Counting Distinct Elements in a Data Stream Using Durand-Flajolet and Flajolet-Martin Algorithms

anon1 anon2

October 9, 2018

Abstract

In this assignment we implemented the Probabilistic Counting with Stochastic Averaging (PCSA) and LogLog algorithms to count the number of distinct elements in a large data stream. Over a set of experiments, we find the PCSA algorithms performs with an error of $\sigma \approx \frac{0.78}{\sqrt{m}}$. The LogLog algorithm performs $\sigma \approx \frac{1.30}{\sqrt{m}}$. This trade is minimal, considering that the memory required to run the LogLog algorithm is substantially smaller than what is required for PCSA.

1 Introduction

The problem of cardinality, or determining the distinct elements of a large data set, is a common problem in multiple fields. As noted in Durand & Flajolet (2003), data streams can often be too large to fit to memory all at once, or even store, in some cases delivering 0.5 GB per hour of compressed data. Simple approaches like keeping in memory a list of all elements becomes impossible for data streams of this size, and are inefficient in other ways, such as run time. We set out to implement algorithms that would solve alleviate this problem and run efficiently. We explore two algorithms, the Flajolet-Martin algorithm (FMA, or Probabilistic Counting with Stochastic Averaging algorithm, PCSA) and the Durand-Flajolet (LogLog) algorithm, to solve cardinality.

2 Methods

2.1 Flajolet-Martin algorithm

The PCSA algorithm aims to solve the basic problem of cardinality. For simple counting techniques, because the necessary number of operations is $\log_2 N$ (Flajolet 2004), arising from the fact that there are only 2^l ways of distinguishing possibilities, one cannot keep a counter of unique items between 1 and N in a stream with less than $\log_2 N$ bits. So, when dealing with large streams of data, an estimation becomes necessary when resources and run time do not allow for a full storing of the stream.

In estimating the number of distinct elements in a stream in a single pass, Flajolet & Martin (1985) make extended use of hash functions, which maps data

to a fixed size. In the FMA, for every hash function calculation, we write the binary equivalent for each result and write the count of the trailing zeros. For example, if 010, 100, 011, and 000 are the resulting trailing zeros for a sample hash function, the written amount of trailing zeros would be 1, 2, 0, and 0 respectively. In general, it is observed that if the values hash(x) are uniformly distributed, the pattern $0^k1...$ appears with probability 2^{-k-1} (Flajolet & Martin 1985). Following this probability pattern (that is, analyzing the trailing 0 bits) we can use the trailing zeros to define a value R as an indicator of $\log_2 n$, namely,

$$E(R) \approx \log_2 \phi n,$$

where $E(R)$ is the estimated value of R, and ϕ is a correction value for biases that could be present in the implementation of the algorithm (Flajolet & Martin 1985). We seek to obtain estimates on unique elements in the stream n from the observation of the parameter R. However, as Flajolet & Martin (1985) note, the dispersion of results corresponds to a typical error of 1 binary order of magnitude (factor of 2) which is too high for many applications. The FMA idea to remedy this situation consists in using a set H of m hashing functions, where m is a design parameter and computing m different computed (Flajolet & Martin 1985). We obtain then m estimates on $R^{<1>}$,..., $R^{<m>}$ parameters. We then consider the average,

$$A = \frac{R^{<1>}, ..., R^{<m>}}{m}, \tag{1}$$

where the random variable A satisfies,

$$E(A) \approx \log_2 \phi n \ \text{ and } \ \sigma(A) \approx \frac{\sigma_{\inf}}{\sqrt{m}}. \tag{2}$$

Thus we expect 2^A to be the estimate of n with relative error $\sigma \approx \frac{K}{\sqrt{m}}$ for some constant K (in bits). To continue with the stochastic averaging approach, the hashing function is used to distribute each record into one of the m lots, computing $\alpha = h(x) \bmod m$. We expect $\frac{n}{m}$ elements to fall into each lot so that $\frac{1}{\phi}2^A$ is a reasonable approximation of $\frac{n}{m}$. The stochastic averaging element of the algorithm improves on the original by an accuracy of roughly $\frac{0.78}{m}$ (Flajolet & Martin 1985). Flajolet & Martin (1985) also define the standard error, the quotient of the standard deviation of an estimate of n by the value of n, and the bias, the quotient of the estimate of n by the exact value of n, for large n.

2.2 Durand-Flajolet algorithm

The LogLog algorithm holds many similarities with the PCSA, chiefly the properties regarding the parameter R (Durand & Flajolet 2003). The idea behind the LogLog algorithm is to create the best general approach to the problem of cardinality. In any random stream of integers, if expressed in binary, approximately a half of the numbers will start with 1, a quarter with 01, and one-eighth with 001. In each case, the cardinality is more likely to be 2, 4, and 8, or $2^{1,2,3}$, respectively. In other words, if the observer reads a binary in 01, we may assume there exists about 25% data points in this stream, so the cardinality has a higher chance to be 2^2. Thus following this logic,in any random data stream, we expect, a cardinality of 2^k, where k is the number of zero bits of hash to

use as a bucket. We then use the size of the longest sequence from the hash to estimate the number of distinct elements. A problem to notice is that the result will not be sufficiently accurate since there is only one number from each size $log(k)$.

