COMP90056 Stream Computing and Applications

Assignment B

Part 2: l_0 sampling and sparse recovery

Shuai Wang (830166)

shuaiw6@student.unimelb.edu.au

1 Introduction

The aim of this report is to develop a quality Java program for l_0-sampling in dynamic stream, and demonstrate the efficacy of the implementation.

2 Datasets

To evaluate the performance of my l_0-sampler, some datasets in Table 1 are used.

Dataset	Size(MB)	Domain	Description
D1	116.7	100000	Each item in [1,10000] with different value in [1,100000] occurs many times [1, 100]
dinner	0.001839	40	Each item in [1, 40] occurs many times with positive frequency
dinner2	0.001861	41	Each item in [1, 41] occurs many times with negative frequency
1SZero	2.1	100000	Each item in [1, 100000] occurs many times, the frequency of each item is zero
1SOne	2.1	100000	Each item in [1, 100000] occurs many times, the frequency of each item is zero. Plus a pair <1000001,1>
1SMore	2	100000	Each item in [1, 100000] occurs many times, the frequency of each item might be positive, zero or negative
SSZero	2.5	100000	Each item in [1, 100000] occurs many times, the frequency of each item is zero
SSRecover	2.5	100000	There are 100 items with non-zero frequency. Other items with zero frequency
SSMore	2.5	100000	There are 150 items with non-zero frequency
D100	0.012	100	There are 100 items. The frequency of each item might be negative, zero or positive. Each item occurs many time with different frequency.

***Table 1**: Testing datasets*

Some challenges I faced when making datasets: when the size of dataset is large, it will take long time to read numbers in files and sample an item from it. For instance, the size of dataset D1 is 116.7 MB, it takes 9.4 seconds to read all numbers and 4.1 seconds to sample one item by *InsertL0Sampler*. In total, 13.5 seconds are taken to do once sampling. If we want to determine whether our l_0-sampler is sampling uniformly, we must sample many times and then check if each item has been sampled nearly number of times. The greater the number of trials, the more accurate our result. For example, do 10000 times trials. But it would take 37.5 hours. This is not feasible for my computer. Therefore, I make smaller size datasets but do more trials.

3 Evaluation Metrics

The evaluation metrics to test our l_0-sampler are Chi-Squared Test, MAE, runtime and memory space.

Chi-Squared Test is used to check if a sampler are uniformly sampling an item from a dynamic stream. The value of the Chi-Squared Test statistic [1] is

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{1}$$

The smaller the value is, the better the sampler.

MAE is mean absolute error. MAE can be computed with

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |T_i - \mu_i| \tag{2}$$

n is the number of distinct items. T_i represents how many times the item i has been sampled. μ_i is the ideal mean times the item i has been sampled. MAE can show that how far away our sampler to perfect sampler in average.

Runtime is a useful metric to evaluate the speed of sampling. The time used to read items from files are not taken into account.

4 Design Choices

4.1 l_0-Sampling in an Insert-Only Stream

In this part, we are going to build a simple l_0-sampler to uniformly sample an item from insert-only stream, in which each item has positive count. In my implementation, I randomly select a hash function from k-wise independent hash function family:

$$h(x) = a_0 + a_1 \bullet x^1 + a_2 \bullet x^2 + \cdots + a_{k-1} \bullet x^{k-1} \; mod \; r \tag{3}$$

Here, $a_i \in [0, r-1]$ for $\forall i \in [0, k-1]$, that is, a_i is randomly selected from 0 to $r-1$. There is a parameter k, which should be greater than $s/2$ (s is the universe of the stream). The parameter r should be a large prime number. In order to avoid collision, in my implementation $r = 1073741789$. One problem I faced when sampling is that this hash function is time-consuming. The greater the parameter k, the longer the time of sampling due to lots of exponential computation. Compared with a hash function randomly selected from pairwise independent hash function family, for instance,

$$h(x) = a \bullet x + b \; mod \; p \; mod \; r \tag{4}$$

This hash function is much faster than the previous one. However, according to the paper by Firmani and Cormode [2], a hash function from k-wise independent hash function family can have higher accuracy of sampling item. In other words, it selects an item in stream more uniformly.

