Advances in Data Mining - Assignment 2

Counting distinct elements in practice

Abstract

We experiment with two algorithms to estimate the cardinality of a stream of data; the Probabilistic Counting with Stochastic Averaging (PCSA, Flajolet and Martin, 1985) and the LogLog algorithm (Durand and Flajolet, 2003). We vary different variables in our experiments to test the performance of the algorithms. The PCSA requires using many hashing functions (H the number of hashing functions used). For the PCSA algorithm we vary the maximum cardinality N_{max} and the number of groups that we average the results of the hashing functions over. No clear relationship between the Relative Approximation Error (RAE) and the varied parameters could be found for this algorithm. The memory used for this algorithm is $\mathcal{O}(\mathrm{H} \log_2 N_{max})$. For the LogLog algorithm we vary the maximum cardinality and the number of buckets m that we divide the elements of the data stream into. We find the reported relationship between the REA and the number of buckets by Durand and Flajolet (2003) of $RAE = \frac{1.30}{\sqrt{m}}$. The memory of this algorithm is only $\mathcal{O}(m \cdot \log_2 \log_2 N_{max})$. We therefore recommend to use the LogLog algorithm to estimate the cardinality of a stream of data. To implement this algorithm, one first determines the maximum cardinality (N_{max}) of the data stream and the desired accuracy and consecutively the necessary amount of buckets (m) can be calculated. The minimum bit-size of the hash function used should be at least $\log_2(N_{max}) + \log_2(m)$.

1 Introduction

A consequence of the technological era that we live in, is that there is much more data, with much larger variety and in a much higher velocity. Data with these characteristics are often referred to as Big Data. Extracting useful information from Big Data requires some ingenuity. For example, in classical data mining we assume all data that we use can be stored (in a database). However, it can occur that data arrives so rapidly that it is not feasible to store it and then process it. In these cases, the data stream must be processed immediately. Examples of such data streams include satellite data and internet traffic on websites such as Google or Yahoo.

There are several different approaches to extract useful information from a data stream, all summarizing the stream. One might extract samples from a stream, filter out the undesirable elements of the stream, look at the last n elements of a stream, or estimate the number of different elements in a stream. In this research we will focus on the last approach.

If we want to know the number of different elements in a stream, the *cardinality*, we can of course count them. This should not be too difficult as we can keep a list in main memory with the distinct elements we have seen in the stream. However, when the number of distinct elements in the stream becomes too high, it does become problematic. We must therefore use estimates of the number of distinct elements in the stream.

In this research we use the following algorithms to approach an estimate of the number of distinct elements in the stream: the Probabilistic Counting Algorithm (Flajolet and Martin, 1985) that uses patterns in the binary representation of a collection as an indicator for the number of distinct values in that collection and the Log-Log Algorithm (Durand and Flajolet, 2003).

In Section 2 we describe our methods used, in Section 3 we describe the results of the PCSA and LogLog algorithms, in Section 4 we recommend one of the algorithms and how it should be used and in Section 5 Section 6 we provide a conclusion and discussion.

2 Method

Suppose we have a data stream with N distinct elements. The first step of both algorithms is to hash each element a that we see in our data stream to a bit-string[1] $h(a)$ using hashing function[2] h. This bit string must of course be long enough to store all elements of the data stream in a different bit string. An important property of hash functions is that they always produce the same result when applied to the same element. Note that in our experiments, we will not use hash functions but simply randomly generate $b = 32$ random bits from the same random seed. This yields equivalent results as when we would apply a hash function to an element of a data stream.

2.1 Probabilistic Counting with Stochastic Averaging

Probabilistic Counting with Stochastic Averaging (PCSA) (Flajolet and Martin, 1985) is based on the idea that the more different distinct elements are in the data set, the more different hash-values we will see. The algorithm is implemented as follows. Of each element a in the data stream, we count the number of 0's at the end of the bit-string, defined as the *tail length* $r(a)$ of the bit-string. The maximum tail length seen so far in the data stream $R = \max r(a)$ is the only value that is stored. The idea is then that the more distinct elements we have in the stream, the longer the maximum tail length will be. We use

$$\hat{N} \approx 2^R \tag{1}$$

as an estimate of the cardinality of the stream.