To produce a better estimate, we separate our hash function into m groups, called "buckets," where m is the design parameter. We set $m = 2^k$ and set the first k bits of a string x, representing in binary the index of a bucket (Durand & Flajolet 2003). R is computed after discarding the first k bits. Then, if $M^{(j)}$ is the (random) value of parameter R on bucket number j, the arithmetic mean approximates $\log_2(\frac{n}{m})$ plus an additive bias. Durand & Flajolet (2003) detail how to compute the bias as well.

The relative error for LogLog is $\frac{1.30}{m}$, compared to the PCSA above–but outperforms by around a factor of 3 because it replaces "words" of 16 or 32 bits with small bytes of typically 5 bits each (Durand & Flajolet 2003). Throughout the script, there is a hash function h available that transforms elements of the data stream into sufficiently long binary strings, in such a way that they resemble random uniform independent bits (Durand & Flajolet 2003).

3 Results

3.1 Probabilistic Counting

We implement the PCSA algorithm using an m value from 2^1,...,2^{10}. We arrive at the following values from Table 1,

m	% rel. err
2	61.0
4	40.9
8	28.2
16	19.6
32	13.8
64	9.7
128	6.8
256	4.8
512	3.4
1024	2.4

Table 1: Relative error and values used for m, the number of bitmaps used.

The required memory goes as $\mathcal{O}(\log m)$ with m unique elements stored to memory. Run time is $\mathcal{O}(n)$, the size of the data stream. From our above tests, we see that the standard error is approximately,

$$\sigma \approx \frac{0.78}{\sqrt{m}}.$$

For this algorithm, we find the magnitude of expected counts N is proportionate to 2^R, where R is the maximum tail length seen in the stream. Thus from our experiments we established the theoretical bounds of the experiment derived from Flajolet & Martin (1985).

The textbook states that using 2^R is the actual estimate, but this levels off in practice–a higher m will not lead to a lower relative error. This is because the algorithm is a more complex than the text's, and the actual estimate should look like,

$$E(A) \approx \log_2 \phi n, \tag{3}$$

as in the Introduction. We followed the algorithm given in the instructions, wherein we simply look at the highest amount of trailing zeroes in a stream and set that value for R. It's not correct because data streams could introduce outliers that ruin the estimation.

3.2 LogLog Counting

We implement the LogLog algorithm in a variety of experiments to arrive at an understanding of the relationship between the magnitude of the cardinality, amount of memory required, and the relative approximation error. We experiment with a variety of numbers of buckets to show that our distribution of cardinality is Poisson-like, as seen graphically in Durand & Flajolet (2003). With an increase of buckets m, we get a smaller standard error, going as $E \approx \frac{1.30}{\sqrt{m}}$, where $m = 2^k$ number of buckets. We run multiple experiments for the number of buckets 2^0,...,2^{10}. The required memory is $\mathcal{O}(\log \log \frac{n}{m})$ with the run time still at $\mathcal{O}(n)$. Thus, a substantial memory requirement is easily overcome–with a slightly larger margin of error. This algorithm in practice typically only requires about 1KB of memory. In Table 2 we can see the Poisson-like distribution. The

m	% rel. err
0	16.7
1	70.6
2	40.0
3	40.0
4	33.0
5	25.7
6	16.5
7	13.4
8	8.3
9	7.4
10	3.2

Table 2: The number of buckets m and corresponding relative errors.

error is appropriately high for smaller data sets because the variance is too high to estimate unique values correctly. With a larger number of buckets, the variance is much lower.

As a general guide, the algorithm works as thus,

1. Generate a large data stream of random 32-bit integers.
2. Explore parameter space for $m = 2^k$, experimenting with the number of buckets between 2^0 and 2^{10}.
3. Average the experiments (for values of k); each set is a "bin"

4. Calculate relative error for each "bin"
5. Find error fit (should scale with value found above)
6. For optimal results, run at least 100 experiments for each set to eliminate the influence of outliers

4 Conclusion

In this assignment we set out to implement two probabilistic counting algorithms on a large data stream. We note that the LogLog algorithm is a markedly improved algorithm compared to PCSA in memory usage. Using PCSA, a particularly unique and large data stream (a la FaceBook, Twitter, Google Search traffic) can lead to substantial run times and memory time requirements. So, with the substantial improvements in memory allocation and marginal decrease in error precision, it is more appropriate to use the LogLog algorithm.

We suggest using updated versions of these algorithms, including Super-LogLog and Hyper-LogLog, as they reduce error and keep memory allocation minimal. Practical applications are abundant in data science: including marketing, where user traffic creates extremely high volume data streams and the idea is to analyze that traffic and give users relevant data back. Websites can implement these algorithms to measure the number of distinct users to monitor growth.

References

[1] Durand and Flajolet. 2003.

[2] Flajolet and Martin. 1985