We compare two l_0-samplers with different hash functions on dataset *dinner.txt* given by teaching staff. For both, we do 10000 trials. The results of sampling are showed in Figure 1. We can observe that the l_0-sampler using a hash function from k-wise independent hash family samples item more uniformly. The sampled times of each item is around 250. But the other l_0-sampler fluctuated significantly.

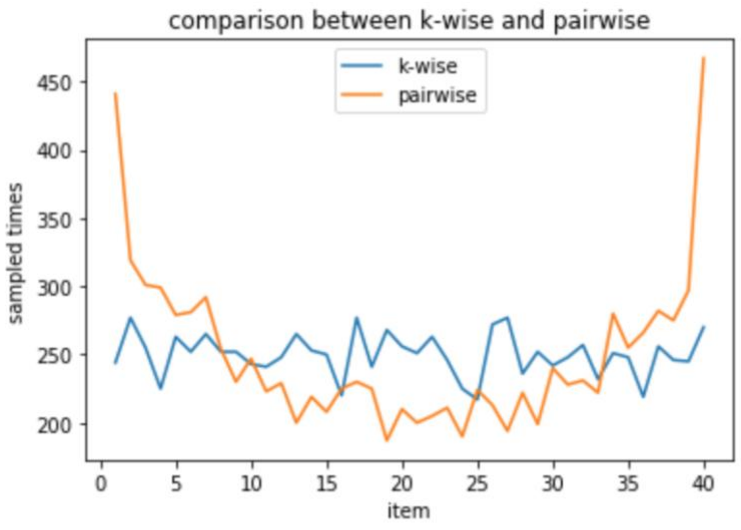

***Figure 1**. Results of l_0-samplers with different hash functions.*

On the other hand, the latter l_0-sampler just takes 2.01 second to finish 10000 trials, while the former takes 6 seconds (when parameter k=20). The χ^2 and MAE of the l_0-sampler with k-wise independent hash function is smaller than pairwise, which proves that it can sample item more uniformly than using pairwise.

Hash Function	χ^2	MAE	Runtime(sec)
k-wise	34.1	11.7	6
pairwise	565.7	41.9	2.01

***Table 2**. Chi-squared, MAE and runtime for l_0-samplers with different hash functions.*

4.2 One-Sparse Recovery

In this part, we aim to build a one-sparse recovery system, which is able to determine whether a stream is one-sparse, in other words, only one item has non-zero frequency in total, and recover the item. We are using a variant Ganguly's test [3], which can deal with items with negative frequencies. We test our implementation of one-sparse recovery system on three datasets: 1SZero, 1SRecover and 1SMore. The outputs for the datasets are showed in Table 3. From the results, we can conclude that our implementation performs correctly. It can accurately tell if a stream is one-sparse and recover the one-sparse vector.

Dataset	Output	Runtime(sec)
1SZero	zero	3.6
1SOne	<1000001,1>	3.3
1SMore	more	5.1

***Table 3**. The outputs of oneSparseTest function on 1SZero, 1SOne and 1SMore datasets.*

4.3 S-Sparse Recovery

This part is to build a s-sparse recovery system, which can determine whether a stream has exactly s items with non-zero frequency and recover those items from the stream. In my implementation, I use a two-dimensional array with r rows and $2 \bullet s$ columns, where each cell contains an instance of one-sparse recovery system. There are two parameters: $r \in [log\frac{s}{\delta}]$ and s is the sparsity size. The greater the parameter r, the s-parse recovery system is more likely to recover the s-sparse vector correctly. In my implementation, set $r = 4$. It is enough to make sure correct recovery with high probability and save space due to less number of one-sparse recovery cells.

