### Homework 2

### Streaming algorithm

### & Locality Sensitive Hashing

Name: Eduardo Wang Zheng

E-mail: eduardo@sjtu.edu.cn

1. Experiments
   1. Implement the DGIM algorithm to count the number of 1-bits in stream
      1. Algorithm

Main function:

1) Initialize the buckets. The buckets are represented in a **dictionary** where the **key** is **bucket size** (the number of 1-bits) and the **value** isa list of **timestamp.** In this way, I merge the bucket with same size but different timestamp for convenience.

2) Read the stream bit by bit.

3) If there is no more input, output the number of 1-bits in the current window and break the loop.

4) Update **timestamp**

5) Traverse the buckets and remove record which is out of the window

6) If the new coming bit is 1, add current **timestamp** to bucket 1 and update buckets.

7) Update index

8) If the index reaches **windowsize,** output the number of 1-bits in the current window.

9) Move to step (2)

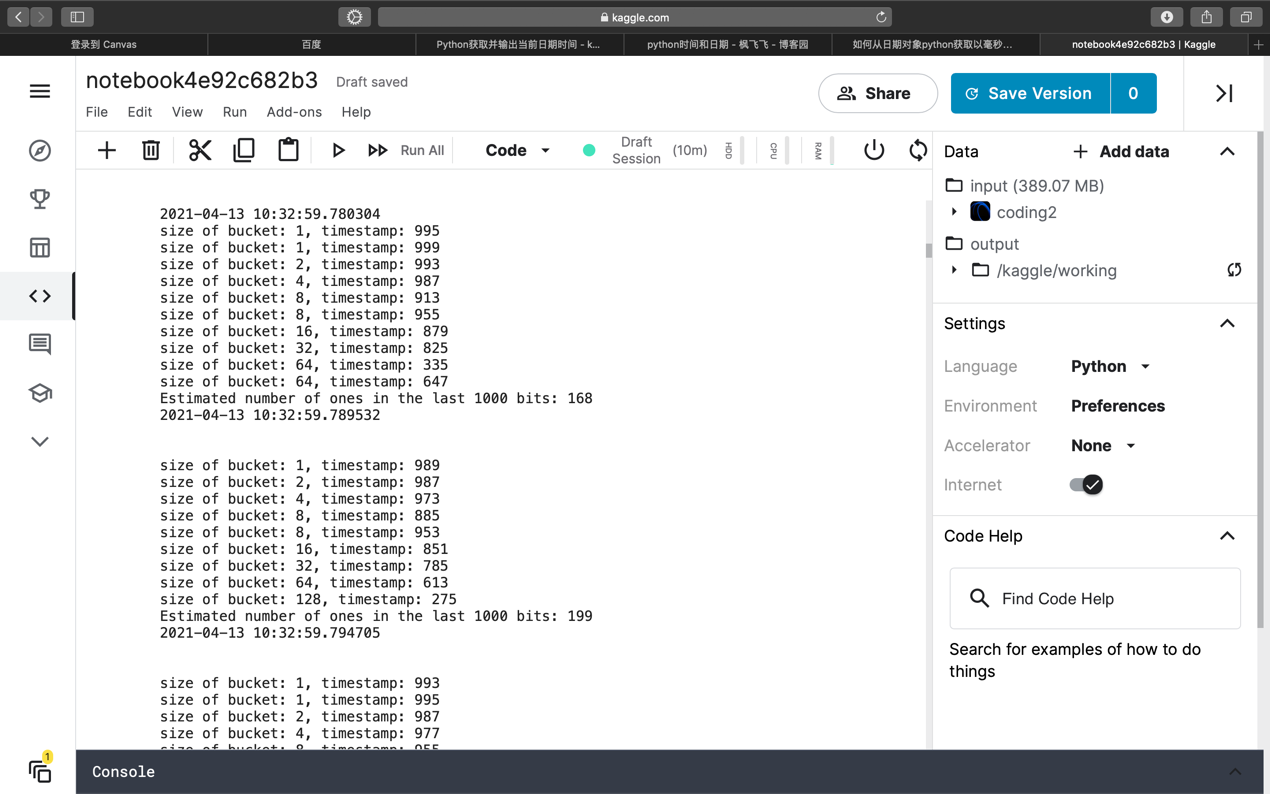
Function **UpdateContainer:**

1) If there are now three buckets of size 1, combine the oldest two into a bucket of size 2

2) If there are now three buckets of size 2, combine the oldest two into a bucket of size 4

3) And so on...

