Summary of Streaming Algorithm

By Group 7

Members:
Tuan Nguyen
Hoo Chin Hau
Min Chen
Jingyuan Chen
Samir Kumar
Anurag Anshu
Zheng Leong Chua

Introduction

1. The reason why we need streaming algorithm:
There are huge amount of data in our life, such as network traffic, database transactions e.t.c, which is too large to be stored in available memory.
Streaming algorithm aims for processing such data stream, usually the algorithm has limited memory available (much less than the input size) and also limited processing time per item.

2. Evaluation of a streaming algorithm
- ➔ Number of passes of the data stream
- ➔ Size of memory used
- ➔ Running time

3. Two general approaches for streaming algorithm
- ➔ Sampling: Keep part of the stream with accurate information of the individuals which have been chosen
- ➔ Sketching: Keep the summary of the whole streaming but not accurately

4. In this presentation, we are going to introduce:
- ➔ Using sampling to calculate the Frequency moment of a data stream
- ➔ Count-Min Sketch

Sampling based approach

The paper “Space Complexity of Approximating Frequency Moments” discusses space efficient sampling algorithms for estimating frequency moments of numbers. By sampling, we mean the elements are to be read once as they appear. The elements already read cannot be read again.

Let m_i be number of times number ‘i’ appears in the stream. Then F_k = SUM_i m_i^k

For computing k-moment,k>0
1) Pick up an element at random (each element with equal probability), and count the number of times this element appears after (and including) the position you picked it.
Let this number be r. Then output m(r^k - (r-1)^k).
2) To pick up an element at random from sequence of elements streaming in, do the following: when first element comes pick it. When second element comes, throw away the first element with prob 1/2. So at this stage, you have made random selection from 2 elements with equal pobabilty=1/2. When 3rd element comes, accept the selection from first two pick-ups with prob 2/3 and accept the new one with probability 1/3. So probab. of choosing one of the first two elements= 1/2*2/3 = 1/3. Continue this way.

Now we notice that "mean" of output in Point 1) is precisely the k-moment. Consider a particular element 'i'. Let m_i be number of times it comes. Wherever it occurs in the sequence, we pick it up with probability 1/m. The average value of output is : (1/m)*m(m_i^k - (m_i-1)^k + (m_i-1)^k - (m_i-2)^k .... 2^k-1^k + 1^k) = m_i^k . Hence average of output, when summing over all distinct elements is the k-moment.
So if X is the random variable that represents the output, then E(X) = F_k.

Now we can calculate that the variance of X is “kF_1F_{2k-1} - F_k^2”, which is bounded by kn^{1-1/k}F_k^2 - F_k^2 < kn^{1-1/k}F_k^2 .

So we have a random variable with mean F_k and variance less than kn^{1-1/k}F_k^2. If we do repeated sampling, say 's' times, we will be able to get better and better estimate. Let Y = (X1 +X2 + ... Xs)/s ...the average obtained after s rounds. Then E(Y)= E(X) and Var(Y) = Var(X)/s.

Now, suppose you want to know the value of F_k in the range (1-lambda)F_k to (1+lambda)F_k. The probability that this does not happen, for s = 8k n^{1-1/k}/lambda^2, is atmost 1/8.

So this way, error is set down to 1/8. This is obtained by computing Y. Now if we want to get it arbitrarily small, we need to repeat more times. So if we want error epsilon, then repeat log(1/epsilon) times. This completes this part.

Improved version for k=2
Bad space inefficient algorithm:

1) For this, we choose a random n-element sequence e_1,e_2....e_n , with e_i in +1 and -1. Now compute: Z = SUM_i e_im_i, in this way: as the element i comes, update Z -> Z + e_i. Then output Z^2.
2) Then E(Z^2) = SUM_i mi^2 = F_2, since e_i and e_j are mutually independent and E(e_i) = 0; E(e_i^2)=1.
Also E(Z^4) = SUM_{i,j,k,l} E(e_ie_je_ke_l)m_im_jm_km_l . Now using E(e_ie_je_ke_l) = 1 iff either i=j=k=l or (i=j and k=l) or (i=k and j=l) or (i=l and j=k), we get E(Z^4) = Z^4 + 4*SUM_{i< j}m_i^2mj^2 < 3F2^2

Since variance is bounded, one can repeat many times once again to obtain the estimate.

Improving space efficiency:
3) We notice that we just need to compute e_i as the number ‘i’ appears. These “e_i” just have to be 4-wise independent and uniform. We can use the theory of orthogonal arrays to generate them. An orthogonal array of strength 4 with n columns and m rows is a list of n-bit strings, containing m strings, such that in any 4 columns, all possible 4-bit strings appear in equal number. A nice way to generate an orthogonal array of strength 4 is
A) Obtain a k*n matrix G in which any 4 columns are pairwise independent.
B) Generate a k-bit row vector v. Consider the row vector v*G. Every choice of v gives a v*G, which is a row of the orthogonal array.
C) A good choice of G is an array of 3*log(n) rows and n columns, which is a column-wise list of elements of a field F[n] (assume n to be a power of 2). Hence k=3*log(n).

