COMP90056 Assignment 2 Frequent Items in a Data Stream

Zhihao Huang 1052452

1.*Set-up* A description of the experimental set-up to aid reproducibility. For example, include the CPU frequency, memory, operating system, programming language, compiler (if applicable). [1 mark]

We implemented all algorithms in Java, compiled with *OpenJdk javac 11.0.6*. The executables are tested under macOS 10.15.7 on a MacBook Pro with a 2.3GHz dual-core CPU, 64MB of eDRAM and 8GB memory.

2.Generate input data according to the power-law distribution and plot the distribution of frequencies of items for z = 1.1, 1.4, 1.7, 2.0 with a logarithmic plot to demonstrate that they are as desired. In these cases, how many items have frequency at least 1% of the total number of items? [2 marks]

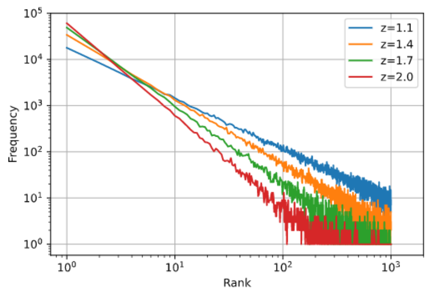


Figure 1: Stream with Zipf distribution

Here, we generated streams of data that is conforming to with a length of 10,000 taking values ranging from 0 to 1000. Figure 1 shows that when z increases, the frequency of rank drops faster as the rank goes up. In theoretical, there are 14, 13, 10, 8 frequent items when z is 1.1, 1.4, 1.7, 2.0, respectively.

3.Describe in your own words (one paragraph) the intuition behind how the Sticky Sampling algorithm identifies frequent items. Cite any references used apart from [1]. [1 mark]

Sticky Sampling tends to track and keep frequent items so that it could identify them. It makes an assumption that items it tracked are potential frequent items; because the more frequent the untracked item is in the data stream, the more likely it will be sampled. Sticky Sampling splits data into windows. And as the length of date stream increases, the size of window increases while the sampling probability decreases; therefore, the algorithm will sample and track frequent items more accurately. Every time we reach the window boundary (i.e., the time we adjust sampling rate r), we scan all tracked items; then, we toss an unbiased coin and we decrease the items frequency for each unsuccessful toss, until the coin toss becomes successful or the frequency becomes 0 at which we do not track that item.[1] Because the number of unsuccessful coin tosses follows a geometric distribution, those items with high frequency are more likely to stay.

4.Complete Table 1 with theoretical values (in big-O notation) as a function of algorithm parameters. Provide some justification of these formulas (they may depend on your implementation) and point out any assumptions made. This will be used as a reference to compare with results you obtain in the practical implementation. [1 mark]

	StickySampling	LossyCounting	SpaceSaving
UpdateTime	$O(\epsilon^{-1}log(s^{-1}\delta^{-1}))$	$O(\epsilon^{-1}log(\epsilon))$	$O(s^{-1}logs^{-1})$
Memory[1]	$O(\epsilon^{-1}log(s^{-1}\delta^{-1}))$	$O(\epsilon^{-1}log(\epsilon))$	$O(s^{-1})$
Accuracy	1-ϵ	$\frac{\epsilon N}{f}$	1-ϵ

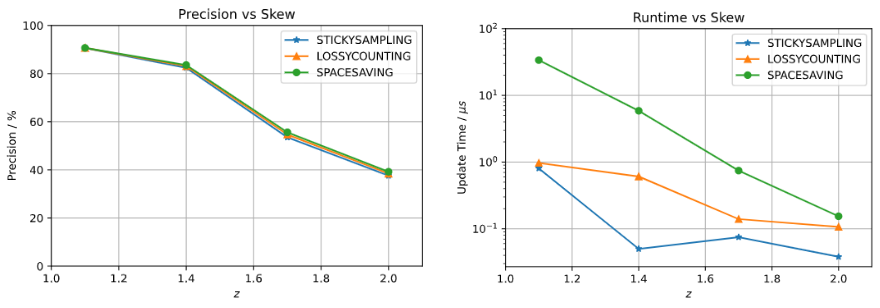
Table 1: Theoretical Performance of the Algorithms

For Lossy Counting, N is the length of stream and f is the true frequency of item. We have not included N in our time and space complexity because it is not a parameter for algorithms that we cover here.

