Harshal Harish Shah
Student Id - 1020849

Stream Computing and Applications
Assignment B Report

INTRODUCTION :

"Sampling, statisticians have told us, is a much more effective way of getting a good census"

- *By Rob Lowe*

Data in the modern world is very diversified and has a lot of features to work on. This data-driven world has data large quantum of data that can't be processed together. This is where sampling comes into play. The element or the group of elements, known as samples which represent entire data based on the characteristics and other important factors. Sampling effectively helps us to determine the type of dataset along with its varied features such as its frequency distribution, the type of data it holds, the distribution it follows, etc. The project of "Sampling" and its variants along with Sparse recovery systems helps us to apply theoretical knowledge in the practical world.

In Streaming data processing, there is a concept of "L0 sampling" which is used effectively to sample one item from the entire dataset which represents the qualities of that group. In this report, we are going to focus on L0 sampling along with its primitives. In L0 sampling there is an equal chance of each item to be sampled irrespective of the number of times it occurs in the stream. The goal of the L0 sampler is to uniformly sample an item from the large dataset which assigns some weights to the elements. The output of the sampling varies only by the overall frequency/weights of the item and not by the number of times it appears.(Cormode&Firmani, 2013)

There are various applications of L0 sampling within computational geometry, graph algorithms, and data analysis. When we want to analyze the data over a large dataset, its beneficial to use sampling and analyze the properties of the sample to understand the data. Also, in graph algorithms, sampling helps us to understand the connectivity amongst its components thereby depicting the properties of the data.

THEORY :

The L0 sampler which is used to sample an item uniformly from a large dataset made improvements in its working. To overcome the issue of items with frequency zero and to maintain its information, the L0 sampler made the use of hash functions. Hashing is used to compare and store the element at a particular level whenever it appears in the stream so that it can be referred in the future. A k wise independent hash function is used which chooses a hash function randomly from the entire hash family. At each level, a data structure is created that summarizes the elements and its frequencies. This data structure is created to understand various approaches to L0 sampling and to solve the problem of sparse recovery. If there are very fewer items with non-zero weights at a particular level which are determined after using a hash function, then the full set can be recovered, and one item can be sampled. (Cormode&Firmani, 2013)

1 sparse recovery systems or algorithms are used when we have one item with non-zero weight at a particular level. The main goal behind achieving the s-sparse or the 1-sparse recovery systems is to retrieve a set of values from a vector that represents the original vector of the entire dataset. 1 sparse recover one vector with one non-zero frequency along with its index with a space complexity of O(log n + log l1) which handles updates, queries and the functioning of the 1 sparse algorithm. The updates in the 1 sparse recovery can be done in O(log n) time. S-sparse performs the same operations as 1 sparse multiple times based on the number of rows viz (d, no of hash functions) and the number of columns viz (2s) the matrix has. This methodology is useful in real-time as we have data in huge numbers! So, a good sparse vector can represent our input vector in a very good manner. (Cormode&Firmani, 2013)

1 Sparse recovery systems can be worked out with the help of Ganguly test and Fingerprint test. Since Ganguly test can handle turnstile stream only , it doesn't generate accurate answers in case of negative

1-Sparse	1-Sparse	1-Sparse	1-Sparse
1-Sparse	1-Sparse	1-Sparse	1-Sparse
1-Sparse	1-Sparse	1-Sparse	1-Sparse
1-Sparse	1-Sparse	1-Sparse	1-Sparse

Table 1: Exact s-sparse with exact 1-sparse

(item, frequency)

frequencies. As the last primitive, the Dynamic L0 sampler also focuses on the deletions of items. It works well for a general stream on the principle of Insert L0 sampler itself but with some additional functionalities. Dynamic L0 sampler after processing returns the final candidate which corresponds to the minimum value. Note that, as and when the size of dataset increases the accuracy and correctness of the algorithms data structures increases.