Why 2^R works as an estimate for the number of distinct elements makes intuitive sense. Suppose we have a data stream of N distinct elements. The probability that an element a in the stream has a tail length of at least r is then

$$P(R \geq r \mid h(a)) = 2^{-r} \tag{2}$$

The probability that an element a in the stream does not have a tail length of at least r is of course $1 - 2^{-r}$. The probability that no elements in a stream with N distinct elements have tail length at least r is

$$\begin{aligned} P(R \geq r) &= (1 - 2^{-r})^N && (3)\\ &= ((1 - 2^{-r})^{2^r})^{N \cdot 2^{-r}} && (4)\\ &\approx (1/e)^{N \cdot 2^{-r}} \qquad \text{assuming } r \text{ reasonably large} = e^{-N \cdot 2^{-r}} && (5)\\ &= \begin{cases} 1, & \text{for } N \ll 2^r \\ 0, & \text{for } N \gg 2^r \end{cases} && (6) \end{aligned}$$

Therefore we find that 2^R will always be around m and therefore is a good estimate.

We calculate this estimate with g different hash functions. Note that for our experiments, these different hash functions are accomplished by different random seeds. To combine the results of all hash functions, we use Stochastic Averaging. We divide all hash functions in groups of size $2 \log_2 N$ (it can be any multiple of $\log_2 N$, we choose 2). Of each group, we take the average of the estimates of $\hat{N}$, and then of all groups we take the median. This gives us an estimate of the cardinality $\hat{N}$. Finally, we multiply our cardinality by the magic factor $\phi = 0.77352$ (Flajolet; Finch, 2003) to reduce the bias towards larger estimates. An overview of this algorithm can be found in Algorithm 1.

The memory used for this algorithm is $\mathcal{O}(\mathrm{H} \log_2 N_{max})$ with $H = \#groups \cdot 2 \log_2 N_{max}$ the number of hash functions (Flajolet and Martin, 1985).

2.2 LogLog Algorithm

An improvement on Probabilistic Counting with Stochastic Averaging is the LogLog algorithm (Durand and Flajolet, 2003). Instead of using multiple hash functions, which are computationally expensive, we only use one hash function, and split the results of this hash function into buckets. Suppose we want $m = 2^l$ buckets, we will then use the first l bits of the bit string $h(a)$ of element a as the 'bucket-ID' to determine which bucket this element belongs to. The total size of our bit string will be $b = 32$ bits and the remaining $b - l$ bits will be used to determine the tail length $r(a)$ (the number of 0's at the end of the bit string). The value of this tail length will be put in the bucket that the element belongs to. If two elements are put in the same bucket, we only keep the maximum tail length of the two. The final estimate of the cardinality is then returned as follows:

$$\hat{N} = 2^{\frac{\sum_i^m r_i}{m}} \cdot m \cdot \psi \tag{7}$$

[1] A *bit-string* is a sequence of bits.
[2] A *hash function* maps keys to values.

Algorithm 1 Probabilistic Counting with Stochastic Averaging (Flajolet and Martin, 1985)

initialize $\phi = 0.77352$, N, groups, group_length $= 2 \cdot \log_2 N$
//where groups is the number of groups and N is the number of distinct elements
use groups $\cdot$ group_length hash functions
function TRAIL_LENGTH(num)
    $p = 0$
    while (num $\gg p$) & 1 == 0 **do**
        $p+ = 1$
    return p
function STOCHASTIC_AVERAGING(est)
    for g in range(groups) **do**
        group_avg[g] = MEAN(est[g $\cdot$ group_length : (g+1) $\cdot$ group_length])
    return MEDIAN(group_avg)
for h in hash functions **do**
    for a in data **do**
        $r(a)$ = TRAIL_LENGTH($h(a)$)
    $\hat{N}_h = 2^{\text{MAX}(r)}$
$\hat{N} = \phi \cdot$ STOCHASTIC_AVERAGING($\hat{N}_h$)

with $\psi = 0.79402$ (Durand and Flajolet, 2003).

The memory used for this algorithm is typically very low. The equation for the memory used, to which this algorithm also grants its name is $\mathcal{O}(m \cdot \log_2 \log_2 N_{max})$ with m the number of buckets and N_{max} the maximum cardinality.