Regarding Space Saving, we used built-in sort method to get the integer with minimum frequency, which needs at most $\frac{n}{2}$, where n is the number of tracked items, temporary storage; so that the theoretical memory usage is 1.5×(the number of tracked items), that is $1.5 \times s^{-1}$. And comparisons in sorting makes majority contribution to update time, and the time complexity of the built-in sort method is $O(nlogn)$, that is $O(s^{-1}logs^{-1})$.

And in terms of Sticky Sampling, the expected update time of each tracked item is $\sum_{n=1}^{\infty} n\frac{1}{2^n} = 2$; so that for all tracked items, the expected update time is 2×(the number of tracked items).

5.Evaluate the impact of different values of z (listed above) on the performance of the three algorithms. Fix s = 0.001 and choose an appropriately large stream length. Plot precision vs skew and runtime vs skew, commenting on your results. [2 marks]



(a) Precision vs Skew (b) Runtime vs Skew

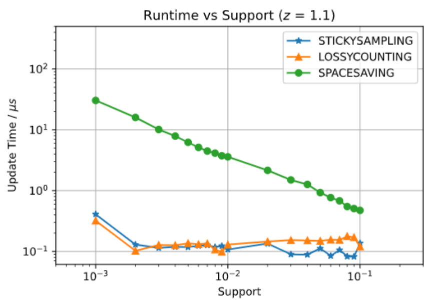
Figure 2: Impact of different values of z on precision and runtime

From the Figure 2a, the precision of all of three algorithms is very similar and decreases with increasing z. And as shown in the Figure 2b, the update time of all of three algorithms decreases as z increases. The elbow points of update time of Sticky Sampling and Lossy Counting are at z=1.4 and z=1.7 respectively, while update time of Space Saving keeps falling as z rises.

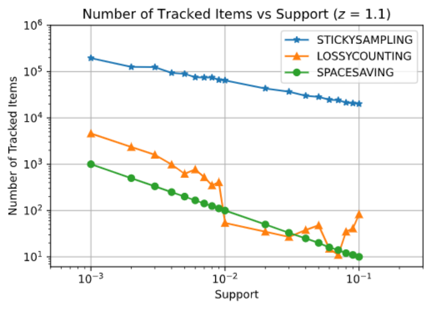
According to figures above, we could conclude that all three algorithms have best performance when z is equal to 1.1.

6.Evaluate the influence of support on memory and runtime for each of the three algorithms. Plot the maximum number of tracked items vs support and also the runtime vs support (minimum s at least 10^{-4}). Using these, obtain a third plot of the

maximum number of tracked items vs runtime for all three algorithms. Point out where the algorithms differ and possible reasons for this. [3 marks]



(a) Runtime vs Support



(b) Number of Tracked Items vs Support

Figure 3: Influence of support on memory and runtime

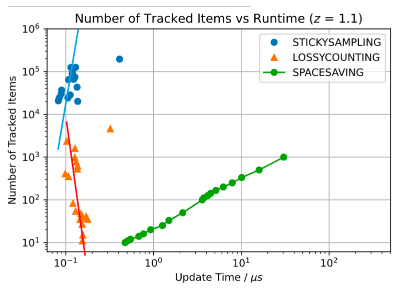


Figure 4: Number of Tracked Items vs Runtime

As in shown in Figure 3a, as the support rises, both of the update time of Sticky Sampling and Lossy Counting fluctuates but has an overall decrease where z is between 10^{-3} to 10^{-1}, while the update time of Space Saving sharply decreases. According to Figure 3b, when the support goes up, all of the maximum number of tracked items of three algorithms decreases. Although there is a fluctuation of Lossy Counting between 10^{-2} and 10^{-1}, it still has an overall reduction in memory usage. The rate of decrease of Sticky Sampling is less than both of those of Lossy Counting and Space Saving. Figure 4 shows that the update time of Space Saving is proportional to its memory usage, while both of the update time of Sticky Sampling and Lossy Counting are not proportional to their tracked items.

