Advances in Data Mining

Assignment 2 - Counting Distinct Elements

October 16, 2017

Abstract

These days billions of data is being streamed on daily basis which one needs to process and establish a good understanding of its platform to either generate new content for its users or establish a basic statistics for itself like how many users on daily basis; how much of those are unique; and what if one need to query the data with complex SQL queries? Having a lot of memory to calculate these incoming data is quite essential these days which might make life easier for some of us but what if one has few bytes of memory and yet willing to interact with the data, for instance, counting the elements as efficiently as possible. In this paper we are going to establish a good understanding for Probabilistic counting by Flajolet-Martin and LogLog counting by Durand-Flajolet.

1 Introduction

In this paper my goal is to count the number of distinct elements in a data stream of size n. This could be done in number of ways but whether these theories or observations are efficient in time and space is entirely different topic. One of the ways to read the data in chunks this way you could establish a better understanding and perhaps speed. Another approach is to estimate the count in an unbiased way. In some of the approaches the count might result errors and so the percentage of accuracy goes down. I will implement and talk about Algorithms based on Philippe Flajolet theorem and Marianne Durand and whether I could count n size of data within limited memory resource.

2 Probabilistic Counting (Flajolet-Martin)

2.1 Brief description And Theoretical Properties

This algorithm was published during 1983-1985 by Flajolet and Martin. The principle idea behind this algorithm is to use hashing on the data coming in and output the cardinality (number of distinct elements) of that stream.

The algorithm goes as following. Hash function maps $h(x)$ each of the m elements to least $log2$ m bits; Count trailing zeros of each $r(a)$ in its $h(a)$; Record maximum number of trailing zeros R you have seen so far. Do this for all the elements in the data stream and calculate the distinct elements. Aside from this the algorithm needs to maintain a bitmap table used to record the values of $r(a)$, also to increase the process accuracy of the algorithm it uses stochastic averaging. So for instance one binary order of magnitude to $O(1)$. The idea is to split the input stream into m groups = 2t which are determined by the first l bits of hashed values. In this case R is the average of each group and estimate the scale by m. The final formula to estimate the cardinality is

$$\alpha = \frac{m}{\Phi} 2^{A} ve$$

in which m is number of elements, Φ is the correction factor (0.77351) and Ave is average of R. Based on number of researchers the algorithm proves that it can be possible to estimate number of cardinality of large data sets using m words of size log2 N which comes close to

$$\alpha = \frac{0.78}{\sqrt{m}}$$

The drawback of this algorithm is that 2^R always results to a power of 2. Also which will result in huge errors. The suggested workaround for these problems is to run several copies of the algorithm in parallel with different groups of hash functions. You should then calculate the average of each group and then take the median of the averages as your result.

2.2 Experiments

I have done number of Experiments and playing around the parameters, true values and m. For each value of m, I ran the code a number of times and calculated the average of the error rate. The formula I used for the calculation of the error rate is:

$$RAE = \frac{true_count - estimated_count}{true_count}$$

I started with 100000 random numbers as a true count. I ran the code for values of m starting from 2 to 252 with increments of 5. Pretty much plotted different values of m and number of other input values. Again using the random value (with different params) I generated 200000 random numbers as an input (true count) in this case value of m was starting 80 up until 580 with 5 as an increasing value.

I also worked with real datasets[1] In the experimental folder you find the files in which I worked with real datasets. Using the computers at the university to calculate the cardinality and get some testable results

2.3 Results

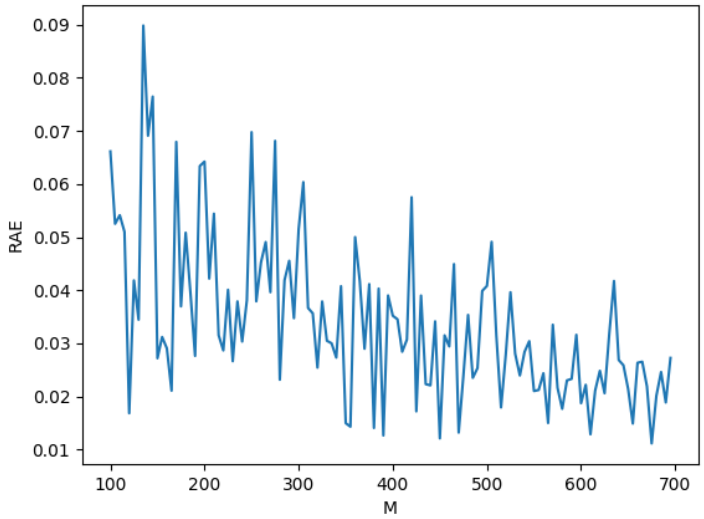

Figure 1: A trade off between memory and error rate using the Probabilistic counting algorithm. 100000 as Input value, m ranging from 2 to 252