	L0 Sampling – insert only	L0 Sampling– general	L0 Sampling – turnstile
Update Time	O(log n)	O(log n)	O(log n)
Query Time	O(1)	O(1)	O(1)

Table 2: Complexity Analysis

	L0 Sampling – insert only	L0 Sampling– general	L0 Sampling – turnstile	1 Sparse Recovery	S Sparse recovery
Memory	O(log n)	O(log n)	O(log n)	O(log n + log u +log p)	O(s log n log(s/δ))

Table 3: Space Complexity Analysis (Cormode&Firmani, 2013)

IMPLEMENTATION :

Some of the key points in the implementation are:

1. The programming language used is Java which is usually used for large datasets.
2. Although C/C++ is better when we look for memory optimization, Java is easier to understand and implement as it has a lot of libraries.
3. In the implementation, a MainFile.java is created which is the main driver file that has the dataset file. All the objects are created, and all functions called with the help of these objects.
4. HashMap's and Array lists have been used for storing and processing the data. Also, the Random number generator of the Uniform Distribution has been used for generating the ID's of the data.
5. ID's are generated randomly for both the streams, turnstile and general and the appropriate functions are performed.
6. In the implementation, the HashMap can handle any kind of data. For example, in the current scenario, the key and values are both integers. Hashmap can have only the unique ID's thereby adding up the frequency of the item which appears for the second time.

7. A random hash function was selected from the hash family as the main objective behind returning a key was "equal probability" which does not depend on the number of times the item appeared in the stream.
8. We randomly selected h[n]->h[n^3] from the hash family and performed the sampling procedure based on this function.
9. Implementation is done for Ganguly and Fingerprint test both. (Cormode&Firmani, 2013)
10. The other java files except for the mainFile are InsertL0, OneSparse, SSparse, Dynamic L0 and the Hash.
11. In the k sparse I have chosen the value of s to be 4 so that I get a wide range (8 columns) and I can get some better results.
12. In the 1 Sparse Fingerprint test, the value of p is taken as a very large prime number and the value of q is chosen randomly which is never larger than p.
13. One more file is for the OneSparseTesting file which has all three kinds of datasets manually added by us to check the accurate working.
14. The calculate function, add function computes the respective value for the L0 sampling, One Sparse, SSparse and Dynamic L0. The out function returns the respective key and the vector which does the sparse recovery.
15. Constructors are used in all the programs to initialize the values.
16. The memory consumption for the program is very minimal and therefore space efficient.
17. Also, the data structures used are very compact and responsive to fast queries thereby taking less time for answering the query functions.
18. All the algorithms are implemented as per the lecture slides and the reference paper and thereby not mentioned in detail in this report. (Cormode&Firmani, 2013)

EXPERIMENTAL SETUP :

1. In my implementation, I have used the "The Moby Dick dataset" (Gillespie, 2019) which is being retrieved from the PoweRlaw package. These values are integers and are used as the frequency counts in the experiments.
2. I had to install the package in R and then extract the dataset for processing it in the java implementations (all datasets represent real world)
3. This file is used when we consider the Insert only as every element present is positive.
4. Another dataset is used in the general case and the values of the frequency count are generated from the uniform distribution and stored in a separate file.
5. The ID's of the corresponding frequencies are generated randomly from the uniform distribution and stored as well.
6. The ID's generated are stored in a file so that the experiment can be replicated whenever needed.
7. The java.util.concurrent.TimeUnit library is used to evaluate the time taken to modify and test op erations, as well as the overall program execution. To get the time in milliseconds, its divided by the appropriate unit.
8. The functions getRuntime.totalMemory() and getRuntime.freeMemory() are used to evaluate the memory used by the operations. To get the total amount of memory used in Bytes, we subtract the free storage from the total memory. This is done at the program start and end.
9. The implementation works well on insert only streams, general streams, and turnstile streams as well.
10. The datasets used has a single column which we have considered as the frequency count for the particular element. The first dataset has approximate 20k rows and approximate 50k rows.