* + 1. Evaluation



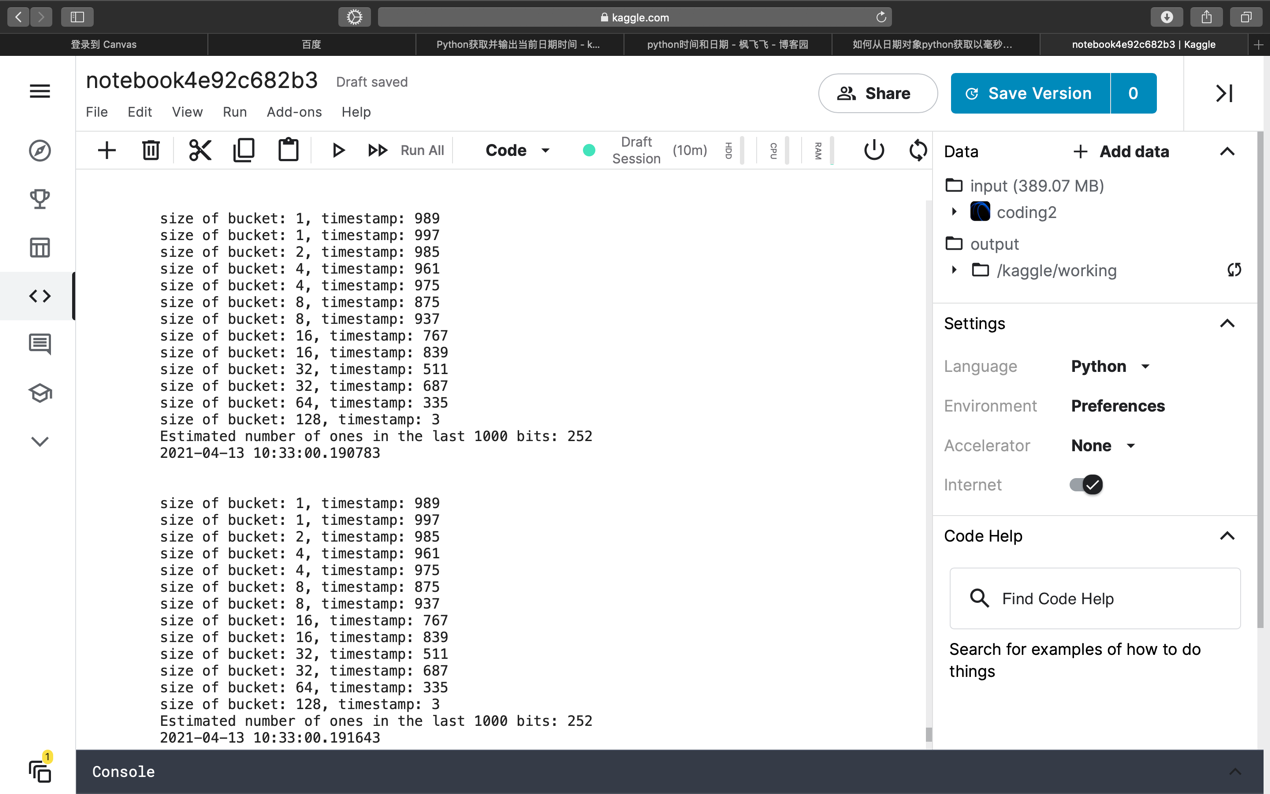


Figure 1 DGIM algorithm

* 1. Traditional way to count the number of 1-bits in stream
     1. Algorithm

1) Read the stream bit by bit.

2) If there is no more input, output the number of 1-bits in the current window and break the loop.