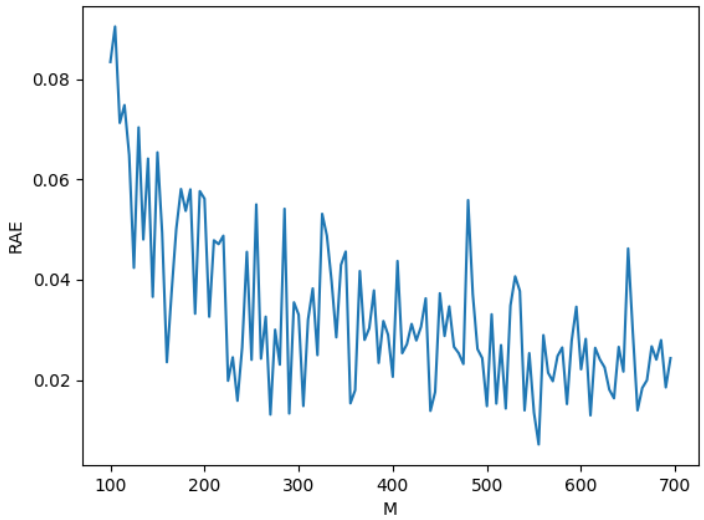

Figure 2: A trade off between memory and error rate using the Probabilistic counting algorithm. 200000 as Input value, m ranging from 2 to 252

[1] I downloaded the dataset from SNAP

3 LogLog counting of large cardinalities (Durand-Flajolet)

3.1 Brief description And Theoretical Properties

After the first algorithm Flajolet and Durand came together again to refine their latest work and as such that the thoughts were to could make the memory consumption of the algorithm even better and so they came up with LogLog Counting algorithm. The LogLog algorithm is pretty much similar to the Probabilistic Counting algorithm - some say its just a improvement.
LogLog has pretty much the same principle and so it uses hash function as well on the input data to output a binary numbers, it then takes the binary and finds the cardinality estimates using only a very small amount of auxiliary memory, namely m memory units, where a memory unit, a "small byte", comprises close to $loglogN$ max bits, with N max an a priori upper bound on cardinalities. It also uses the same principles of stochastic averaging as the Probabilistic Counting algorithm and uses a correction factor of 0.79402.

The final formula to the estimate of the cardinality is then:

$$m\Phi 2^{(}Ave)$$

In which m is number of elements, Φ is the correction factor and Ave is average of R. The algorithm proves that it is possible to estimate the cardinality of large multi sets (up to several billion distinct elements) using m short bytes.

3.2 Experiments

My experiments were different in m and amount data as input. To get some visual representation I have done some plotting just to show the difference between the algorithms. For each change in m I ran the code at least twice and on two different machines just to get some ideas. The formula used for the calculation of the average of the error rate is:

$$RAE = \frac{true_count - estimated_count}{true_count}$$

At the beginning I have used random function to generate either 100000 or 500000 random numbers. Those two numbers were the $true_count$

3.3 Results

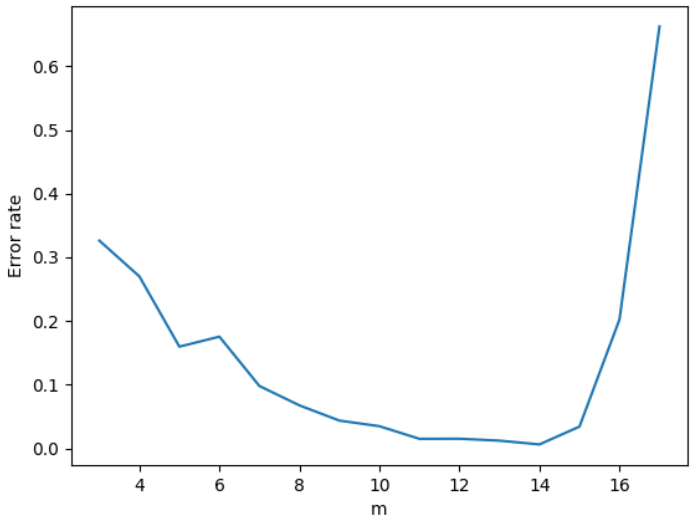

Figure 3: A trade off between memory and error rate using the LogLog counting algorithm. 100000 as Input value, m ranging from 2 to 252

