COMP 480/580 — Probabilistic Algorithms and Data Structure Oct 08,2024

Lecture 11: Stream estimation

Lecturer: Anshumali Shrivastava

Scribe By: Alex Bock (ab215), Natalie Kokroko (nk62), Jaywon Koo (jk125)

1 Count sketch

1.1 Motivation

Imagine a scenario in which we have access to a data stream, which reports individual data points separated by time intervals. Thus, as a timestep t, we receive an observation i_t and have observed all $i_1, \ldots, i_{t-1}$. Using a data stream, how can we keep count of the number of instances of values of i we observe? For example, in a stream of URLs, how can we know the number of times we've observed a specific URL?

We have already seen count-min sketch, which estimates this using storage space sublinear in the number of unique values observed (n). Count-min sketch achieves this by selecting d independent hash functions from a 2-universal family, each of which map to $k << n$ addresses in memory. For an input i, count-min sketch updates a matrix C at index $j, h_j(i)$ for all $j \in [d]$. The algorithm is then able to estimate the number of occurrences of i by choosing the smallest index $j, h(j)$, incurring some uncertainty due to collisions.

However, count-min sketch is designed specifically for incremental positive counts (+1 with every observation). Let's generalize the notion of counting by assuming that associated with each i_t is some weight Δ_t. Instead of adding 1 with each observation, we add its weight. A key observation here is that weights may be negative. Count-min sketch chooses the smallest count with which a requested token is associated, since noise introduced by collisions only accumulates in the positive direction. However, we need to account for noise accumulating in both the positive and negative directions.

1.2 Algorithm

Count sketch employs a uniformly sampled set of independent, 2-universal hash functions $h_1, \ldots, h_d$ as well as a separate set of hash functions $s_1, \ldots, s_d$, the **sign functions**, where s_i maps tokens to $\{-1, +1\}$ independently of how h_i maps tokens to $[k]$. We define two operations:

1. **Update**: Given a token i and a weight Δ_i, for all $j \in [d]$, we increment $C_{j,h_j(i)}$ by $s_j(i) \cdot \Delta_i$
2. **Estimate**: Given a token i, find all counts to which i maps, or all $C_{j,h(_j(i)}$ for all $j \in [d]$, and report the median of these counts.

Since the update and estimation procedures rely on hash-based indexing, we are faced with the possibility of collisions, meaning that there is some uncertainty in any counter's ability to measure the actual number of observations of a token i; it includes increments for tokens that clash with i as well. (This error is formalized below.) However, the fact that our sign functions $s_1, \ldots, s_d$ map to $\{-1, +1\}$ uniformly allows us to treat collision-induced noise as zero, in expectation (employed below). For all d counts associated with some value i, we choose the median, expecting the median count to be closest to zero and to include minimum noise.

1.3 Analysis

The following analyses[1] apply to some choice of $h \in \{h_1, \ldots, h_d\}$ and $s \in \{s_1, \ldots, s_d\}$ — effectively, $d = 1$. The count matrix C in this section is therefore a single-row matrix.

1.3.1 How good is our estimator?

Let c_i be the actual count of some token i. With a hash function h, the best estimate our sketch can provide is:

$$\hat{c}_i = s(i)\, C_i$$

Let $\mathbb{1}_{x=y}$ be an indicator random variable indicating equality of x and y. Our estimate expands to

$$\begin{aligned}
\hat{c}_i &= s(i) \sum s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)} \\
&= s(i)^2\, c_i\, \mathbb{1}_{h(i)=h(i)} + \sum_{j \neq i} s(i)\, s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)} \\
&= c_i + \sum_{j \neq i} s(i)\, s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)}
\end{aligned}$$

We achieve the last simplification because:

- As $s(i) \in \{-1, +1\}$, $s(i)^2 = 1$
- Clearly, $\mathbb{1}_{h(i)=h(i)} = 1$

This expression of $\hat{c}_i$ is essentially the true count of i plus an error term. The error term here represents noise introduced by tokens j that collide with i, i.e. for which $h(i) = h(j)$. Our expected estimated count is therefore:

$$\mathbf{E}[\hat{c}_i] = c_i + \mathbf{E}\left[\sum_{j \neq i} s(i)\, s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)}\right]$$

Given that s distributes *uniformly and independently* over $\{-1, +1\}$, $\mathbf{E}[s(i)] = 0$. Hence we eliminate the expected error term, yielding:

$$\mathbf{E}[\hat{c}_i] = c_i$$

We conclude that we have a reasonable estimator.

