Advances in Data Mining: Assignment II.

Haoran Ding
bitdhr@hotmail.com

Petra Kubernatova
pkubernatova@gmail.com

Introduction

A data stream consists of elements chosen from a set of size n. However, this n can be very big. In this assignment, we are aiming to count the number of distinct elements in a stream. One would say that the obvious way to do this is to maintain the set of elements already seen, but what if we don't have enough memory to store the complete set? Another approach would be to simply estimate the count in an unbiased way. Within this approach we must accept that the count may be in error. Thus, we lose accuracy.

In this report we take a closer look at two more advanced algorithms which could be the answer to our problem, the number of distinct elements can be gathered with little loss of information, only a small probability of failure and minimal required storage. We also seek to answer the question "How to count distinct elements in limited memory?" by implementing the Flajolet-Martin and Loglog Algorithm as follows.

1 Probabilistic Counting (Flajolet-Martin)

1.1 Description and Theoretical Properties

The complete algorithm was published in the period 1983–1985 by Flajolet and Martin. It uses a technique combining hashing and signatures. The input of the algorithm is a data stream and the output the cardinality (number of distinct elements) of that stream.

You first pick a hash function h which maps each of the m elements to at least $\log_2 m$ bits. For each element a of the stream, you count the number of trailing zeroes $r(a)$ in its $h(a)$. You record the maximum number of trailing zeroes R you have seen so far. You go through all elements in the stream like this and then calculate the number of distinct elements as 2^R.

The algorithm needs to maintain a bitmap table. It is used to record on the go values of $r(a)$ that we have observed.

The algorithm also uses stochastic averaging. This process increases the accuracy of the algorithm significantly (from one binary order of magnitude to $O(1)$. You split the input stream into m groups $m = 2^l$ which are determined by the first l bits of hashed values. You average the R of each group and then scale the estimate by m.

The algorithm as described so far would be biased, so a correction factor has been devised to make it asymptotically unbiased. The correction factor φ is equal to 0.77351.

The final formula for the estimate of the cardinality is then

$$\frac{m}{\varphi} 2^{Ave}$$

where:

m = number of elements
φ = correction factor (0.77351)
Ave = average of R

The algorithm proves that it is possible to estimate the cardinality of large data sets (up to over a billion distinct elements) using m words of auxiliary memory and a relative accuracy close to

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

Later on, Flajolet and Durand realised, that they could make the memory consumption of the algorithm even better and the LogLog Counting algorithm was born.

The drawback of this algorithm is that 2^R always results to a power of 2. Also, it is prone to bad luck, 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.

1.2 Experiments

We have experimented with the number of the groups m. We have plotted the results into a graph. For each value of m, we ran the code a number of times and calculated the average of the error rate. The formula we used for the calculation of the error rate is:

$$RAE = \frac{|true\_count - estimated\_count|}{true\_count}$$

First, we generated 100000 random numbers. We took 100000 as the true count. We ran the code for values of m starting from 2 to 252 with increments of 5. Next, we have tried plotting graphs for numerous different values of m and numerous different input values (random strings, different sets of random integers etc.). Another interesting setup was this one:

We used the *random* function from the random module to generate 500000 random numbers. We took 500000 as the true count. We ran the code for values of m starting from 75 until 575 with increments of 5.

