Data Streams and Frequency moments

Asad Narayanan

November 14, 2015

1 Introduction

Data streaming is an area that is growing in the field of computer science and has many applications in areas such as databases, network monitoring, theory of algorithms etc. Data stream is a sequence of voluminous data arriving at high speed. In most cases the data can be accessed only once. A good example of data stream application would be a network monitoring system which monitors network traffic for some type of attacks. Our aim is to compute statistics of these data with limited amount of time and space. If we plan to keep record of all the IP addresses, then we would require a space of the order 2^{32}. For this purpose, instead of storing entire data we have to generate a sketch of data which can later be reused to compute statistics [3].

2 Definition

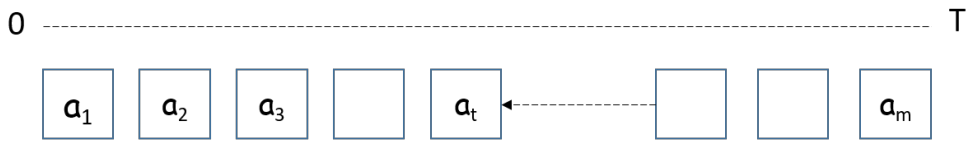

Figure 1: Data Stream

We represent data stream as a sequence of data $A = (a_1, a_2, a_3, \ldots, a_m)$ of length M where each element $a_i \in (1, 2, 3, 4, \ldots, N)$ N distinct elements. Each element a_i occurs m_i times i.e. $m_i = |\{j|a_j = a_i\}|$ [1]. The sequence of data is completely arbitrary. The length of M and N could possibly be very large which makes it impossible to store in local disk. Our aim is to process these data in constant to logarithmic time and space.

3 Frequency Moments

Frequency moments of data streams is a powerful statistical tool which can be used to determine demographic information of data [1]. The k-th frequency moment of sequence A for $k \geq 0$ is defined as:

$$F_k(A) = \sum_{i=1}^{n} m_i^k$$

F_0 is the zeroth frequency moment and it represent number of distinct element in a sequence of data. One of the main application of F_0 is in large databases where query optimizer can find the number of unique elements of an attribute without performing expensive sorting algorithm on entire column [2]. F_1 gives the number of elements in the data stream. Second frequency moment F_2 is used to calculate repeat rate or ginnis index of homogeneity and moments $k > 2$ is used to calculate the skew of the data which has applications in parallel databases to design access plan and query result size [1] [2].

4 Calculating Frequency Moments

A direct approach to find the frequency moments requires to maintain a register m_i for all distinct elements $a_i \in (1, 2, 3, 4, \ldots, N)$ which requires at least memory of order $\Omega(N)$ [1]. But we have space limitations and requires an algorithm that computes in much lower memory. This can be achieved by using approximations instead of exact values. An algorithm that computes an (ε, δ)-approximation of F_k, where $Pr[|F_k' - F_k| \leq \varepsilon F_k] \geq 1 - \delta$, F_k' is the (ε, δ)- approximated value of F_k [3]. Where ε is the approximation parameter and δ is the confidence parameter [4].

4.1 Calculating F_0

4.1.1 FM-Sketch Algorithm

Flajolet et. al in [5] introduced probabilistic method of counting which was inspired from a paper by Robert Morris Counting large numbers of events in small registers. Morris in his paper says that if the requirement of accuracy is dropped, a counter n can be replaced by a counter $logn$ which can be stored in $loglogn$ bits [7]. Flajolet et. al in [5] improved this method by using a hash function h which is assumed to uniformly distribute the element in the hash space (a binary string of length L).

$h : [m] \rightarrow [0, 2^L - 1]$

Let $bit(y, k)$ represent the kth bit in binary representation of y

$$y = \sum_{k \geq 0} bit(y, k) * 2^k$$

Let $\rho(y)$ represens the position of least significant 1-bit in the binary representation of y_i with a suitable convention for $\rho(0)$.

$$\rho(y) = \begin{cases} Min(bit(y, k)) & \text{if } y > 0 \\ L & \text{if } y = 0 \end{cases}$$

Let A be the sequence of data stream of length M whose cardinality need to be determined. Let $BITMAP[0 \dots L-1]$ be the hash space where the $\rho(hashedvalues)$ are recorded. The below algorithm the determines approximate cardinality of A.

