COMP 480/580 — Probabilistic Algorithms and Data Structure Sept 27, 2022

Lecture 11:Count Distinct and Norm Estimation on Streams

Lecturer: Anshumali Shrivastava Scribe By: Bocheng Zhang, Chieh Ju Chueh, Haixiao Wang

1 Introduction

Estimating the number of distinct values in a data set is a well-studied problem with many applications. For example, estimates of the number of distinct values for an attribute in a database table are used in query optimizers to select good query plans. In addition, Distinct-values estimates are also useful for network resource monitoring, in order to estimate the number of distinct destination IP addresses, source-destination pairs, requested urls, etc. In network security monitoring, determining sources that send to many distinct destinations can help detect fastspreading worms.

This lecture is mainly focus on the method to count the distinct element in the stream space efficiently. The motivation for this is that in the most of the real world scenarios, you are supposed to process the big amount of data with limited storage capacity. One of the application is anomaly detection, which is to detect the unusual behavior in the big data stream.

2 Naive Approaches

2.1 Approach 1: Hash Table

One of naive approaches would be to use a hash function to store every element in the data stream. Once you receive a new element, you hash it and store the value in a hash table. However, this method requires a huge amount of storage, and it is at least the count of unique numbers in the stream. If there are M distinct numbers in the stream, then the space complexity would be O(M).

2.2 Approach 2: Sampling method

Another approach might be sampling-based method, which means collecting a sample of the data and then applying sophisticated estimators on the distribution of the values in the sample. However, it turns out to fail as well.

3 Advanced Approaches

3.1 Approach 1

3.1.1 Core Idea

The biggest problem is how to save the space. And here comes the first idea. You have a hash function, and once you see the new item you maps them into a uniformly random real number in [0, 1]. Define Y to be the smallest hash value of m distinct elements. Then we update the Y once we get a smaller hash value.

$$Y = Min_{i=1}^{i=m}(h(x_i))$$

The expectation of the minimum of this random variable is $\frac{1}{m+1}$, which is our observation Y, we can invert the o to get $\frac{1}{Y}$ and reduce it by 1 and then we get our result m. The space complexity is constant because we only store the minimum hash values.

3.1.2 Proof

Assuming a is uniformly distributed from [0, 1]. We have the equation as follows.

$$Pr(Min(x_i) <= x) = 1 - Pr(Min(x_i) >= x) = 1 - (1 - x)^m$$

So the density function is

$$f(x) = m(1 - x)^{m-1}$$

Then

$$\int_0^1 xf(x)dx = m\int_0^1 (1 - t)t^{m-1}dt = \frac{1}{m+1}$$

However, when we take a look at its variance

$$Var(Y) = \frac{m}{(m+1)^2(m+2)} \approx \frac{1}{(m+1)^2}$$

We can draw a conclusion that our observation Y is random variable with mean $\frac{1}{m+1}$ and standard deviation $\frac{1}{m+1}$. Thus, it is not enough to use this as our estimator because of the high variance.

3.1.3 Improvement

The intuitive thinking is to try to smooth the variance by take the mean of Y, thus, we can use set a of hash function to reduce the variance of the observation.

3.2 Approach 2

3.2.1 Core Idea

The second idea is to keep a bloom filter and once you see a new item, apply a sign function to it to decide whether to keep it or discard it (Sign function only outputs two values 0 or 1). Basically, you are just scanning the $\frac{N}{2}$ elements where N is the total number of elements in the stream. Now, you are supposed to count the distinct element in $\frac{N}{2}$ elements and you can multiply it by 2 to get the total number of distinct values in the whole data stream. And if you continuously repeat this process on the $\frac{N}{2}$ to get an estimation of unique value in $\frac{N}{4}$, $\frac{N}{8}$, etc. Finally, you can make the space complexity to be O(Log(N)).
There is a similar algorithm called Flajolet-Martin which uses position of rightmost set and unset bit to count the distinct elements in the stream. The space complexity is O(Log(M)) where M is the number of distinct elements in the stream. The probability that the rightmost set bit is at position is at 0 is $\frac{1}{2}$ (imagine half numbers is odd and half even), probability of

