SEEM5020 Course Project Report[1], 2023 Fall

YAN Yuhang Henry
StudentID
Email

Abstract

This report focuses on the characteristics and effectiveness of four classic Streaming Algorithms for Frequency Estimation and Range-based Frequency Estimation. It also introduces a new algorithm, Augmented Sketch(ASketch), which significantly reduces the error and time in performing both above estimations.

1. Introduction

This is the course project for SEEM5020: Algorithms for Big Data in CUHK at 2023 Fall. This project compares the performance of Misra-Gries Algorithm(1), Space-Saving Algorithm(2), Count-Min Sketch(3), Count-Sketch(4) and Augmented Sketch(5) when performing Frequency Estimation and Range-based Frequency Estimation on three datasets following a uniform, normal, and exponential distribution, respectively. After comparison, we found that: Misra-Gries and Space-Saving algorithms are suitable for identifying the most frequently occurring values; Count-Min can evaluate the frequency relationship of all data and can count with lower precision; Count-Sketch is more suitable for Range-based Frequency Estimation; ASketch shows excellent performance in all scenarios except uniformly distributed datasets.

2. Datasets

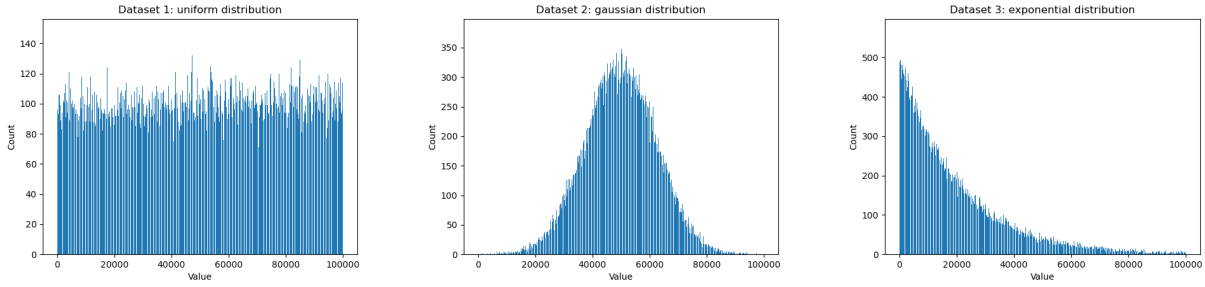
This project uses three self-created datasets, each of which consists of 10,000,000 data from different distributions, each of which should be an integer between 0 and 100,000. The data in each dataset follows different distributions as shown in the Fig. 1.
Dataset 1 Uniform distribution $X \sim U[a, b]$, where $a = 0, b = 100000$.
Dataset 2 Gaussian distribution $X \sim N(\mu, \sigma^2)$, where $\mu = 50000, \sigma = 12500$.
Dataset 3 Exponential distribution $X \sim \text{Exp}(\beta)$ where $\beta = 20000$.

3. Algorithms

3.1. Misra-Gries

The Misra-Gries algorithm is a frequency estimation algorithm in streaming data. For a data stream with n elements, Misra-Gries can ensure that the frequency estimation error for all items does not exceed γn (i.e. $(\gamma \cdot n, 1)$-approximation) by maintaining $k = \lceil 1/\gamma \rceil - 1$ counters. However, it tends to underestimate the frequency of occurrences of items with a significant likelihood.

[1]All the codes, datasets and results in this project can be found at `https://github.com/YanY-Henry/SEEM5020-Algorithms-for-Big-Data-Project`.



(a) Dataset 1 (b) Dataset 2 (c) Dataset 3

Fig. 1.—: The distribution of the datasets

3.2. Space-Saving

The Space-Saving algorithm is another frequency estimation algorithm in streaming data. For a data stream with n elements, Space-Saving can find all items that appear more than $\gamma n (\gamma < 1)$ times, and it can also ensure that the frequency estimation error for all items does not exceed γn (i.e. $(\gamma \cdot n, 1)$-approximation) by maintaining $k = \lceil 1/\gamma \rceil$ counters. However, it tends to overestimate the frequency of occurrences of items with a significant likelihood. Compared to Misra-Gries, the Space-Saving algorithm can typically provide more accurate frequency estimates as it always attempts to preserve the elements with the highest frequency. However, this can also require more computation time.

3.3. Count-Min

The Count-Min Sketch is a probabilistic data structure used for stream data processing. It can effectively perform point estimation (querying the frequency of an element) and range-based estimation (querying the sum frequency of a group of elements). It maintains a 2-dimensional array A$[d][w]$, and sets up a hash family $\mathscr{H}$ containing d 2-wise independent hash functions. Each hash function h_i is a mapping from $[m]$ to $[w]$. The Count-Min ultimately checks the smallest A$[i][h_i(e)]$ corresponding to element e, ensuring an $(\varepsilon n, \delta)$-approximation within $O(\frac{\log\log(1/\delta)}{\varepsilon})$ space.