Algorithm 2 LogLog algorithm (Durand and Flajolet, 2003)

initialize $\psi = 0.79402$, N, l, h, $i = 25$ //with N the number of distinct elements and h a hash function
$m = 2^l$ //number of buckets
function TRAIL_LENGTH(num)
    $p = 0$
    while (num $\gg p$) & 1 == 0 **do**
        $p+ = 1$
    return p
r = [0] $\cdot m$
for i iterations **do**
    for a in data **do**
        bucket = $h(a)$ & $(m-1)$
        bucket_hash = $h(a) \gg l$
        r[bucket] = MAX(r[bucket], TRAILING_ZEROES(bucket_hash))
$\hat{N} = 2^{\text{MEAN}(r)} \cdot m \cdot \psi$)

2.3 Experiments

We will perform several experiments to derive the relation between the magnitude of the expected count, the amount of required memory, and the Relative Approximation Error. The Relative Approximation Error (RAE) is computed as follows:

$$RAE = \frac{|\hat{N} - N|}{N} \tag{8}$$

where $\hat{N}$ is the estimated value of the cardinality and N is the true value of the cardinality. For the PCSA algorithm, we will vary over the number of groups $\#groups$ used, and the maximum cardinality N_{max}

$$\#groups \in \{5, 10, 15, 20, 25, 30, 35, 40\}$$
$$N_{max} \in \{10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$$

For the LogLog algorithm, we vary the amount of buckets m, as well the maximum cardinality N_{max}.

$$m = \{2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}\}$$
$$N_{max} \in \{10^3, 10^4, 10^5, 10^6, 10^7, 10^8\}$$

In both cases we use $b = 32$ bits hash functions, and we analyse the RAE of all different combinations of the above mentioned parameters. Each combination of m and N_{max} is tested 25 times, after which the mean is taken as our actual estimate N to calculate RAE.

3 Results

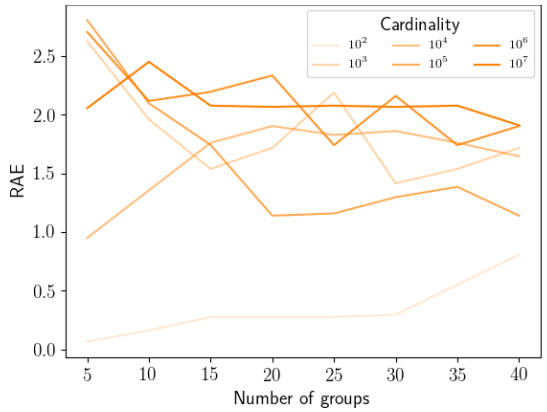
3.1 PCSA algorithm

The results of the experiments for the PCSA algorithm can be seen in Figure 1. They are shown in two different ways. In Figure 1a the Relative Approximation Error (RAE) is shown as a function of the number of groups for different maximum cardinalities. In Figure 1b the RAE is shown as a function of the maximum cardinality for different numbers of groups. Figures 2a and 2b display what Figures 1a and 1b should look like, using the following equation (Flajolet and Martin, 1985):

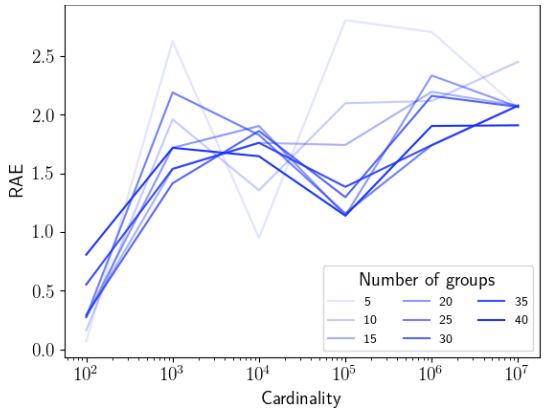
$$RAE = \frac{0.78}{\sqrt{\mathrm{H}}} \tag{9}$$

with H the number of hash functions.

From Figure 1a we conclude that the RAE seems to decrease a little in general when increasing the number of groups, although this effect is not very clear and varies per maximum cardinality. For each cardinality, there seems to be a different relation between the RAE and number of groups. The lowest RAE we reach for cardinalities $N_{max} \leq 10^4$ is $RAE \approx 1.0$ which is not at all comparable to what we observe in Figure 2. Therefore we conclude that PCSA does not achieve any good results for large cardinalities.

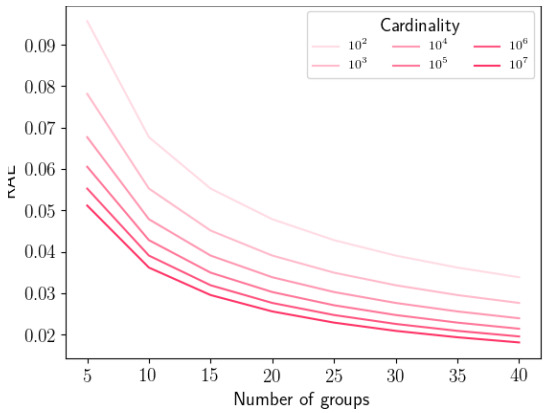


(a) Relative Approximation Error vs. the amount of groups for different values of the maximum cardinality using the PCSA algorithm