[1] Adapted in part from (1). See (3) for an original introduction and further analysis.

1.3.2 Finding an error bound

Our next goal is to find a bound on the size of the error term. We can express this bound relative to the variance of $\hat{c}_i$, which is defined as follows:

$$\begin{aligned}
\mathbf{Var}[\hat{c}_i] &= \mathbf{E}[(\hat{c}_i - \mathbf{E}[\hat{c}_i])^2] \\
&= \mathbf{E}[(s(i)\, C_i - c_i)^2] \\
&= \mathbf{E}[(c_i + \sum_{j \neq i} s(i)\, s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)} - c_i)^2] \\
&= \mathbf{E}[(\sum_{j \neq i} s(i)\, s(j)\, c_j\, \mathbb{1}_{h(j)=h(i)})^2] \\
&= \mathbf{E}[s(i)^2 \cdot \sum_{j \neq i} \sum_{k \neq i} s(j)\, s(k)\, \mathbb{1}_{h(j)=h(i)}\, \mathbb{1}_{h(k)=h(i)}] \\
&= \mathbf{E}[\sum_{j \neq i} \sum_{k \neq i} s(j)\, s(k)\, c_j\, c_k\, \mathbb{1}_{h(j)=h(i)}\, \mathbb{1}_{h(k)=h(i)}] \\
&= \mathbf{E}[\sum_{l \neq i} s(l)^2\, c_l^2\, \mathbb{1}_{h(l)=h(i)}] + \mathbf{E}[\sum_{j \neq i} \sum_{k \neq j} s(j)\, s(k)\, c_j\, c_k\, \mathbb{1}_{h(j)=h(i)}\, \mathbb{1}_{h(k)=h(i)}]
\end{aligned}$$

In the right-hand term, we use the same intuition as before and conclude that since $\mathbf{E}[s(j)] = 0$, we can eliminate the right-hand term. But in the left-hand term, we see that $\mathbf{E}[s(l)^2] = 1$. We are left with:

$$\mathbf{Var}[\hat{c}_i] = \mathbf{E}[\sum_{l \neq i} c_l^2 \mathbb{1}_{h(l)=h(i)}]$$

An indicator function like $\mathbb{1}$ yields only 0 or 1, and so clearly the square of an indicator function's output is equal to the output; thus, we can drop the exponent on the indicator as above.

Given that h is a universal hash function, we can see that $\mathbf{E}[\mathbb{1}_{h(l)=h(i)}] = 1/k$ for a hash table of size k; the probability of any two tokens colliding is essentially random chance. Using this, and observing that the summation over squares of elements of c is by definition the squared l_2 norm of c, we have:

$$\mathbf{Var}[\hat{c}_i] = \frac{1}{k}\mathbf{E}[\sum_{l \neq i} c_l^2] = \frac{1}{k}\sum_{l \neq i} c_l^2 = \frac{\|c\|_2^2}{k}$$

With a simple expression of variance, we can now bound our error with respect to the standard deviation σ, by definition the square root of the variance. We use Chebyshev's inequality[2]:

$$\mathbf{P}[|\hat{c}_i - \mathbf{E}[\hat{c}_i]| > \epsilon\sigma] = \mathbf{P}[|\hat{c}_i - \mathbf{E}[\hat{c}_i]| > \epsilon \cdot \frac{\|c\|_2}{\sqrt{k}}] < \frac{1}{\epsilon^2}$$

Choosing $\epsilon = \sqrt{3}$ gives us $\mathbf{P}[|\hat{c}_i - \mathbf{E}[\hat{c}_i]| > \sqrt{\frac{3}{k}}\|c\|_2] < \frac{1}{3}$.

[2] ϵ is normally k, but we use ϵ to avoid overloading k, which already stands for hash table size.