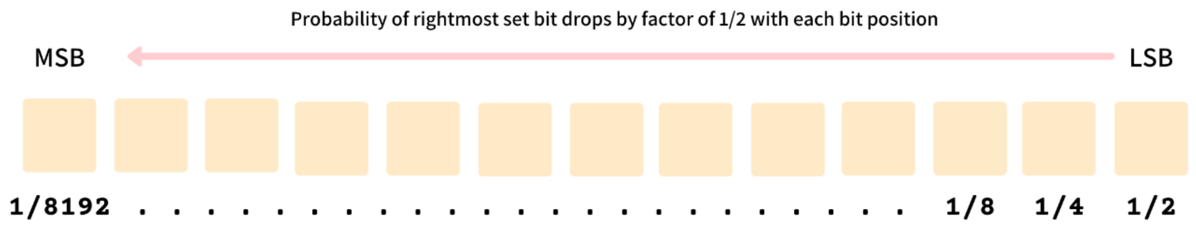
Probability of rightmost set bit drops by factor of 1/2 with each bit position
MSB
LSB
1/8192 . . . . 1/8 1/4 1/2

Figure 1: probability of the rightmost set bit drops by a factor of 1/2

rightmost set bit is at position is at 1 is $\frac{1}{2} * \frac{1}{2} = \frac{1}{4}$ and so on. Thus, the probability that the rightmost set bit is at position is at k is

$$Pr[position = k] = 2^{-k-1}$$

The probability should become 0 when b is greater than log(M) where M is count of distinct number in stream. Thus, if we find rightmost position b such that the probability is 0 which means that b = log(M), so the M is equal to 2^b. This is the core idea behind the Flajolet-Martin.

3.3 Approach 3

3.3.1 Core Idea

The shortcomings of the Flajolet-Martin is that it cannot work for every moment, so here we need another method. This method should work for all moments, and gives an unbiased estimate.

Here we just concentrate on 2^{nd} moment. Based on calculation of many random variables X. Each random variable is analogous to recording the maximum number of zero in the tail of stream using a fixed hash function like we did for the Flajolet-Martin algorithm we mentioned above. Each element requires a count in main memory, so the number will be limited.

Assume the stream has the length n, and chosen time has element a in the stream. So: X = n*((twice the number of a's in the stream starting at the chosen time)) - 1. So the 2^{nd} moment is $\sum_a (m_a)^2$. $E(X) = (\frac{1}{n}) * (\sum_t n^*$(twice the number of times the stream element at time t appears from that time on)-1).

So for the each X: it will be $\sum C_1$, $\sum C_2$, $\sum C_3$ and so on. And for the position j in those X:

$$S(X_j)X_j = C_j * S(X_j)^2 + \sum_{i \neq j} C_i * Pr[h(X_i) = h(x_j)]^{S(X_i)} * S(X_j)$$

for which is our estimator. So we will also have our expectation:

$$E(X_j * S(X_j)) = C_j$$

We can also get this result for squared condition:

$$S[X_j * S(X_j)]^2 = [C_j^2 + \sum_{i \neq j} C_i^2 * Pr[h(X_i) = h(x_j)]] + NULL = [C_j^2 + \frac{1}{R} \sum_{i \neq j} C_i^2]$$

So we will get:

$$S[X_j * S(X_j)]^2 = \sum C_i^2$$

3.3.2 Improvement

We can also improve the current solution by running the algorithm multiple times with k different hash functions and try to combine those results. And here we can take several solutions to improve the algorithm.

- One idea is to take the mean of k results from each hash function, and obtain a single estimate of the cardinality. However, there are also some problems with this method. For example, the averaging is very susceptible to outliers.

- Another idea is to use the median, which is less prone to be influenced by outliers. However, the problem to this improving solution is that the result can only take from $2^R/\phi$, where R is an integer.

- The final solution is to combine both the mean and the median. Creating the $k \cdot l$ hash functions and split them into k distinct groups, for each group with a length l. Within each group use the mean for aggregating the l results, and finally take the median of the k group estimates as the final estimate.

References

[1] Phillip B. Gibbons. *Distinct-Values Estimation over Data Streams.* 2016.

[2] Pemi Nguyen "Section 4: Approximating Distinct Elements in a Stream" [Online]. Available: https://courses.cs.washington.edu/courses/cse312/. [Accessed: 16 July 2020].

[3] Arpit Bhayani."Approximate Count-Distinct using Flajolet Martin Algorithm?" https://arpitbhayani.me/blogs/flajolet-martin.

[4] [Online]. Available: https://en.wikipedia.org/wiki/Flajolet%E2%80%93Martin_algorithm.