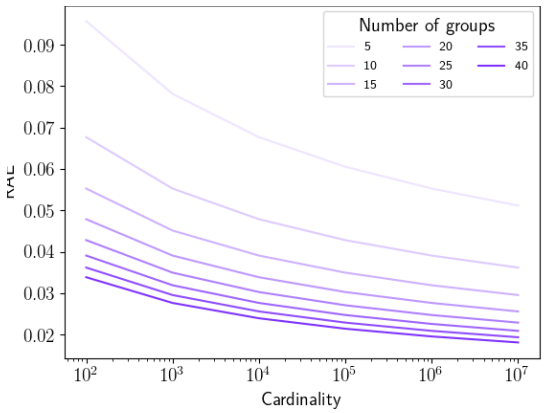


(b) Relative Approximation Error vs. the maximum cardinality for different amounts of groups using the PCSA algorithm

Figure 1: Results of the PCSA algorithm. The Relative Approximation Error (RAE) is shown as a function of either the number of groups (left) or the maximum cardinality (right).



(a) Theoretical Relative Approximation Error vs. the amount of groups for different values of the maximum cardinality



(b) Theoretical Relative Approximation Error vs. the maximum cardinality for different amounts of groups

Figure 2: Theoretical results of the PCSA algorithm using Eq. 9. The Relative Approximation Error (RAE) is shown as a function of either the number of groups (left) or the maximum cardinality (right).

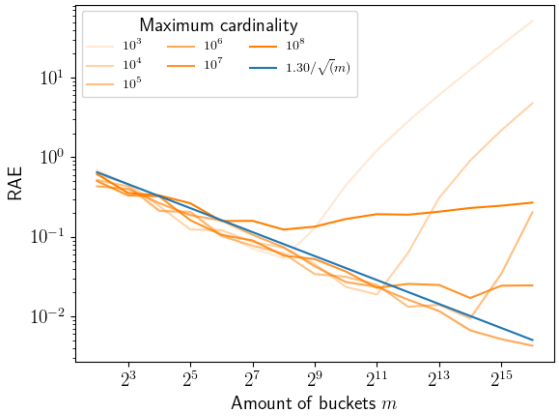
3.2 LogLog algorithm

The results of the experiments for the LogLog algorithm can be seen in Figure 3. They are shown in 2 different ways as well. In Figure 3a the Relative Approximation Error (RAE) is shown as a function of the number of buckets for different values of the maximum cardinality. In Figure 3b the RAE is shown as a function of the maximum cardinality for different numbers of buckets. It is clear that both an increasing amount of buckets and increasing the cardinalities has a positive effect on the RAE, up to a certain point. Initially, the RAE decreases with relation (Durand and Flajolet, 2003)

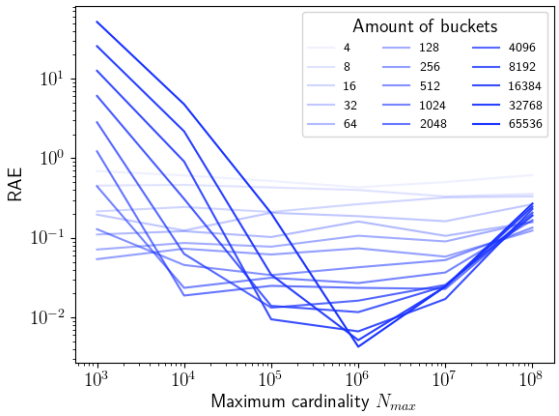
$$RAE = \frac{1.30}{\sqrt{m}} \tag{10}$$

Thus, the higher the number of buckets, the lower the RAE. However, in Figure 3a, for the lowest three cardinalities $N_{max} = \{10^3, 10^4, 10^5\} \approx \{2^{10}, 2^{13.3}, 2^{16.6}\}$ we observe a cut-off point. These cut-off points are at $B \approx \{2^8, 2^{11}, 2^{14}\}$ respectively. The average number of elements per bucket is there $N_{max}/m \approx 4 - 6$. When we increase the number of buckets, the elements will be distributed over a too large number of buckets. Some buckets might end up with zero elements in them, and their value will remain zero. This has a large effect on the estimated cardinality, as can be observed by the steep increase in RAE in figure 3a, making the estimated cardinality more inaccurate.