Actual results are almost consistent with the theoretical performance in Table 1. In the case where δ is fixed and $\epsilon = \frac{1}{10}s$, the update time of Space Saving is longer and more sensitive to the change of s than the other two, while the support has minimal effect on the number of items tracked by Sticky Sampling. And Sticky Sampling uses at most $\frac{2}{\epsilon}log(s^{-1}\delta^{-1})$ entries, larger than LossyCounting ($\frac{1}{\epsilon}log(\epsilon N)$) and Space Saving ($s^{-1}$). All of the memory usage of three algorithms are proportional to their update time, and the proportionality constant of Space Saving is largest.

7.Plot average relative error vs support for an appropriate choice of parameters for the three algorithms commenting on your results. [2 marks]

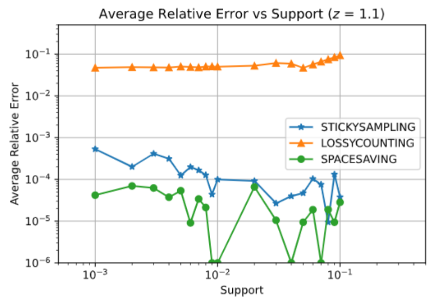


Figure 5: Average Relative Error vs Support

Figure 5 shows the change in ARE (Average Relative Error) with support rising. Here, we calculate the ARE of true heavy hitters and set the value of 0 to 10^{-6} to make sure that those points are plotted appropriately on the logarithmic plot. The ARE of Sticky Sampling fluctuates as well as that of Space Saving, but in total, Space Saving has a lower ARE than Sticky Sampling.

As for Lossy Counting, because We set the length of the data stream to a fixed value of 10^6, the ARE of Lossy Counting is significantly greater than either of Sticky Sampling and Space Saving, and it grows up as support increases. Even though the theoretical ARE of this algorithm, $\frac{\epsilon N}{f}$, seems to be high, the actual ARE can be controlled to a small extent because the data stream is conforming to a Zipf distribution and therefore $\frac{N}{f}$ is small.

8.*Practical considerations*—point out any difference between practice and theory when running the algorithms, and possible explanations for the difference. [1 mark]

According to the update time complexity from Table 1, the update time of Sticky Sampling and Lossy Counting should keep falling as s is going up. However, from Figure 3a, it seems like the update time could not be shorter. We think this happens because the performance of the algorithm is limited by the properties of the programming language and the performance bottleneck of the physical machine.

9.*Conclusion and recommendation*—which would be your recommended algorithm for practical use? Indicate how your answer depends on the application (e.g., speed, accuracy or memory requirements). What additional experiments would you suggest running to gain further insight? [2 marks]

We recommend Space Saving when we are in an extremely memory-constraint scenario, but the trade-off is that its runtime is long when support is small. And if the update time is the most important metric to consider in your scenario, we think either Sticky Sampling or Lossy Counting could be a better choice. If you do not need a high accuracy, just an approximate number of frequent items, e.g. what type of websites people are visiting most frequently these days, then I recommend using Lossy Counting; otherwise, like if you need to count the number of tweets with hashtags of Top-K topics, then I think Sticky Sampling would be more suitable. To gain further size, there is a lot that can be done, such as finding the influence of the size of stream and the frequency distribution on the performance of three algorithms.

References

[1] Manku, G. S., & Motwani, R. (2002, January). Approximate frequency counts over data streams. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases (pp. 346-357). Morgan Kaufmann.