Harshal Harish Shah
Student Id - 1020849

11. Further, histograms are plotted to check whether the values after sampling are uniformly distributed or not. Also, chi-square test is performed to check the uniformity.

RESULTS & DISCUSSION :

After the successful implementation of the L0 sampler and its primitives along with various sparse recovery systems. The results are based on the accuracy, time taken for execution and the memory used and some general observations.

1. As mentioned in the notes, the fingerprint test for general stream sometime succeeds even if there is more than one item with non-zero frequency (tried and figured out).
2. The program is generalized and can be tested and replicated on any kind of datasets.
3. The max integer value used in the Hash and the large prime number p, randomly chosen q would be more beneficial and the results would be much better if the data is really of the size of the real world (for example of size approx.. 100,000,000)
4. For the sampling part, the histogram helped me to verify whether the values after sampling followed a uniform distribution.
   The figures show that the values after sampling are uniformly distributed.

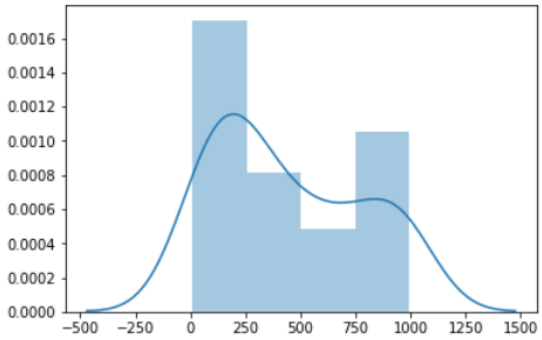

Figure 1: L0 Sampling Uniformity -1

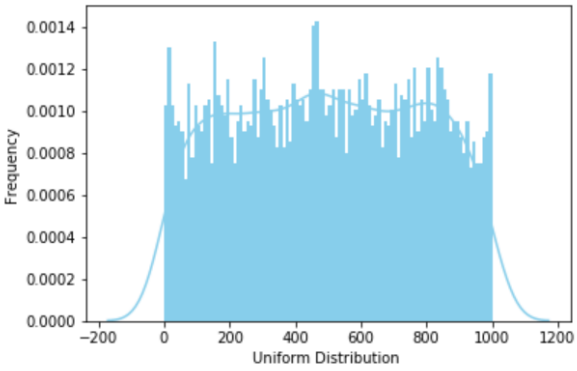

Figure 2: L0 Sampling Uniformity -2

5. Also, chi square test was used to check the uniformity. The p value is really very less and always weighted towards 0. That is how we determine that the values are uniformly distributed.

6. Further the Testing was done manually for the one sparse testing to cover all its aspects because in the datasets used its very rare to get exact One Sparse output. The datasets and the outputs are as follows:
   Dataset 1:
   int[] numbers = {1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
   int[] frequency = {-5,20,-30,-20,25,-5,20,30,-20,25,10,-40,0,40,-50};

   Dataset 2:
   int[] numbers1 = {1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
   int[] frequency1 = {-5,20,-30,-20,25,-5,20,30,-20,25,10,40,0,40,-50};

   Dataset 3:
   int[] numbers2 = {1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
   int[] frequency2 = {-5,20,-30,-20,25,5,20,30,20,25,10,40,0,40,-50};