2 Count distinct

2.1 Idea

The Count Distinct problem is a challenge in streaming algorithms where it estimates the number of unique elements in a stream using sublinear memory. As data streams continuously grow, it becomes impractical to store all the observed elements, necessitating approximate methods to track distinct items efficiently. This problem finds applications in many areas, including database optimization and web analytics, where systems must estimate the number of unique elements without requiring vast amounts of memory.

2.2 Problem Definition

Let's assume there is a data stream $S = (i_1, i_2, ..., i_t)$, where each element i_t is observed at time t. The goal is to estimate the number of distinct elements, $D(S)$, seen in the stream. In large scale data scenarios, it is hard to achieve this estimation using significantly less space than would be required to store all unique elements. This is a non-trivial problem as memory constraints often prevent storing the complete set of distinct elements. Also, it is often only possible to process each element once without revisiting earlier data in streaming applications. Therefore, the algorithm must balance the trade-off between accuracy and memory efficiency, ensuring the estimate remains reasonably accurate while minimizing space usage.

2.3 Algorithm

The HyperLogLog algorithm (4) is one of the most widely used algorithms to address the Count Distinct problem. It hashes each element to a uniformly distributed value and tracks the position of the leftmost 1-bit in the binary representation of the hash, which provides insight into the rarity of the element. The algorithm uses multiple registers, each corresponding to a portion of the hash space, to record the maximum number of leading zeros observed for elements that map to it. After processing the stream, the harmonic mean of the register values is calculated, and bias correction is applied to estimate the cardinality. The algorithm achieves sublinear space complexity, requiring only $O(m)$ memory, where m is the number of registers, making it highly efficient for large-scale data streams.

3 Norm estimation on streams

3.1 Introduction

Norm estimation on data streams is a fundamental problem in streaming algorithms, where the goal is to estimate various norms of a vector or matrix with limited memory. There are different types of norms like (e.g., ℓ_1, ℓ_2, ℓ_p norms).(2)

This is particularly useful in applications such as network traffic monitoring, frequency analysis, and anomaly detection.

Consider a vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, where each y_i represents the frequency of an item in a data stream. The goal is to efficiently estimate different norms of $\mathbf{y}$ using minimal memory.

The p-norm of $\mathbf{y}$ is given by:

$$\|\mathbf{y}\|_p = \left(\sum_{i=1}^{n} |y_i|^p \right)^{1/p}$$

Some streaming norms include:

- ℓ_1 norm: $\|\mathbf{y}\|_1 = \sum_{i=1}^{n} |y_i|$
- ℓ_2 norm: $\|\mathbf{y}\|_2 = \left(\sum_{i=1}^{n} y_i^2\right)^{1/2}$
- ℓ_∞ norm: $\|\mathbf{y}\|_\infty = \max_i |y_i|$

The main challenge is that $\mathbf{y}$ cannot be stored in full due to memory constraints. Instead, updates are seen as pairs (i, δ), where each update modifies y_i by δ. The task is to estimate these norms accurately while minimizing memory usage and computational overhead.

3.2 Applications of Norm Estimation

Norm estimation in data streams is widely used in:

- **Frequency moment estimation:** Estimating moments $F_p = \sum_{i=1}^{n} |y_i|^p$, which is closely related to the ℓ_p norm.
- **Network traffic monitoring:** Estimating heavy hitters and anomaly detection by tracking frequency distributions. (5)

References

[1] Paul Beame. More heavy hitters: Count sketch, 2014. URL: `https://courses.cs.washington.edu/courses/cse522/14sp/lectures/lect05.pdf`.

[2] Amit Chakrabarti. Data stream algorithms, 2024. Accessed: October 14, 2024. URL: `https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf`.

[3] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 2004. Original introduction of count sketch. URL: `https://www.sciencedirect.com/science/article/pii/S0304397503004006`.

[4] Cheng-Wei Hu. Hyperloglog: A simple but powerful algorithm for data scientists, 2021. Accessed: October 15, 2024. URL: `https://towardsdatascience.com/hyperloglog-a-simple-but-powerful-algorithm-for-data-scientists-aed50fe47869`.

[5] S. Muthukrishnan. Data streams: Algorithms an applications, 2024. Accessed: October 15, 2024. URL: `https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/Muthu-Survey.pdf`.