For the dataset of 100000 random input values, we have reached error rate of (0.00611) when the memory m was 14.

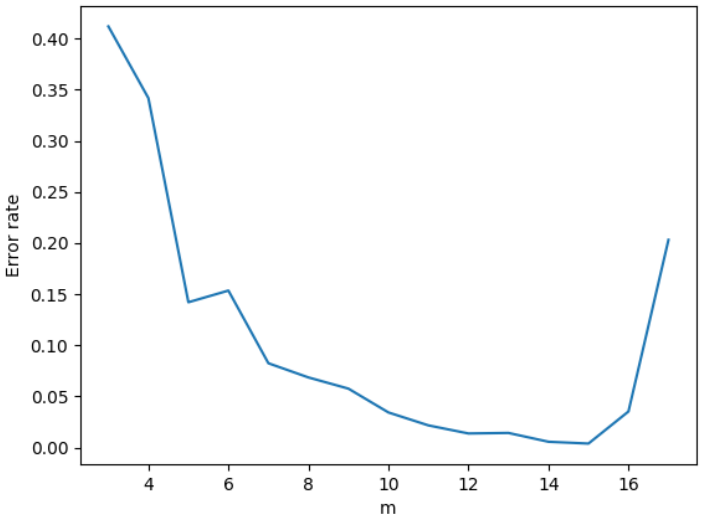

Figure 4: A trade off between memory and error rate using the LogLog counting algorithm. 200000 as Input value.

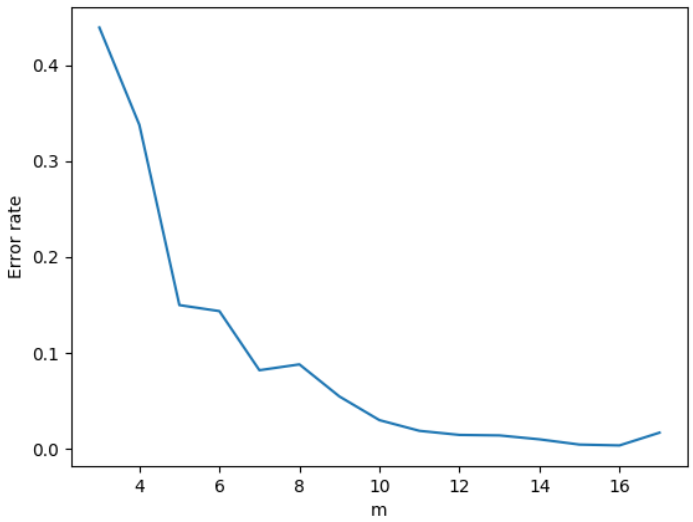

Figure 5: A trade off between memory and error rate using the LogLog counting algorithm. 500000 as Input value.

4 Conclusion(s)

The trick regarding Probabilistic Counting algorithm is within the error rate and the amount of memory used. The higher the amount the memory used the lower the error rate which this could be seen in the graph. For LogLog Counting Algorithm lies within the prober range of the number of groups m, the more the groups is divided (more memory), the lower the rate will be as well. Both algorithms has been proven by number of experts in this field. I was also to discover that once the data set has been excessively divided,the error rate stops decreasing and starts increasing.

According to the papers, the Probabilistic Counting algorithm is slightly more accurate than the LogLog Counting one, as it uses the bitmap method to keep track of the trailing zeros. However, when it comes to memory consumption,the memory units of LogLog Counting are smaller by about a factor of 3-5, depending on implementation in comparison to the Probabilistic Counting which this might result differences in the accuracy. As final thoughts LogLog is, as mentioned before still being used these days for small operations and so by the end I would recommend using this Algorithm for counting distinct elements.