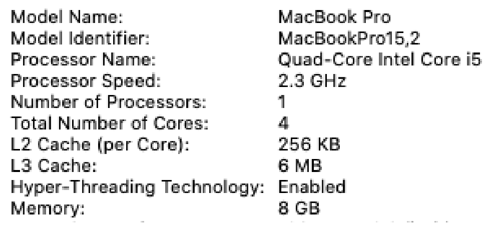
Stream Computing and Applications Assignment 2
Frequent Items in a Data Stream
ChienLin Chen

1. Setup

Implemented all algorithms in Java, complied using Eclipse IDE. The Hardware Overview is showed in Figure 1. Assume epsilon equals support divide by 10.



Model Name:	MacBook Pro
Model Identifier:	MacBookPro15,2
Processor Name:	Quad-Core Intel Core i5
Processor Speed:	2.3 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Hyper-Threading Technology:	Enabled
Memory:	8 GB

Figure 1 Hardware Overview

2. Input Data

Generated power-law distribution using python NumPy zipf function. The choices of the z are 1.1, 1.4, 1.7 and 2.0. The power-law distribution plot and logarithmic plot were plotted. The least most frequent item that has a frequency at least 1% of the total items is showed in the green line. The items appear on the left side of the line all have frequencies of at least 1% of the total items.

a. Power-law Distribution

Figure 2 shows the zipf distribution plots. As the z becomes a bigger number, the distribution approaches the red line which formula is 1 / (i^z) and i is the rank. The number of data generated is 10^8 and not all of the data shows in the diagrams indicated in the titles.

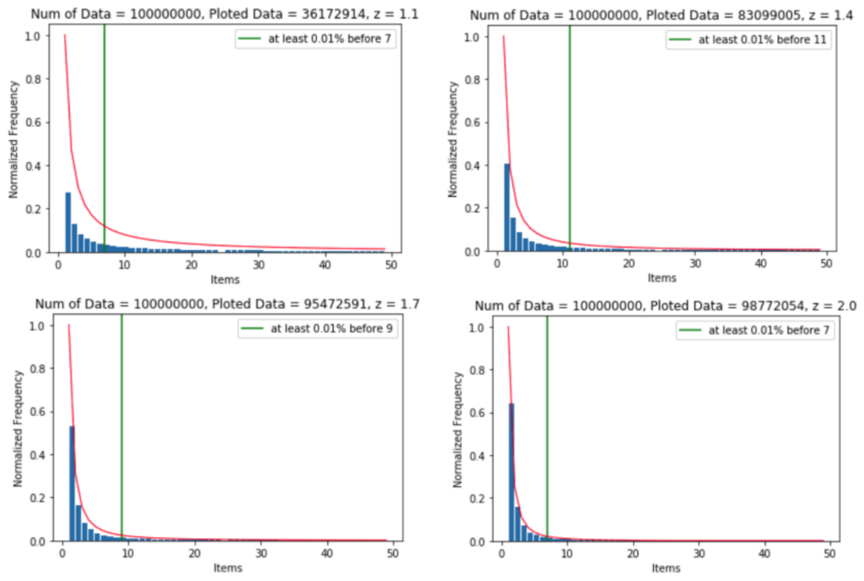


Figure 2 Power-low Distribution Plot

b. Logarithmic Plot

Figure 3 illustrated log-log plots. The blue lines are steeper when the z is larger.