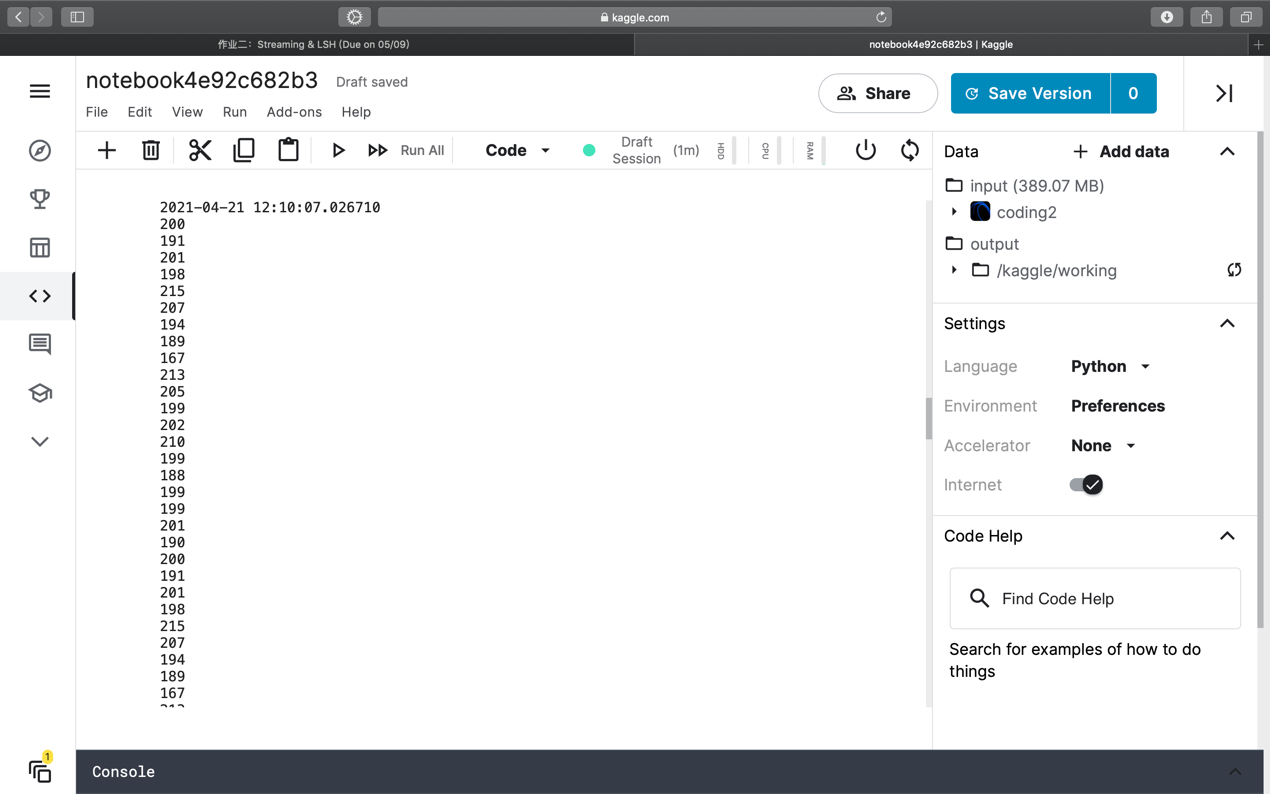
3) Update index

4) If the new coming bit is 1, add 1 to the counter

5) If the index reaches **windowsize,** output the number of 1-bits in the current window and clear the counter

6) Move to step (1)