Procedure 1 FM-Sketch

```
for i in 0 to L − 1 do
  BITMAP[i]:=0
end for
for x in A: do
  Index:=ρ(hash(x))
  if BITMAP[Index] = 0 then
    BITMAP[Index] := 1
  end if
end for
B:= Position of left most 0 bit of BITMAP[]
return 2^B
```

If there are N distinct elements in a data stream.

- For $i >> log(N)$ then $BITMAP[i]$ is certainly 0
- For $i << log(N)$ then $BITMAP[i]$ is certainly 1
- For $i \approx 1$ then $BITMAP[i]$ is a fringes of $0's$ and $1's$

Example (FM-Sketch)

Let the following represents the datastream.

a_1	a_2	a_3	a_4

Let the hashed values be:
$h(a_1) = 0\mathbf{1}1001$
$h(a_2) = \mathbf{1}00101$
$h(a_3) = \mathbf{1}01100$
$h(a_4) = 0\mathbf{1}1011$

Then according to algorithm BITMAP will be equal to
$BITMAP = 11\mathbf{0}00000$

First occurrence of $0 - bit$ is at position 2
$\implies F_0 = 2^2 = 4$

To improve the accuracy, Flajolet and Martin extended the algorithm by taking an array of bit strings instead of one and the position of 0 is averaged. This algorithm is called Probabilistic counting with stochastic averaging (PCSA).

Next we will discuss an algorithm which is an improvement of this algorithm.

4.1.2 K-Mninmum Value Algorithm

We have seen the first attempt to approximate F_0 in the data stream by Flajolet and Martin. Their algorithm picks a random hash function which they assume to uniformly distribute the hash values in hash space. Such an ideal hash function is not possible in reality.

Bar-Yossef et al. in [4], introduces k-minimum value algorithm for determining number of distinct elements in data stream. They uses a similar hash function h which can be normalised to [0,1] as $h : [m] \rightarrow [0, 1]$. But they fixed a limit t to number of values in hash space. The value of t is assumed of the order $O(\frac{1}{\varepsilon_2})$ (i.e. less approximationvalue ε requires more t). KMV algorithm keeps only t smallest hash values in the hash space. After all the m values of stream are arrived, $\upsilon = Max(h(a_i))$ is used to calculate $F_0' = \frac{t}{\upsilon}$. That is, in a close-to uniform hash space, they expect atleast t elements to be less than $\frac{t}{F_0}$.

Procedure 2 K-Minimum Value

```
Initialize first t values of KMV
for a in a_1 to a_n do
  if h(a) < Max(KMV) then
    Remove Max(KMV) from KMV set
    Insert h(a) to KMV
  end if
end for
return t/Max(KMV)
```

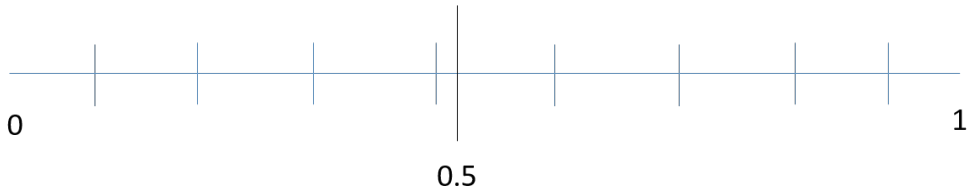

Figure 2: Values distributed in hash space

Example KMV

Let us assume there are 8 distinct element in a data stream and the number of hash values $t = 4$.

Then we can say that atleast 4 hashed values will be less that 0.5, as shown in Figure 2.
This means t-th smallest will be approximately equal to 0.5. So we can assume $\upsilon \approx 0.5$.