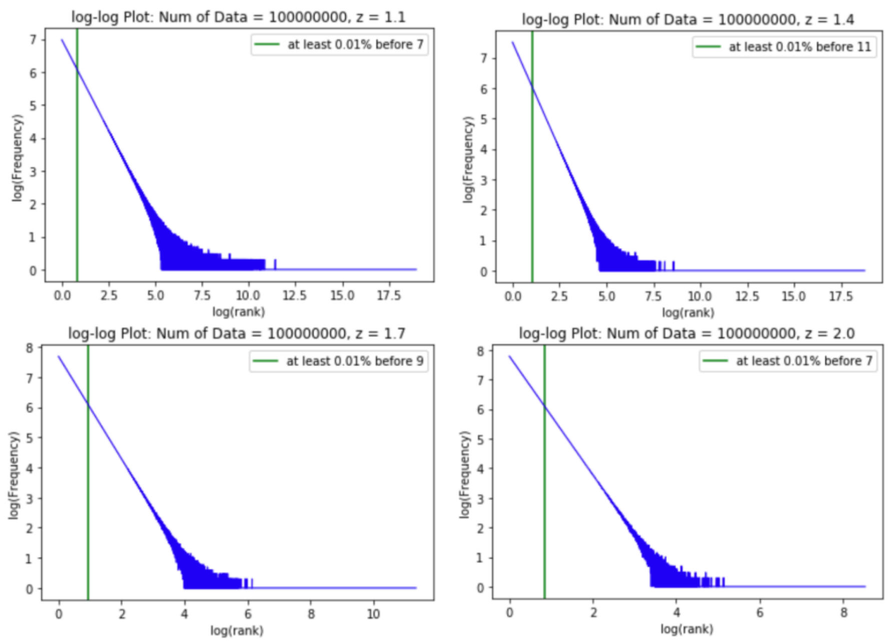


Figure 3 log-log Plot

3. Sticky Sampling Algorithm

The reason Sticky Sampling algorithm works for identifying frequents items is because if any of these frequent items appears in the data structure, then the corresponding frequency will increment. The new element not in the stream will be selected by the sampling rate, so items that appear more often are more likely to be in the data structure. Items that appear latter will have a smaller probability of being selected is because the sampling rate diminishes after processing more items. In addition, for the non-frequent item fortunately selected in the data structure also have chances to be deleted when the sampling rate changes.

4. Theoretical Analysis

	Sticky Sampling	Lossy Counting	Space Saving
Update Time	O(1/ξ log (1/s 1/δ))	O(1)	O(1/s log (1/s))
Memory	O(1/ξ log (1/s 1/δ))	O(1/ξ log (ξN))	O(1/s log (1/s))
Frequency Estimate Error	e^(-ξt)	ξN	Overestimate by Ns
Note	ξ: error rate δ: probability s: support t = 1/ξ log (1/s 1/δ))	ξ: error rate s: support N: current length of the stream Java HashMap has complexity of O(1) for lookup	s: support/ maximum number of items tracked N: current length of the stream

Table 1 Theoretical Performance of the Algorithms

5. Evaluation

In the following section will show the results for running three algorithms on four datasets with data stream length 100,000,000. Precision and Average Relative Error are for heavy hitters only. Assume epsilon equals to 0.1 times support.

a. Fixed Support

In the top right and the bottom left picture in Figure 4 show that Sticky Sampling and Lossy Counting have similar plots. Space Saving need more time for smaller z but the precision for heavy hitter is better than all the other. In contrast, both sticky sampling and lossy counting require a short period of time for any z, but the heavy hitter precision is not as good as Space Saving. From the pictures below, when z is bigger, the performance of three algorithms became similar.

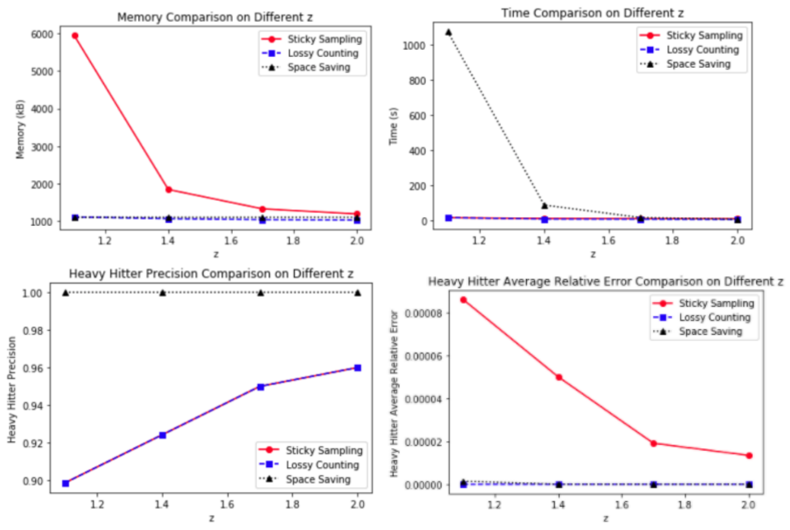


Figure 4 Comparison of z with fixed support = 0.001, epsilon = 0.0001 on Data Stream length 100,000,000

b. Different Values of Support

i. Memory, Time and Max Number of Tracked Items

From Figure 5 shows that memory has some relationship with max number of items tracked by these line patterns. For Sticky Sampling, these red lines are stable for larger z on any support values. It is a bit similar to Lossy Counting, but Lossy Counting require more space when the max number of items tracked increasing. Although the max number of items tracked reach the largest value 350,000 among them for Lossy Counting when support is 10^(-5) and z = 1.1, the memory require is still less than the other. The reasons may be low frequency items fortunately retain in the data structure for Sticky Sampling and Space Saving need space to get the minimum frequency value.

Space Saving needs more time to run for smaller z compared to the other two can finish fast for any z. The reason is that for smaller z, the distribution for heavy hitters are not outstanding; hence, with only limit items can be tracked, it requires time for finding and kicking out the minimum value. From the log-log plot of max number of items tracked versus time, Sticky Sampling has the small variance on the max number of items tracked on given time. In addition, both Sticky Sampling and Lossy Counting not affect by the time on different support values as the bottom right diagram of Figure 5 illustrated that they need around same time for same stream dataset.