For the largest cardinalities, there will eventually be a combination of amount of buckets and cardinality that will surpass the 32-bits available in such a way that it will also make the estimated cardinality more inaccurate. This can be observed for $N_{max} = \{10^7, 10^8\} \approx \{2^{23.3}, 2^{26.6}\}$. Suppose we want an accuracy of 1 %, then we will need $m \approx 2^{14}$ buckets. For this number of buckets, $l = 14$ bits will be used to determine the bucketID of an element and the remaining $32 - 14 = 18$ bits will be used to determine the tail length (32 because we have used 32-bits hash functions). However, our maximum cardinality is $N_{max} \approx \{2^{23.3}, 2^{26.6}\}$ and therefore, 18 bits is by far not enough to accurately determine the tail length. Therefore we see an increase in RAE in figure 3 for the two highest cardinalities and thus these maximum cadinalities have lower accuracies.



(a) Relative Approximation Error vs. the amount of buckets for different values of the maximum cardinality using the LogLog algorithm. In blue, the theoretical relationship from Equation 7 is shown.



(b) Relative Approximation Error vs. the maximum cardinality for different amounts of buckets using the LogLog algorithm.

Figure 3: Results of the LogLog algorithm. The Relative Approximation Error (RAE) is shown as a function of either the number of buckets (left) or the maximum cardinality (right).

3.3 Comparison of the algorithms

The LogLog algorithm clearly gave better results in our experiments. Even for relatively large cardinalities, the amount of memory necessary is very small and good accuracy can still be achieved. Using a single hashing function and dividing the elements into different buckets is a far better option memory-wise than using multiple hashing functions and dividing those into groups.

4 Recommendations

When trying to estimate the cardinality in a stream, we recommend using the LogLog algorithm (or one of its successors) as this gives accurate estimates while using only a very limited amount of memory. To determine the parameters to be used with this algorithm, one should first make an estimate on the maximum cardinality N_{max}. When this has been estimated, you should decide what level of accuracy you want to achieve with your estimation. The amount of necessary buckets to achieve the desired accuracy can then be decided from Equation 10. It is possible to use more buckets than this amount to decrease the error on your estimate. You must make sure however, that the amount of buckets is at least a factor 2^3 lower than N_{max}. To determine the minimal size of the hash function necessary, you should look at the number of buckets m, or more precisely $l = \log_2 m$, and the number of bits to count the trailing zeroes $t = \log_2(N_{max})$. In general, these 2 values added together $(t + l)$ should not exceed the bit-size b of your hash functions.

5 Conclusion

We have implemented the Probabilistic Counting with Stochastic Averaging (PCSA) algorithm and varied over the number of groups that the different hash functions were divided into, and over the maximum cardinality. The relationship between the total number of hash functions used and the Relative Approximation Error (RAE) should be $RAE = \frac{0.78}{\sqrt{H}}$ (Flajolet and Martin, 1985). However, this relation could not be reproduced for a large cardinality ($N_{max} \leq 10^4$). For a large cardinality, the minimum RAE that could be reached was $RAE \approx 1.0$ and the memory used for this algorithm is $\mathcal{O}(\mathrm{H} \log_2 N_{max})$.

Much better results were achieved by the LogLog algorithm. For the LogLog algorithm we have varied over the number of buckets that the elements were divided into, and over the maximum cardinality. The theoretical relationship of $RAE = \frac{1.30}{\sqrt{m}}$ could be reproduced and the memory used for this algorithm is $\mathcal{O}(m \cdot \log_2 \log_2 N_{max})$. For large cardinalities, one must keep in mind to use a large enough hash function. The minimum bit-size of the hash function used should be at least $\log_2(N_{max}) + \log_2(m)$.

6 Discussion

After the publication of the LogLog algorithm, improvements have been made on the algorithm. By taking out 30% of the largest estimates, the accuracy can be improved from $\frac{1.3}{\sqrt{m}}$ to $\frac{1.05}{\sqrt{m}}$. Furthermore, taking the Harmonic mean instead of the geometric mean that we took, the accuracy should increase as well, giving the HyperLogLog (HLL) algorithm with an accuracy of $\frac{1.04}{\sqrt{m}}$. The algorithm can also be trivially parallelized to speed it up.
We recommend using the LogLog algorithm (or one of its derivatives) and not the PCSA algorithm. It is superior in terms of both memory and time used and can be used for very large data streams.

Bibliography

Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617. Springer, 2003.

Steven R Finch. *Mathematical constants*, volume 93. Cambridge university press, 2003.

Philippe Flajolet. Counting by coin tossings. In *Algorithms Project*. INRIA-Rocquencourt.

Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.