In this project, we use the following method to construct the eligible hash family $\mathscr{H}$ of hash functions from $[m]$ to $[w]$:

- Pick a prime number p such that $p \geq m$ and $p \geq w$.
- Select random $\alpha \in \{1, 2, ..., p-1\}$ and random $\beta \in \{0, 1, 2, ..., p-1\}$, define:

$$h_{\alpha,\beta}(e) = 1 + (((\alpha e + \beta) \mod p) \mod m).$$

- This defines at most $p(p-1)$ hash functions, which constitute our $\mathscr{H}$.

3.4. Count-Sketch

The implementation and characteristics of the Count-Sketch are almost identical to those of Count-Min Sketch. However, it needs to additionally construct d mappings $g_i : [m] \to \{1, -1\}$, and ultimately uses the median of $g_i(e) \cdot A[i][h_i(e)]$ for estimation. In this project, we will use

$g_i = (e + \alpha_i)$ % $2 * 2 - 1$, random $\alpha_i \in \{0, 1\}$. (p.s. Lecture 3, Page 17 of this course introduces another variable, c_i. For the sake of simplicity in this project, we will assume that all $c_i = 1$.)

3.5. ASketch

Augmented Sketch is a hybrid frequency estimation algorithm that integrates the Space-Saving structure for top-k estimation and the Count-Min Sketch for frequency estimation of other items. It also provides an $(\varepsilon n, \delta)$-approximation with improved accuracy, particularly for low-frequency items. However, this comes at the cost of increased space complexity. In this project, we implement ASketch with $k = \lceil 1/\varepsilon \rceil$ counters for a Filter by Space-Saving structure and d hash functions for the Count-Min Sketch. The exact implementation of the algorithm can be found in the reference paper(5). Only pseudocode is given here.

Algorithm 1 Stream processing algorithm

Ensure: insert tuple (e, 1) into ASketch
1: lookup e in filter
2: **if** item found **then**
3: $new_{count}[e] \leftarrow new_{count}[e] + 1$
4: **else if** filter not full **then**
5: $new_{count}[e] \leftarrow u$
6: $old_{count}[e] \leftarrow 0$
7: **else**
8: update sketch with (e, 1)
9: **if** $freq[e] >$ min $freq[F]$ **then**
10: find minimum $freq$ item e_i in F
11: **if** $(new_{count}[e_i] > old_{count}[e_i])$ **then**
12: tmp = $new_{count}[e_i] - old_{count}[e_i]$
13: update sketch with (e_i, tmp)
14: **end if**
15: add k to filter
16: $new_{count}[e] \leftarrow$ estimated $freq[e]$
17: $old_{count}[e] \leftarrow$ estimated $freq[e]$
18: **end if**
19: **end if**

Algorithm 2 Query processing algorithm

Ensure: estimate frequency of item e
1: lookup item e in filter
2: **if** item found **then**
3: **return** $new_{count}[e]$ from filter
4: **else**
5: **return** estimated $freq[e]$ from sketch
6: **end if**
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:

4. Experimental Results[2]

For the datasets, we adjusted the parameters in the different algorithms so that for each point estimate, the error is as low as possible below 150 (i.e. constructing an $(150, 1-\delta)$-approximation, where δ is as large as possible). Because both the datasets and the algorithms contain random numbers, all results were experimented with three times and averaged.

[2]The figures and tables mentioned in Section 4 can be found in the Appendix.

4.1. Frequency Estimation[3]

We estimated all the values in the interval $[0, 100000]$ using the algorithms respectively, calculated the standard deviation(Table 1) and maximum error(Table 2), and plot scatter plots to compare the differences between the real and estimated frequencies(Fig. 2).

4.2. Range-based Frequency Estimation

Based on the Frequency Estimation, we performed five more Ranged-based Frequency Estimation using the dyadic tree-based method. The intervals for the five estimations are [0,1000], [24500, 25500], [49500, 50500], [74500, 75500], [99000, 100000]. In this problem, we are more interested in the standard deviation of the algorithm(Table 3) as well as the time consumed(Table 4).

5. Conclusion

All algorithms accurately identified the characteristics of the uniformly distributed dataset: the distribution frequencies are roughly the same, only the counting accuracy varies. However, in most cases, we are more concerned with the special values that appear in the data stream (i.e., the most frequent data), so our subsequent analysis will focus on Gaussian distributed datasets and exponential distributed datasets.

Misra-Gries is effective for identifying data frequency peaks but lacks in precise counting. Space-Saving can identify and count frequently appearing elements more accurately. But both performance is tied to the number of counters used. As counters decrease, so does their ability to count low-frequency data.

In Frequency Estimation, Count-Min and Count-Sketch provide rough frequency relationship estimates across the dataset, but their counting accuracy needs improvement. Nonetheless, their performance in point estimation is acceptable given their primary design for Range-based Frequency Estimation.