1.3 Results

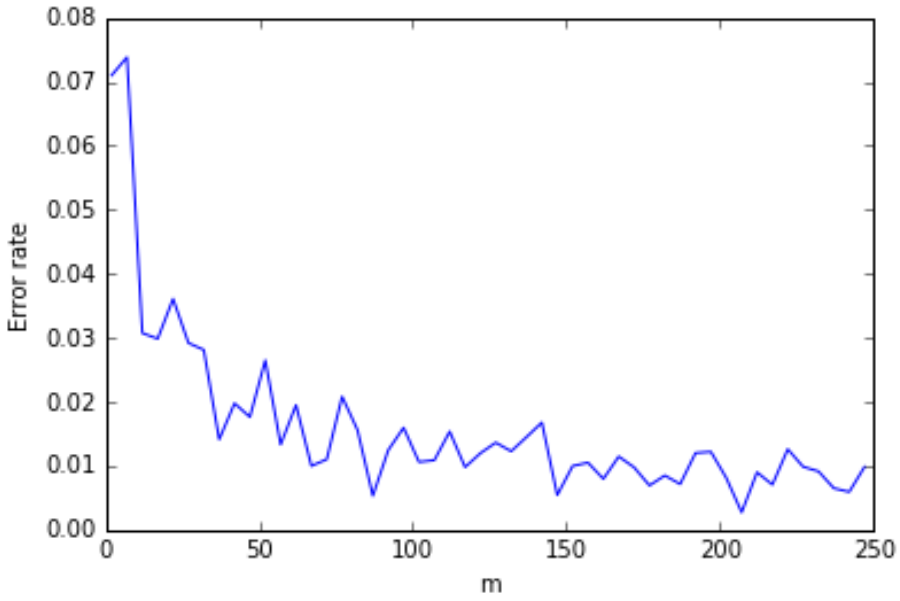

Figure 1: Trade off between memory and error rate of the Prob. Counting algorithm, 100000 input values, m ranging from 2 until 252 in steps of 5

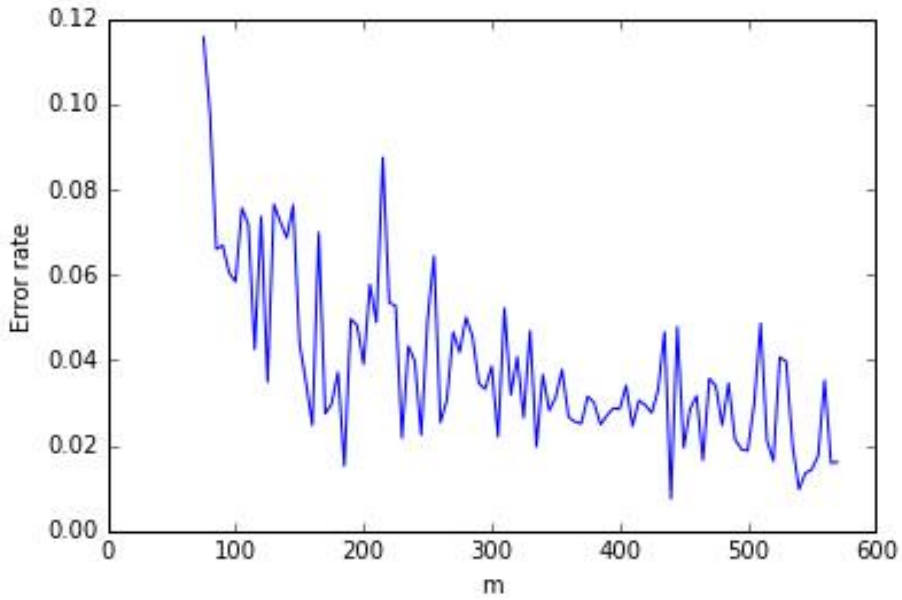

Figure 2: Trade-off between memory and error rate of the Prob. Counting algorithm, 500000 input values, m ranging from 75 until 575 in steps of 5

At first we did not expect the plot to bounce like that. We predicted it would be more steady. We have tried to get rid of the bouncing in numerous ways. First for each number of group m we ran the algorithm 10 times and used the mean error rate as the result. And we also used the random number generator method instead of the hash function. However the results still showed the bouncing. In the end we believe that these plots are sufficient to show our results, because the error rate has a declining tendency.

2 LogLog Counting (Durand-Flajolet)

2.1 Description and Theoretical Properties

The LogLog algorithm is very similar to the Probabilistic Counting algorithm. It has the same input and output. It also uses a hashing function to bring the input data to a form that resembles random binary data. It then takes this hashed data set and makes cardinality estimates. It performs various tests on the hashed data set, compares "observables" to what probabilistic analysis predicts and "deduces" an estimate of the cardinality value.

In the case of the LogLog algorithm, this "observable" is the right-most 1-bit of the hashed input value $h(a)$. As stated in the paper, the "observable" should only be linked to cardinality and hence be totally independent of the nature of

replications and the ordering of the data input (since we have no information about these anyways).

It uses the same principles of stochastic averaging as the Probabilistic Counting algorithm and uses a correction factor φ of 0.79402.