So, $F_0 = t/\upsilon = 4/0.5 = 8$

Complexity analysis of KMV

KMV algorithm can be implemented in $O((\frac{1}{\varepsilon_2}).log(m))$ memory bits space. Each hash value requires space of order $O(log(m))$ memory bits. There are hash values of the order $O(\frac{1}{\varepsilon_2})$. The access time can be reduced if we store the t hash values in a binary tree. Thus the time complexity will be reduced to $O((\frac{1}{\varepsilon}).log(m))$.

4.2 Calculating F_k

Alon et al. in [1] estimates F_k by defining random variables that can be computed within given space and time. The expected value of random variable gives the approximate value of F_k.

Let us assume length of sequence m is known in advance.

Construct a random variable X as follows

- Select a_p be a random member of sequence A with index at p. $a_p = l \in (1, 2, 3, n)$
- Let $r = |\{q : q \geq p, a_p = l\}|$, represents the number of occurrences of l within the members of the sequence A following a_p.
- Random variable $X = m(r^k - (r-1)^k)$

They assume S_1 be of the order $O(n^{1-1/k}/\lambda^2)$ and S_2 be of the order $O(log(1/\varepsilon))$. Algorithm takes S_2 random variable $Y_1, Y_2, \ldots, Y_{S_2}$ and outputs the median Y. Where Y_i is the average of X_{ij} where $1 \leq j \leq S_1$.
Now we calculate expectation of random variable $E(X)$.

$$E(X) = \sum_{i=1}^{n} \sum_{i=1}^{m_i} (j^k - (j-1)^k)$$

$$= \frac{m}{m}[(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k))$$

$$+(1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)) + \ldots$$

$$+(1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k))]$$

$$= \sum_{i=1}^{n} m_i^k = F_k$$

Complexity of F_k

From the algorithm to calculate F_k discussed above, we can see that each random variable X stores value of a_p and r. So, to compute X we need to maintain only $log(m)$ bits for storing a_p and $log(n)$ bits for storing r. Total number of random variable X will be the $S_1 * S_2$.

Hence the total space complexity the algorithm takes is of the order of

$$O\left(\frac{klog(\frac{1}{\varepsilon})}{\lambda^2} n^{1-\frac{1}{k}} (logn + logm)\right)$$

Simpler approach to calculate F_2

The previous algorithm calculates F_2 in $O(\sqrt{n}(logm + logn))$ memory bits. Alon et. al in [1] simplified this algorithm using four-wise independent random variable with values mapped to $\{-1, 1\}$.

This further reduces the complexity to calculate F_2 to

$$O\left(\frac{log(\frac{1}{\varepsilon})}{\lambda^2} (logn + logm)\right)$$

5 Exercise

1. In the algorithm to calculate F_k we assumed the size of m is known. Explain how the algorithm will function if size of m is unknown.

References

[1] Alon, Noga, Yossi Matias, and Mario Szegedy. 'The Space Complexity Of Approximating The Frequency Moments'.Journal of Computer and System Sciences58.1 (1999): 137-147

[2] Woodruff, David. 'Frequency Moments'. (2005): 2-3.

[3] Indyk, Piotr, and Woodruff David. 'Optimal Approximations Of The Frequency Moments Of Data Streams'.Proceedings of the thirty-seventh annual ACM symposium on Theory of computing - STOC '05(2005): 202.

[4] Ziv, Bar-Yossef et al. 'Counting Distinct Elements In A Data Stream.'.International Workshop on Randomization and Approximation Techniques2483 (2002): 1-10.

[5] Philippe, Flajolet, and Nigel Martin G. 'Probablistic Counting Algorithms For Database Applications'.Journal of computer and system sciences31.2 (1985): 182-209.

[6] Morris, Robert. 'Counting Large Numbers Of Events In Small Registers'.Communications of the ACM21.10 (1978): 840-842.

[7] Flajolet, Philippe. 'Approximate Counting: A Detailed Analysis'.BIT25.1 (1985): 113-134.

[8] `http://research.neustar.biz/2012/07/09/sketch-of-the-day-k-minimum-values/`