The main problem is how to recover a sparse vector by using this array. In my implementation, count how many cells that return a one-sparse in each row. Choose the row with maximum number of one-sparse count. In this row, extract items from the cells that say one-sparse. Those items consists of the recovery.

Moreover, if the number of items recovered is zero, then the algorithm regards the stream as zero-sparse; if it is s, then the algorithm thinks this stream is s-sparse; Otherwise, the stream is not s-sparse.

The s-sparse recovery system has been tested on three datasets: SSZero, SSRecover and SSMore. The results are showed in Table 4. Base on the results, we can conclude that our implementation performs correctly.

Dataset	Output	Time(sec)
SSZero	zero	43.85
SSRecover	100 pairs (correct)	41.8
SSMore	more	41.7

***Table 4**. The outputs of sparseRecTest function on SSZero, SSRecover and SSMore datasets.*

4.4 l_0-sampling in dynamic/general stream

This part is to build a l_0-sampler which can uniformly sample an item from a stream of items, such items can have negative frequencies.

To have higher probability of successfully sampling an item from a stream, we need to tune the parameter of $sparsity = 4 \bullet log\ ^1/_\delta$. For 90% success probability of sampling, we set $sparsity = 13$. If this parameter is small, the l_0-sampler is likely to return $null$, In other words, it fails to sample. Figure 2 shows the failure rates on different $sparsity$ settings on *dinner2* dataset. In fact, when the sparsity increases to 11, there is no $null$ sample returned. To make sure, we can successfully sample an item without $null$, we use $sparsity = 13$.

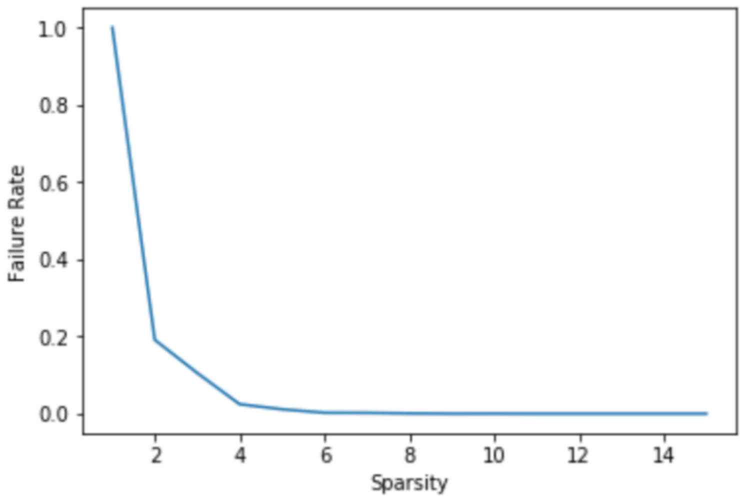

Figure 2. *Failure rate and sparsity relationship*

We also use a hash function randomly chosen from k-wise independent hash family. For *dinner2* dataset, we set $k = \frac{sparsity}{2} + 1 = 7$ and the range is set to n^3 where n is the size of universe. $log_2 n$ s-sparse recovery systems are used.

We test our l_0-sampler on datasets *dinner2* and D100. The results are showed in Figure 3 and 4. We observe that the sampled times of each item is around 250 in Figure 3.