The final formula for the estimate of the cardinality is then

$$m\varphi 2^{Ave}$$

where:

m = number of elements
φ = correction factor (0.79402)
Ave = average of R

The algorithm proves that it is possible to estimate the cardinality of large multisets (up to several billion distinct elements) using m short bytes (less than 8 bits) of auxiliary memory, with a relative accuracy close to

$$\alpha = \frac{1.30}{\sqrt{m}}$$

2.2 Experiment

We have experimented with the number of the group m. We have plotted the results into a graph. For each value of m, we ran the code 20 times and calculated the average of the error rate. The formula we used for the calculation of the error rate is:

$$RAE = \frac{|true\_count - estimated\_count|}{true\_count}$$

We have used the *random* function from the random module to generate 100000 random numbers. We took 100000 as the true count.

2.3 Result

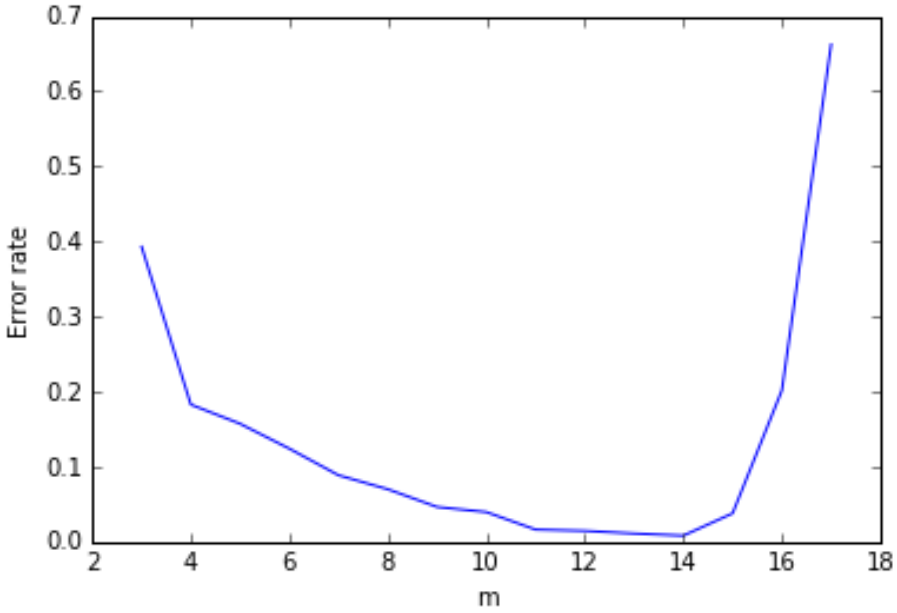

Figure 3: Trade-off between memory (number of groups) and error rate of the LogLog algorithm

For the dataset containing 100000 random elements, we have reached the optimum of the error rate (0.00819) when the group number m was 14.

The declining and the increasing were both predicted before the experiment. As the declining shows the result correlated to the predicted error rate $\frac{1.30}{\sqrt{m}}$. And the increasing suggests that when we choose the number of m, we take the size of the dataset into consideration. As we divide the dataset into too many groups the accuracy will be lost.

3 Conclusions

The trade-off between the error rate and the amount of required memory for the Probabilistic Counting algorithm is that within the proper range of the number of groups, the higher the amount of memory used, the lower the error rate (proven by the graph, where the plot of the error rate is decreasing with increasing m).

For the LogLog Counting algorithm, within the proper range of the number of groups m, the more the group is divided (the more memory is used), the

lower the error rate will be. It has been proven theoretically and experimentally. Also we have found that once the data set has been excessively divided, the error rate stops decreasing and starts increasing. Because it will dilute the properties (the trailing zeros or ones) we used to estimate the distinct elements of the data set. More specifically, as we use the average of the maximum number in all the groups, the excessive division would make the big maximum number in one group less significant (averaged out). We would then lose track of the properties. This has been experimentally proven by the graph we plotted in the experimental section of this document.

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. This difference makes the difference in accuracy insignificant. Our results show that these theories are true.

So for counting distinct elements with limited memory, the LogLog Counting algorithm is recommended.