Counting Distinct Elements

Advances in Data Mining. Assignment 2

Abstract

In this report we implement three different algorithms to count the distinct elements in a data set: probabilistic counting, an algorithm as described in the book *Mining of Massive Datasets* and LogLog counting. All three methods are largely based on the Flajolet-Martin algorithm. The data set contains 10^5 elements made out of 32 bits. We find the algorithm as described in the book to have the worst performance and LogLog counting to have the best. The fact that LogLog has a better performance than probabilistic counting is in disagreement with literature. However, the LogLog counting has a Relative Approximate Error in agreement with the theoretically expected value of $\sim$4% for our setup, adding confidence to the implementation of this algorithm.

1. Introduction

As Gordon Moore predicted in 1965 with his famous law about the number of transistors in a dense integrated circuit, computational hardware has been improving at a constant rate for the last few decades. This means that the computers are getting faster and capable of processing more data in less time. One disadvantage of this, is that it has created a tendency to deal irresponsible with huge amount of data, not taking into account the memory or computational time needed to process it. To still be able to process large waves of information in a reasonable time and with a reasonable system, programmers have recurred to smart algorithms that deal with concrete problems when working with enormous data sets.

One of these problems is the counting of distinct elements. If you want to know the number of distinct words in a paper, it is not a big problem. But if instead of one paper, you want to count the distinct words in a large book, or maybe even a thousand books, an algorithm that scales well with the amount of data is required. Over the years, various different methods have been de-

veloped to solve this problem. They all try to find the best balance between the accuracy with which the distinct words are counted and the memory and computational time needed. In this report we implement and discuss the results presented by the following three different algorithms used to count the distinct elements in a data set: probabilistic counting, an algorithm as described in the book *Mining of Massive Datasets* (we will refer to this as the MMD algorithm) and LogLog counting.

2. The data set

In the general case we would have a data set $\{x_i\}_{i=1}^{n}$ with m distinct elements. The elements could represent various things, for example words in a document, temperature measurements of a sensor array or items in stock of a store. In the end, we don't really care about the original data format, we just want to count the number of distinct elements. Therefore we use a very general algorithm which can be applied to all of the mentioned examples. To achieve exactly this, a hash function is applied to the data, generating a random list of binary numbers. The desired properties of this list is still the same for a well-behaved hash function: it will also contain n binary numbers of which m are distinct.
Rather then working with a 'real' data set, we will work directly with the list of binary numbers. In other words, as if a hash function was applied to a 'real' data set. The generated data set consist out of one 2-dimensional matrix with shape $(100000, 32)$. This matrix simulates the result of hashing 10^5 objects (words in a book for example). Every hashed element contains 32 bits (1 or 0), filled randomly. Note that the maximum amount of distinct elements is equal to $2^{32} = 4294967296$, which is significantly higher than the total amount of elements.

3. Methods

We make use of three different algorithms to count the distinct number of elements in the data set. Firstly, the algorithms are described in Section 3.1. Secondly, Section 3.2 describes how we evaluate the performances of the algorithms. Lastly, an overview of our system parameters is given in Section 3.3

3.1. Our algorithms

All algorithms are largely based upon the Flajolet-Martin algorithm [2]. First assume one has a uniformly distributed data set of binary numbers of m distinct elements. In this case the probability that the last number is a zero equals 2^{-1}. In the case of the last two number it would be $2^{-1} \times 2^{-1} = 2^{-2}$. In general the last r numbers have a probability 2^{-r} to be all zeros. For our data set we would expect that the longest tail of zeros R satisfies $2^R \approx m$. Turning this argument the other way around: if we measure the longest tail of zeros is of length R, we would estimate the amount of distinct elements in the data set to be 2^R. This property is also supported mathematically: In our data set of R distinct elements there would be a probability $(1 - 2^{-r})^m$ no element has a tail length of r. Now as m becomes much larger than r this probability goes to zero. Conclusively, 2^R is indeed a good estimate for the number of distinct elements in a data set, provided $m \gg r$.
However, using this method the result will be biased. This is due to the fact that whether we observe r at probability p, $r + 1$ would be observed at a probability of at least $p/2$, but the estimate of the amount of distinct values will always double. Therefore, we will overestimate the number of distinct elements. There are different ways to deal with this issue. In the following sections we will discuss our three algorithms and their differences regarding how to address this problem.