* + 1. Evaluation



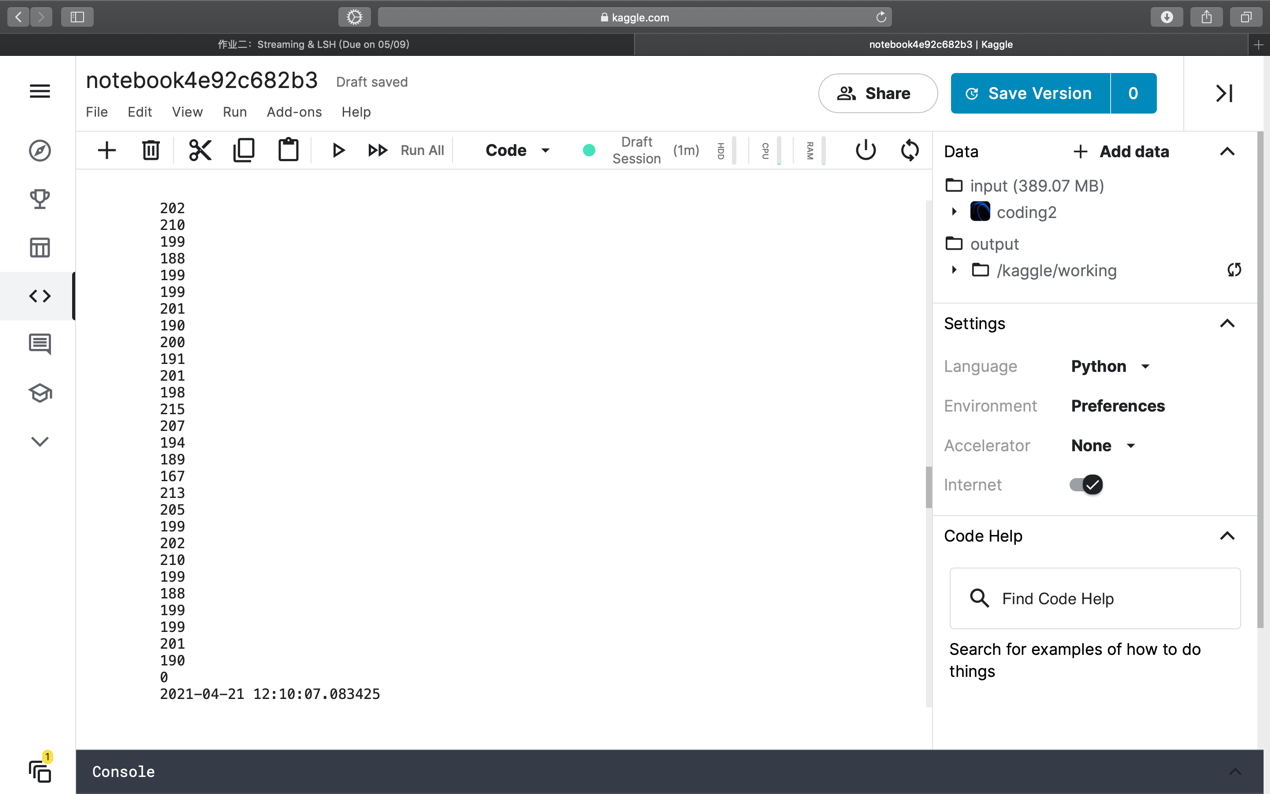


Figure 2 Traditional way

* 1. Comparison
     1. Running time

As figure 1 and figure 2 show:

The running time of DGIM algorithm is 0.4 s.

The running time of Traditional way is 0.06 s which is smaller than the DGIM algorithm.

* + 1. Space usage

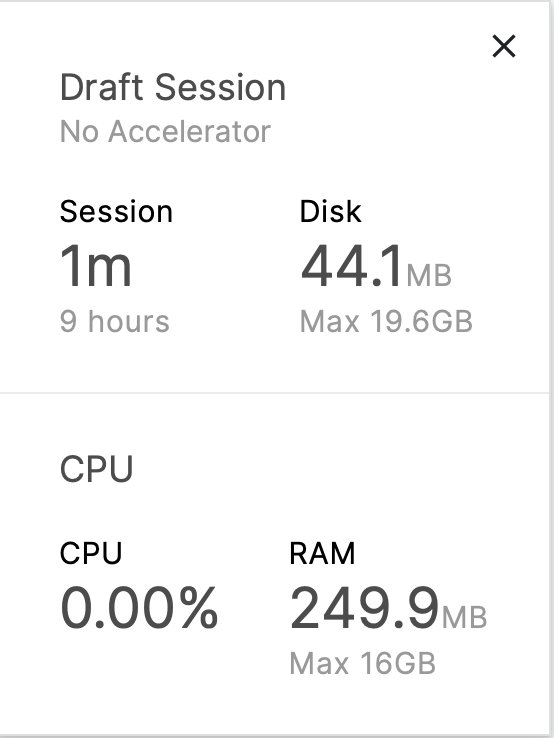


Figure 3 Space usage of DGIM algorithm

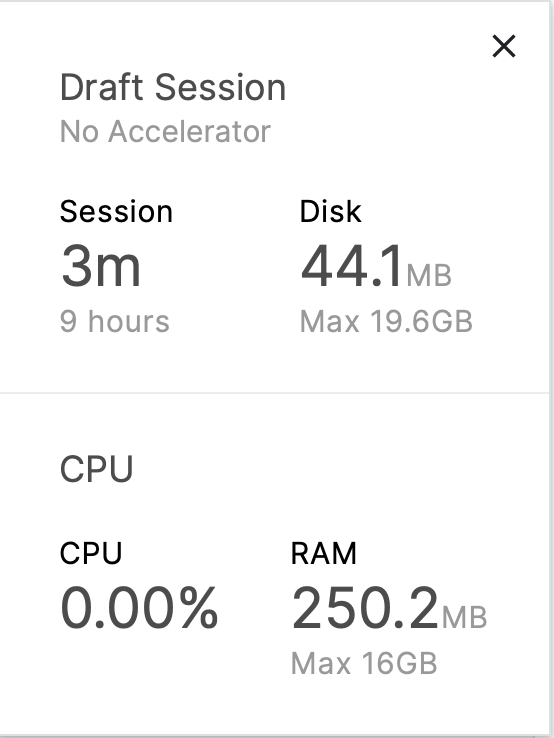


Figure 4 Space usage of traditional way

From figure 3 and 4, we can find that space usage of two algorithm is almost the same.