4) hence the algorithm is: generate a random v, which needs 3*log(n) bits. If the number ‘i’ appears, compute the ith element of F[n], which can be done in O(log(n)) space. Then multiply v with this i-th element to get e_i.

Estimating F_0

The Flajolet Martin paper had the required substrate to compute this moment. Assumption was the existence of a family of hash functions that map log(n) bit input to log(n) bit numbers randomly. Then 1/2 of the inputs will be mapped to all the log(n)-bit numbers which have at least one leading ‘0’s. 1/4 will be mapped to those with at least two leading '0's and so on. So if there are p distinct elements, then on an average, at most one element will be mapped to the numbers with atleast log(p) leading ‘0’s. Hence, looking at the max number of leading '0's and taking power of 2 of this number, we get an average estimate.

The present paper uses pairwise independent family of linear hash functions to perform the same analysis. They show that probability that the estimate for F_0 is not between cF_0 and F_0/c is less than 2/c, where ‘c’ is the input to the algorithm.

Count-Min Sketch

Definition:

Count-min Sketch was proposed by Graham and Muthukrishnan in the paper "An improved data stream summary: the count-min sketch and its applications." (Journal of Algorithms 55.1 (2005): 58-75). It is used to solve the problem of keeping track of large number of **n** items, represented as **n** elements in the array **a.** One arbitrary element $\mathbf{a}_i$ can be updated at any time by a value **c**: $\mathbf{a}_i = \mathbf{a}_i + \mathbf{c}$.

Count-min sketch consists of two basic operations:

- *Count*: counting the number or **UPDATE**.
- *Min*: computing the minimum value across the entries or **ESTIMATE**.

The sketch is a two-dimensional **d**-by-**w** array **count: count[1,1], … count[d,w]** determined by two parameters:

- ε: the error of estimation
- δ: the certainty of estimation

in which:

$$d = \left\lceil \ln(\frac{1}{\delta}) \right\rceil, w = \left\lceil \frac{e}{\varepsilon} \right\rceil$$

(*e* is the natural number)

In addition, **d** hash functions are selected uniformly at random from a pair-wise independent hash function family to achieve the transformation: *{1...n} ↦ {1..w}*. Each hash function is corresponding to each row of the array **count**.

UPDATE Operation:

The **UPDATE** operation is defined formally as:

- *UPDATE(i, c)*:
  - Add value **c** to the **i-th** element of **a**
  - **c** can be non-negative (*cash-register* model) or anything (*turnstile* model).
- *Operations*: for each hash function h_j, the obtained hash value of **i** is used as the index to the **count** array and **c** is updated to the array accordingly:

$$count[j, h_j(i)] += \mathrm{c}$$

Queries

There are three types of query:

- Point query, **Q(i)**, returns an approximation of $\mathbf{a}_i$.
- Range query, **Q(l, r)**, returns an approximation of:

$$\sum_{i \in [l,r]} a_i$$

- Inner product query, **Q(a,b)**, approximates:

$$a \circ b = \sum_{i=1}^{n} a_i b_i$$

Point query is the basic query that is utilized by range and inner product query; therefore only point query is covered in this report.

Point query Q(i) in cash-register model:

The result of this query is calculated by first getting **d** corresponding entries (by calculating hash values of **i**) in the **count** array then returning the minimum value:

$$Q(i) = \hat{a}_i = \min_j count[j, h_j(i)]$$

This estimation is guaranteed to be in below range with probability at least **1-δ.** This is the *Theorem 1* in the discussing paper.

$$a_i \le \hat{a}_i \le a_i + \varepsilon \|a\|_1$$

From this theorem, the estimated value is always greater than the actual value of **i-th** element; hence the minimum value among **d** entries is the closest one to the actual value. This theorem can be proved by utilizing the expected collision rate of hash functions and Markov inequality.

Point query Q(i) in turnstile model:

Unlike cash-register model. the estimated value in this model is taken by calculating the *median* of **d** entries:

$$Q(i) = \hat{a}_i = median_j count[j, h_j(i)]$$

Since the estimations returned from **d** rows of sketch can be negative, the *minimum* method provide an estimation which can be far away from true value. Thus, by sorting the values in the increasing order, the *bad* values will be placed in the upper/lower half (too high/too low), while the good values will be placed in the middle which is actually the *median*. Besides, this method can only work well when the number of *bad* estimations is less than **d/2**. The calculation of the probability of getting good estimation was presented in the presentation and it is greater than:

$$1 - \delta^{9/32}$$

Count-min sketch implementation:

Because of the simple and independent operations, the count-min sketch can be implemented in both sequential or parallel/distributed manner to satisfy the updating and querying rate.

Conclusion

In this presentation, we mainly introduced two parts, frequency moments and count-min sketch. Frequency moments are often used to provide useful statistics on the stream. And count-min sketch are used when we need to summarize a data stream.

The frequency moments of a data set represent important demographic information about the data, and are important features in the context of database and network applications.

Count-Min sketch is a compact summary of a large amount of data. We can also view it as a small data structure which is a linear function of the input data. Count-min sketch is widely used in different research areas, such as compressed sensing, networking, databases, NLP and so on.