3.1.1. Probabilistic counting

Probabilistic counting is the way originally proposed in the paper of Flajolet and Martin [2]. They noted the bias towards larger estimates of the total number of distinct elements. Without going into mathematical depth of their analysis, they derived a magic number, $\phi = 0.7735162909$, which corrects for this bias. In practice, if one finds the longest tail of zeros is of length R the will not be 2^R, but $\phi \cdot 2^R$ instead. Since probabilistic counting stores the longest tail of zeros as a binary number and this is also a logarithmic estimator, the memory needed is of the order $\mathcal{O}(\log \log n)$ with n the total number of elements in the set. In time it scales as $\mathcal{O}(n)$, since hashing is the most time-consuming step and we need to hash all n elements. The relative accuracy of the estimator of probabilistic counting compared to the actual amount of distinct elements was estimated in the original paper as $0.70/\sqrt{n_{\text{bits}}}$ where $n_{2^{\text{bits}}}$ are the number of bits used.

3.1.2. MMD algorithm

This counting algorithm is described in sections 4.4.2 and 4.4.3 of the book *Mining of Massive Datasets* [3]. It uses yet another way of dealing the bias of one estimator: using multiple estimators instead. At first one can think of using multiple different hash functions $h_1, h_2, ..., h_k$, applying all of them upon the data, counting the best estimates $R_1, R_2, ..., R_k$ and averaging all of them. Since using multiple hash functions can be time and memory consuming as well, we use one hash function, but use the first 10 bits as a label. Data with the same label belongs to the same 'bucket'. The longest tail of zeros is counted for all buckets and averaged.
However, as noted in the book this method suffers a defect: the mean of multiple measurements will still overestimate the number of distinct elements. Another option would be to use the median, which does not suffer this problem. However, since all estimates are a power of two, so will the median. Therefore, the best way is actually to combine both desired properties. First group our buckets into subgroups, then average their estimates and finally take the median. To ensure the right average can be determined the groups should be larger than at least $\log^2 m$.
Since the MMD algorithm determines the estimators of smaller subgroups, the buckets, the stored estimators are smaller in size. Therefore, on average the memory needed is about 4 times less than for probabilistic counting.

3.1.3. LogLog counting

The idea of LogLog counting was first proposed by [1]. The idea is a bit of a combination of the ideas of the two previous algorithms. In the LogLog algorithm the data is also labeled and divided into k different buckets. The counts for the longest tails of zeros for all buckets are averaged to get a best estimate for the longest tail of zeros per bucket R'. The best estimate would thus be $2^{R'} \times k$, however this would still be biased. Therefore, additionally they multiply by the magic number ϕ to obtain the final estimate for the number of distinct elements in the data set.
Since LogLog counting also uses buckets, it scales similar in size and memory as the MMD algorithm. The relative accuracy of the estimator of probabilistic counting compared to the actual amount of distinct elements was estimated in the original paper as $1.30/\sqrt{2^{n_{\text{bits}}}}$ where n_{bits} are the number of bits used.

3.2. Evaluation of the algorithms

An important measure of the performances of the algorithms are their relative accuracy. This is encaptured by the Relative Approximation Error (RAE) defined as

$$RAE = \frac{|true\ count - estimated\ count|}{true\ count}. \tag{1}$$

Multiple experiments are performed to determine the RAE with higher accuracy: in this work a total of 50 experiments were performed to measure the average RAE. Note not only the accuracy of an algorithm is important, but also how long it takes in time to run the algorithms. Therefore, we also measure the time to run the algorithms.

3.3. System parameters

All the codes and data reduction where performed in the same computer with characteristics as summarized in Table 1.

Architecture	x86-64
Operating System	Fedora 25 Twenty Five
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	4
Model name	Intel(R) Core(TM) i5-6500 CPU 3.20GHz
Memory	7.7 GiB
Hard drives	463.7 GB

Table 1: Our system parameters.

4. Results

All three algorithms ran 50 times. The random seed used each run to fill the hash values with random bits is equal to the number of the run, starting at zero. So the first run had a random seed of 0, the second one had a random seed of 1 and so on.
An overview of the average of the measured RAE's and the time needed to run the algorithms once are shown in Table 2.

Furthermore, Figures 1 and 2 show the histograms of the average RAE's for all three methods combined and respectively for each one individually.

Algorithm	Average RAE	Average time (s)
Probabilistic counting	0.3999	0.7733
MMD algorithm	0.9987	0.8023
LogLog counting	0.0390	0.8016

Table 2: Average RAE and time needed to compute all three algorithms.