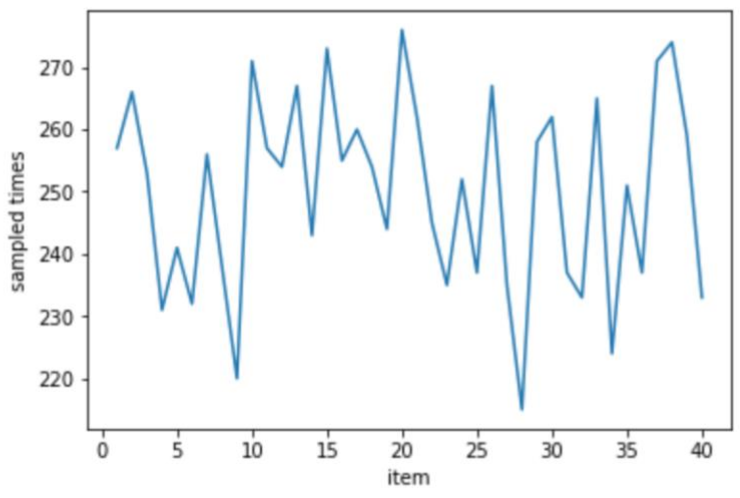

Figure 3. *Results of dinner2 dataset*

In Figure 4, we can observe that some items have zero sampled times. This is because the frequencies of these items is zero. Our l_0-sampler ignores those items.

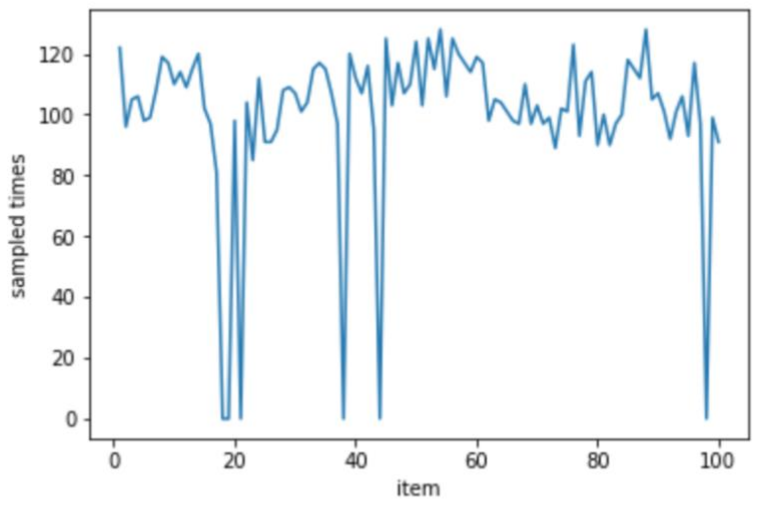

Figure 4. *Results of D100 dataset*

The Chi-Squared, MAE and runtime have been showed in Table 5. Our l_0-sampler has small Chi-Squared value and MAE, which means the item in stream has been sample nearly uniformly. However, the runtime is high due to k-wise independent hash function.

Dataset	χ^2	**MAE**	**Runtime(sec)**
dinner2	39.3	13.5	10.8
D100	96.5	8.7	64.8

Table 5. *Evaluation metrics of the l_0-sampler on dinner2 and D100 datasets.*

5 Conclusion

We first discussed about a simple l_0-sampler which can uniformly sample an item from a stream a items with non-zero frequency. A random hash function is used to sample item. We compared two types of hash function, one from pairwise independent hash family while another from k-wise independent hash family. We observed that l_0-sampler with the former hash function can sample item more uniformly while take longer time. One-sparse and s-sparse recovery systems are introduced and tested on multiple datasets. From the results, we found our implementations performed correctly. At last, we combined one-sparse and s-sparse recovery systems, and the simple l_0-sampler to construct a robust l_0-sampler which is able to uniformly sample item from a stream which allows negative frequencies. From the results of testing, the l_0-sampler can successfully sample an item with high probability. We also talked about the parameter $sparsity$ which has impact on the performance of the l_0-sampler.

6 References

[1] https://en.wikipedia.org/wiki/Chi-squared_test.

[2] Cormode, G., & Firmani, D. (2014). A unifying framework for ℓ 0-sampling algorithms. Distributed and Parallel Databases, 32(3), 315-335.

[3] Ganguly, S. (2007). Counting distinct items over update streams. Theoretical Computer Science, 378(3), 211-222.