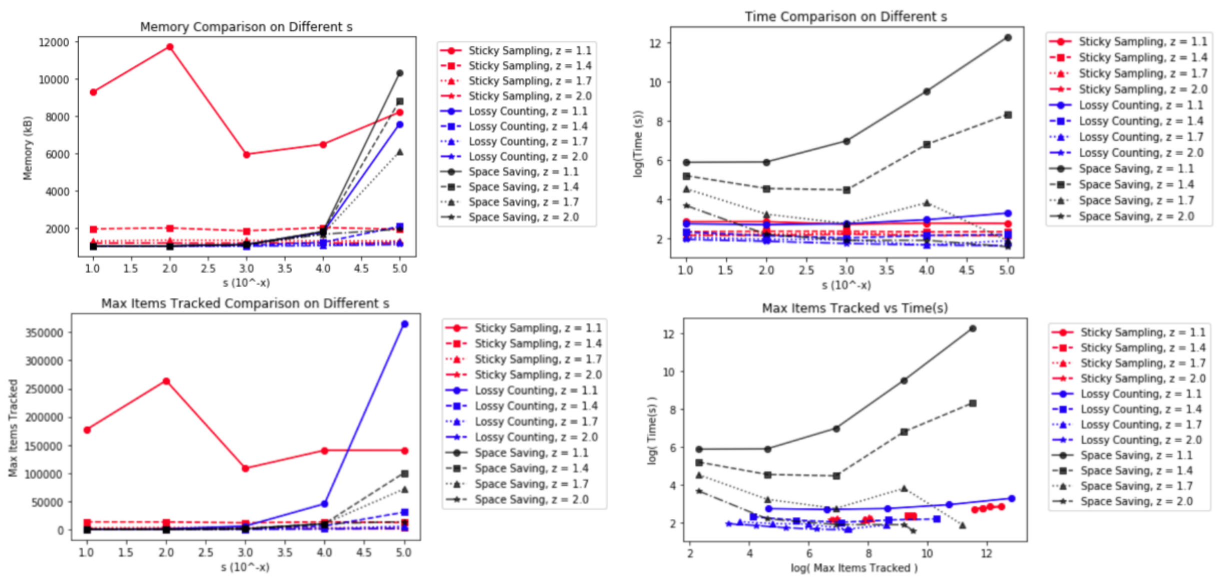


Figure 5 Memory, Time and Max Number of Tracked Items Comparison

ii. Precision and Relative Error

For heavy hitter precision and average relative error, Space Saving algorithm can perform both nearly correctly. Sticky Sampling and Lossy Counting method have similar line patterns in the precision plot but the different in average relative error one. Figure 6 shows Sticky Sampling has larger error than the others and both Lossy Counting and Space Saving have almost the same results. Unlike in the Figure 5 shows memory requirement and number of max items need to be tracked work worser for small z, the average relative error of heavy hitter behaves better. All of them cannot output the correct heavy hitters if the support is set to 0.1 for z = 1.1.

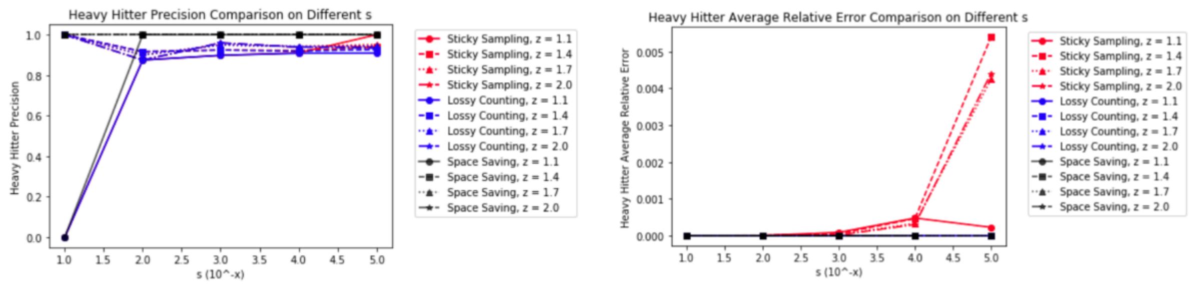


Figure 6 Heavy Hitter Precision and Relative Error Comparison

6. Practical Consideration

For frequency estimate error and the assumption that support is 10 times large than epsilon, both Lossy Counting and Space Saving produce similar results in practical while in theory it shows there are 10 times difference. In addition, in Table 1 shows the worst-case update time for Space Saving is shorter than Sticky Sampling; however, in the implementation, Sticky Sampling runs all input data faster. Similarly, in memory comparison, Space Saving produce smaller value in theory but large in practical. The distribution of the dataset also has an influence in the practical results.

7. Conclusion and Recommendation

In conclusion, for memory aspects, I will recommend Lossy Counting. For time consideration, Sticky Sampling works better than the others. Regarding to high heavy hitter precision and low heavy hitter average relative error, Space Saving can perform both well. All of these three algorithms can identify heavy hitters correctly, require a small memory and run in a short time on larger z and smaller support values.

I will recommend running more z for the input stream data because the distribution affect these algorithms a lot as discussion in the previous sections.