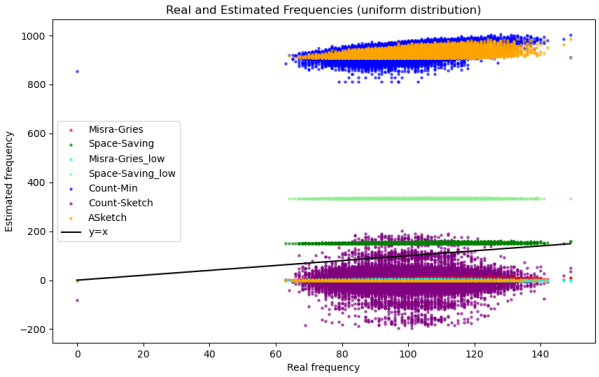
ASketch outperforms all in Frequency Estimation, providing high counting accuracy across the entire dataset. In Range-based Frequency Estimation, ASketch stands superior to Count-Min and Count-Sketch, completing estimations with greater speed and accuracy (here we don't consider Misra-Gries and Space-Saving which have numerous counters).

The below-par performance of Count-Min and Count-Sketch in both estimation problems was unexpected, suggesting the need for future research to improve their accuracy through parameter adjustments, hash function modifications or algorithm enhancements.

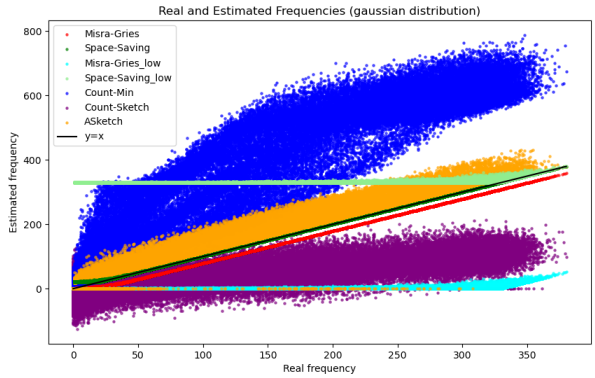
---

[3]There was an issue here! Due to an imbalance in data range and quantity during dataset generation, I allocated an excessive number of counters for the Misra-Gries and Space-Saving algorithms to theoretically ensure equal accuracy across all algorithms. This was UNFAIR as they also used a big space. I hard-coded the counter number for both algorithms to 3000, about half of the original value, and this led to noticeable changes in the results (Misra-Gries$_{low}$ and Space-Saving$_{low}$).

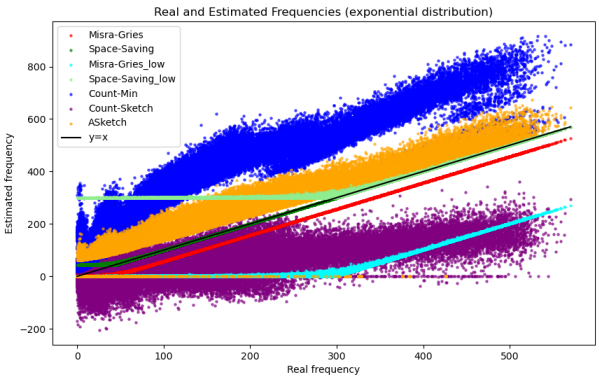
Appendix: Figures and Tables in Section 4



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Fig. 2.—: Comparison of real and estimated frequencies[4]

[4]The closer to the line y=x, the more accurate the estimated value is.

Table 1:: Standard Deviation in Frequency Estimation

Algorithms	Dataset 1 (Uniform distribution)	Dataset 2 (Gaussian distribution)	Dataset 3 (Exponential distribution)
Misra-Gries	99.15	16.56	32.75
Space-Saving	70.00	4.57	12.84
Misra-Gries$_{low}$	100.40	148.63	134.21
Space-Saving$_{low}$	152.46	103.80	94.66
Count-Min	839.01	300.82	265.44
Count-Sketch	102.88	155.95	170.70
ASketch	207.57	32.53	24.08

Table 2:: Maximum Error in Frequency Estimation

Algorithms	Dataset 1 (Uniform distribution)	Dataset 2 (Gaussian distribution)	Dataset 3 (Exponential distribution)
Misra-Gries	145	51	93
Space-Saving	138	51	93
Count-Min	885	747	688
Count-Sketch	298	492	721
ASketch	371	178	166

Table 3:: Standard Deviation in Range-based Frequency Estimation

Algorithms	Dataset 1 (Uniform distribution)	Dataset 2 (Gaussian distribution)	Dataset 3 (Exponential distribution)
Misra-Gries	30.72	119.62	167.14
Space-Saving	73.49	124.76	178.94
Count-Min	299.07	168.48	213.75
Count-Sketch	602.86	769.26	264.83
ASketch	169.26	82.39	149.63

Table 4:: Time consumed in Range-based Frequency Estimation

Algorithms	Dataset 1 (Uniform distribution)	Dataset 2 (Gaussian distribution)	Dataset 3 (Exponential distribution)
Misra-Gries	17	17	15
Space-Saving	342	436	41
Count-Min	42	41	459
Count-Sketch	46	47	44
ASketch	29	20	19

References

J. Misra and D. Gries, "Finding repeated elements," *Science of computer programming*, vol. 2, no. 2, pp. 143–152, 1982.

A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International conference on database theory*, pp. 398–412, Springer, 2005.

G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2004.

P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 1449–1463, 2016.