* + 1. Accuracy



Figure 5 Estimated value by DGIM algorithm



Figure 6 Real value

From figure 5 and 6, we can find that the relative error produced by DGIM algorithm is 8.82% in this case.

1.4 Use minhash algorithm to create signature of each document and find the 30 most similar document ids for document 0 (the one with id '0') under Jaccard similarity

1.4.1 Algorithm

Main function:

1) Load docs\_for\_lsh.csv

2) Create MinHash objects

3) Use MinHash algorithm (MinHash objects) to create signature of each document

4) Create MinHashLSH object

5) Insert MinHash objects into MinHashLSH object to generate candidate pairs

6) Query on candidate pairs find the 30 most similar document ids for document 0

Parameters :

1) Length of signature (number of distinct minhash functions) n=32 (**num\_perm=32)**

2) Number of bands that divide the signature matrix b (using default value)

3) Jaccard distance threshold setting J = 0.8 (**threshold=0.8)**

1.4.2 Evaluation

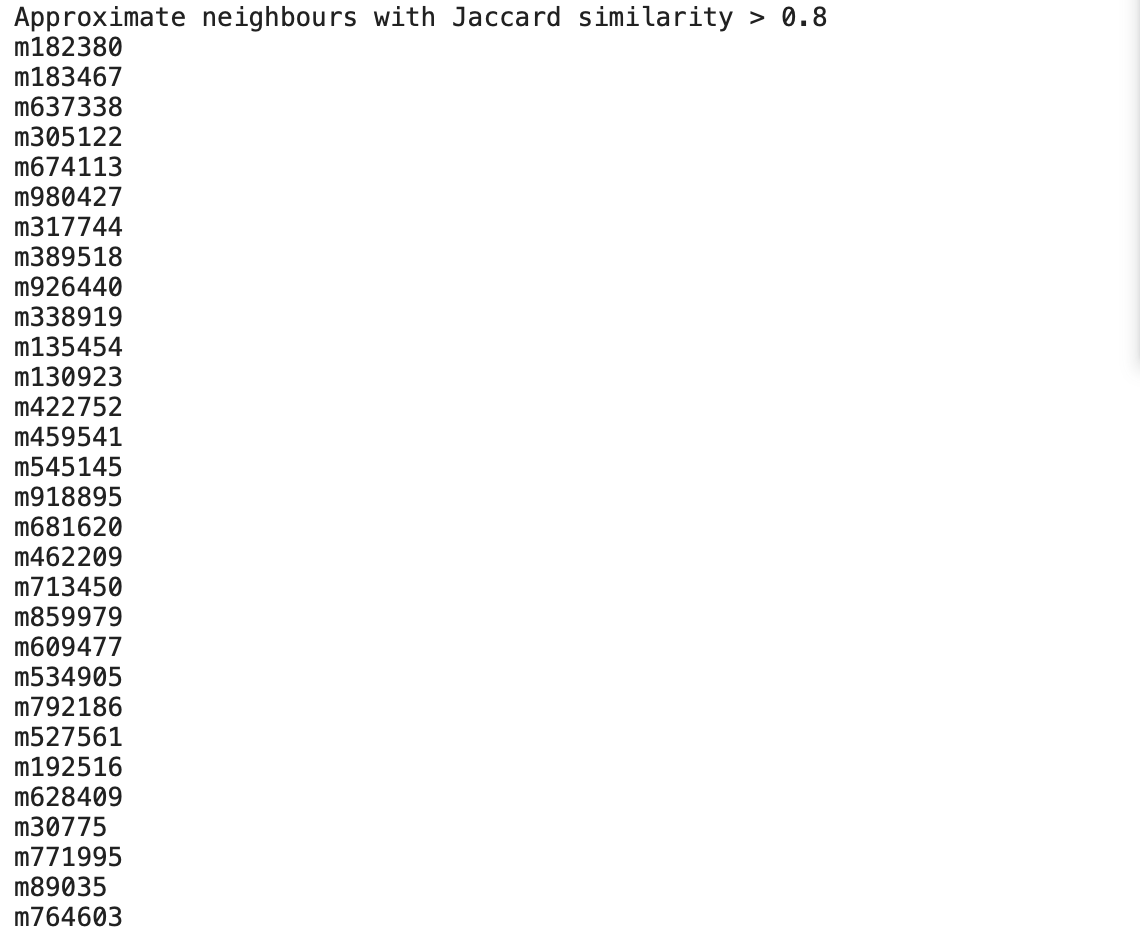


Figure 7 Find the 30 most similar document ids for document 0