It can be seen that the spread of the LogLog method is smaller than the one from the probabilistic counting. Note that the figures show immediately that the results for the MMD algorithm are far from good and present unexpected (and inexplicable) characteristics, indicating that the algorithm performs poorly.

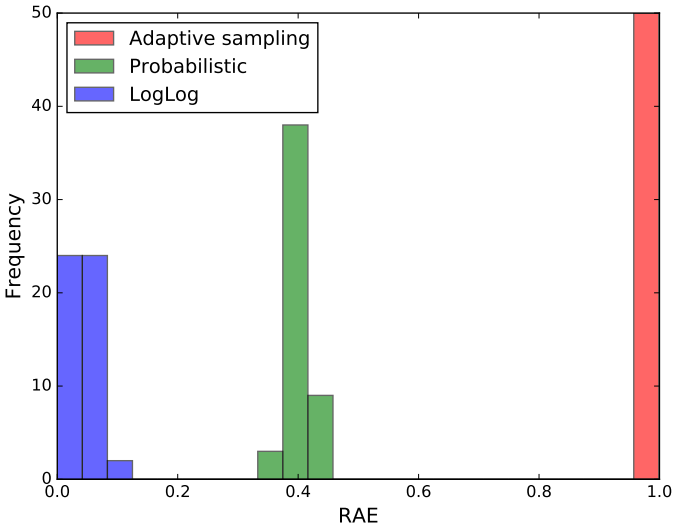

Figure 1: Histogram of the RAE's found for all three algorithms.

5. Discussion

The average RAE taken from the 50 runs gives us a good impression of the accuracy of the tested algorithms. For the probabilistic counting we expected a theoretical accuracy of $0.78/\sqrt{2^{nbit}} \approx 2\%$. This is not what we obtain (see Table 2), suggesting there might be a fault in our implementation

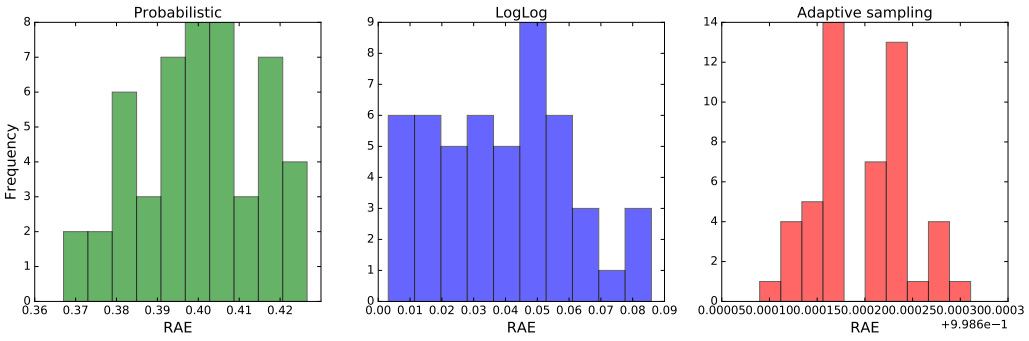

Figure 2: Histogram of RAE's for every algorithm individually.

of the algorithm. Similarly, the expected theoretical standard error of the LogLog counting is $\sim$4%. This is very close to the value obtained. The idea that the implementation of the probabilistic counting may contain an error is reinforced by the fact that its RAE value is higher than the one from the LogLog method. Since the LogLog algorithm requires a smaller amount of memory, its estimates should be less accurate resulting in a larger RAE. As expected, the MMD algorithm did return a very high RAE average, suggesting that the algorithm does not work properly.

The computational times for all algorithms are approximately similar. This is as expected, since in our implementations of the algorithm we use about the same order of operations for all three algorithms. Anyway, it is important to note that one should be cautious to draw conclusions from the computational times needed since the algorithms were not coded for an optimal performance. Memory-wise the probabilistic counting needs about 4 times as much memory as LogLog to compute.

6. Conclusions

We implemented three algorithms which estimate the number of distinct elements in our data set. Our main conclusions are:

- The performance of the algorithms from best to worst is: LogLog counting, probabilistic counting and the MMD algorithm.
- The fact that LogLog counting performs better than probabilistic counting is in disagreement with literature.

- LogLog counting has a RAE of $\sim$4%, which agrees with the theoretically expected value, adding confidence to our implementation of this algorithm.
- As expected the computational times of the algorithms are about equal.

References

[1] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *In ESA*, pages 605–617, 2003.

[2] Philippe Flajolet. *Counting by Coin Tossings*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[3] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.