\varepsilon]}}\|_1$
2. ℓ_1 tail heavy hitters: $query()$ return $L \subseteq [n]$ s.t. :
   (a) $|x_i| > \varepsilon \|x_{\overline{[1/\varepsilon]}}\|_1 \Longrightarrow i \in L$
   (b) $|L| = O(1/\varepsilon)$

Here we use $x_{\overline{[k]}}$ to denote the vector x after zeroing out its largest k entries in magnitude. Note that the number of i such that $|x_i| > \varepsilon \|x_{\overline{[k]}}\|_1$ is at most $k + 1/\varepsilon$, since other than the set $S \subset [n]$ of top k entries in x, the number of other i satisfying $x_i > \varepsilon \sum_{j \notin S} |x_j|$ must be less than $1/\varepsilon$.

3.1 Majority Algorithm

Consider a stream $i_1, i_2, \ldots \in [n]$.

1. Initialize $X \leftarrow i_1$ and `count` $\leftarrow 1$.
2. For $j = 2, 3, \ldots$
   - If $i_j == X$, then `count` $\leftarrow$ `count` $+ 1$
   - Else if $i_j \neq X$, then `count` $\leftarrow$ `count` $- 1$
   - If `count` $== 0$, then set $X \leftarrow i_j$

The total space used is $\log n + \log m$ when we have seen m elements so far. It takes $\log n$ bits to store X. And the counter takes $\log m$ space.

We claim that at any point in the stream, the above algorithm has X set to the majority element seen so far. That is, if there have been m elements seen so far, and i^* has appeared more than $m/2$ times, then $X = i^*$.

Claim 3. *Let f_i be the number of times i appears. If $f_i > m/2$, then $X = i$ at the end.*

Proof. The key idea is to look at when the counter is zero. This may happen many times. Let's keep track of the values X when it does. That is, let's say that $i_1, i_2, \ldots, i_t$ are the values of X right before the counter becomes zero and we switch to something else (so t is the number of times X changes).

We argue that in each of these intervals, the value i_j must occur *exactly* half the times (that is half the length of that interval). The counter goes up every time we see i_j and down when we see anything else. And at the end of the interval, $X = i_j$ changes to something else and the counter is zero. Therefore, the number of counter increments is even, and i_j contributes to exactly half of these.

Looking over all t intervals, this can account for at most $m/2$ times that we see the majority element i. In other words, at the end, the counter must be > 0 and $X = i$ at the end. □

References

[1] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Johnson, William B., and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics.*, 26(1):189–206, 1984.