   Output:

```
run:
ONE SPARSE OUTPUT: All items with frequency zero
ONE SPARSE OUTPUT: Exactly One Sparse
{2=80}
ONE SPARSE OUTPUT: More than one item with non zero frequency
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 3: One Sparse Manual Testing Output

7. The total time take for execution is more for L0 general, followed by turnstile and the least time is taken for insert. This is due to the operations performed for the positive and negative values and retention of 0 frequency as well. There is very less time difference between all the three functionalities. This has been tired and implemented in the One Sparse Testing file present.

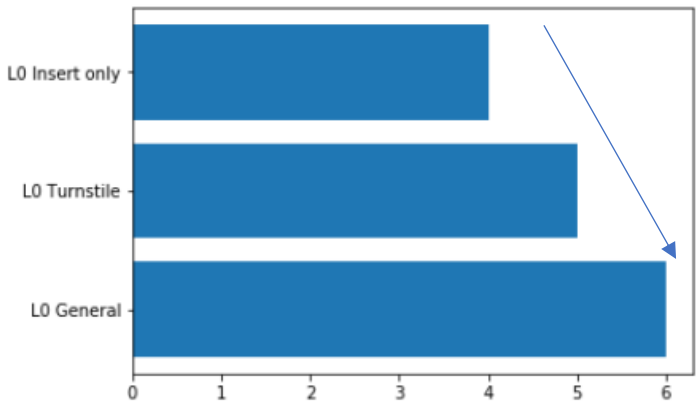

Figure 4: Time Taken for Execution

The above can be explained with the help of a pie chart as well.

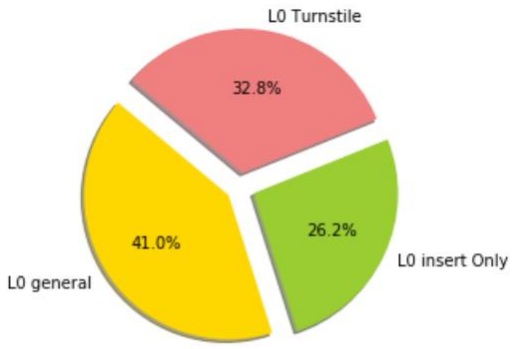

Figure 5: Time taken for Execution (Approximate Comparison)

8. Based on observation, using HashMap's for merging the respective frequencies for their corresponding ID's is a bad practice as it consumes more memory. The solution to this is to parse the stream and as and when element arrives add it to the corresponding key instead of doing it beforehand.
9. The accuracy of the L0 sampler increases as and when the size of the dataset increases. The accuracy is measure using the standard and the maximum deviation.

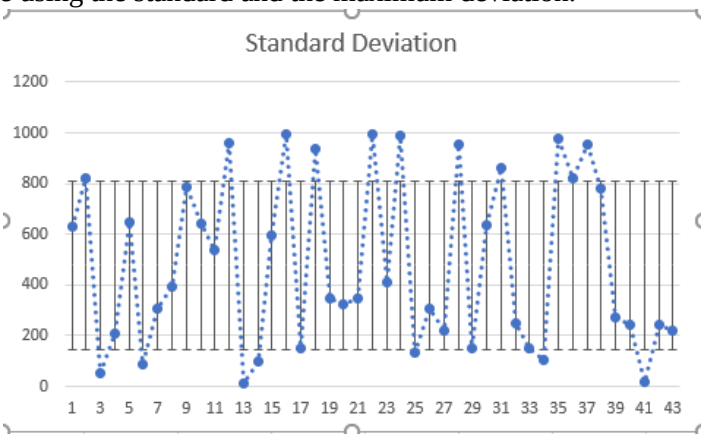

Figure 6: Standard Deviation

10. The values outside the lines and not considered to be normal and in this case there are less than 50% such values. This is indeed for a small dataset. The accuracy reaches up to 90% or more when we consider big datasets.
11. The results obtained follow the facts discussed in the theory and thereby adhere to all the discussions made.
12. Based on the discussions made and the complexity of the algorithms , the L0 sampler achieves the best output when the hash function is used accurately, and the data is really huge.

REFERENCES:

1. The PoweRlaw Package(2019). The PoweRlaw package. Retrieved from: https://cran.rstudio.com/web/packages/poweRlaw/vignettes/b_powerlaw_examples.pdf
2. Cormode & Firmani (July 2013). A Unifying Framework for L0-Sampling Algorithms. Published online via the Springer Conference. Retrieved from http://dimacs.rutgers.edu/~graham/pubs/papers/l0journal.pdf