Assignment 3: Cardinality Estimation

Miguel Alcañiz Moya

December 2024

1 Statement of the assignment

In this programming assignment, we will study experimentally the performance of two different cardinality estimation algorithms, namely, Hyperloglog and Recordinality. These two methods give you an approximation of the cardinality of a set based on probability. We will evaluate how good are these for different datasets and to study different variables of their performance such as the error in the approximation or the variance of the results, because as said these are random probabilistic methods.

Clearly, the cardinality of a multiset can be exactly determined with a storage complexity essentially proportional to its number of elements. However, in most applications, the multiset to be treated is far too large to be kept in core memory. A crucial idea is then to relax the constraint of computing the value n of the cardinality exactly, and to develop probabilistic algorithms dedicated to estimating n approximately. A whole range of algorithms have been developed that only require a sublinear memory, or, at worst a linear memory, but with a small implied constant. All known efficient cardinality estimators rely on randomization, which is ensured by the use of hash functions. The elements to be counted belonging to a certain data domain D, we assume given a hash function, that is, we assimilate hashed values to infinite binary strings of $\{0,1\}^M$ for a big M, or equivalently to real numbers of the unit interval.

2 GitHub

The project is delivered in the public GitHub repository with the following link:
$https://github.com/miguelalcaniz02/Cardinality\_Estimation.git$

3 Performance of cardinality estimation algorithms

In this section we will show the result of the performances of the algorithms Hyperloglog and Recordinality to calculate the cardinality of several datasets. To begin with, we will use some real datasets of famaous books such as Robinson Crusoe, Dracula or The Iliad. Then we will see how our algorithms perform with some synthetic data created with Zipfian law.

For the results shown in the section we repeated the experiments 50 times. We used the hash function SHA-256 from the python library hashlib. To create a random result from every run of the same dataset I added in every run a random suffix of two letters, that way every time we get a different hash table and we are able to study the variance of the methods.

We are going to comment briefly the amount of space used in these two methods, as it is one of the advantages of using this cardinality estimation methods instead of studying them the backtracking way. So for the hyperloglog method we have that the algorithm needs to maintain a collection of registers, what we will name k, each of which is at most $loglogN + O(1)$ bits, when cardinalities $\leq$ N need to be estimated. And for the recordinality method we use k hash values ($klog(n)$ bits) and one counter ($log(log(n))$ bits). So we will use for both methods at most of the order of $klog(n)$ bits.

3.1 Real datasets

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	6939	0.111	2770	3453	12653	1.026	8415
8	6341	0.015	3105	2441	5986	0.041	4220
16	6431	0.030	2079	1726	6455	0.034	4462
32	6528	0.045	1388	1181	5900	0.055	2181
64	6284	0.006	727	822	6226	0.003	1661
128	6361	0.019	581	577	5833	0.066	878
256	6242	0.000	363	405	5926	0.051	660
512	6266	0.003	287	286	5751	0.079	355

Table 1: Sample table for HLL and REC with real dataset **crusoe.txt** with 91813 words and cardinality 6245.

We can see that both methods work quite well for bigger k. For the crusoe.txt data we can see that the best k is 256 for the hyperloglog method and 64 for the recordinality. Using bigger k's makes a worst approximation giving a bigger error but with a smaller standard error. We can see as the expected standard error for the hyperloglog method behaves as expected.

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	9278	0.016	6197	5212	5635	0.402	6355
8	9896	0.050	3372	3685	9193	0.025	8927
16	9259	0.018	2374	2606	9514	0.009	5228
32	9390	0.004	1619	1782	9391	0.004	3116
64	9311	0.012	1173	1241	9726	0.032	2137
128	9359	0.007	864	871	9632	0.022	1657
256	9380	0.005	612	612	9199	0.024	942
512	9389	0.004	453	432	8894	0.056	499

Table 2: Sample table for HLL and REC with real dataset **dracula.txt** with 124249 words and cardinality 9425.

For this dracula.txt data we see that for the hyperloglog method we have the smaller error for k = 32 and k = 512, but for the bigger k the variance is much smaller. For the recordinality method the best approximation is for k = 32. The variance in the hyperloglog method performs as expected, getting reduced as k grows. For the recordinality we can see how the standard error is really big for small k's, only starting to get much better from k = 128 on. We can conclude that we get a really good approximation of the cardinality.

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	7191	0.194	4551	4935	10292	0.153	13169
8	9496	0.064	5495	3489	9659	0.082	14439
16	8755	0.019	2656	2467	7834	0.122	3715
32	9363	0.049	1951	1688	8564	0.040	3551
64	9192	0.030	1175	1175	9061	0.015	2411
128	9040	0.013	737	825	8545	0.043	1181
256	9045	0.013	472	579	8849	0.009	926
512	8875	0.006	349	409	8416	0.057	437

Table 3: Sample table for HLL and REC with real dataset **iliad.txt** with 124944 words and cardinality 8925.

For this third and last dataset with similar numbers than the previous one we just observe the same as said before for the other real datasets. The variance gets reduced when k grows for both methods, and as expected for the hyperloglog. And the best option for k is 512 for the hyperlogog method getting an error of 0.6% and k = 256 for the recordinality method getting an error of almost 1%.

In conclusion these two methods have workout really well for the datasets given. We could have sampled more results from other 5 real datasets but I think there is no more information to obtain from these experiments.

3.2 Synthetic Datasets

Now we are going to see the performance of the methods for a produced synthetic data. That is generating a probability distribution over n words and sampling N words over that probability distribution. In these experiments we sampled 100000 words out of 10000 possible ones (sampling 10 times the cardinality). The probability distribution is parametrized by α. We consider the synthetic dataset $Z = z_1, z_2, \ldots, z_N$, generated following a Zipf law where $\alpha \geq 0$, for n different elements $\{x_1, \ldots, x_n\}$, where:

$$P\{z_j = x_i\} = \frac{c_n}{i^\alpha}, \quad 1 \leq j \leq N, 1 \leq i \leq n,$$

donde

$$c_n = \frac{1}{\sum_{1 \leq i \leq n} i^{-\alpha}}.$$

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	9201	0.080	6825	5530	7993	0.201	9720
8	7594	0.241	2971	3910	8931	0.107	8737
16	8035	0.197	2281	2765	10040	0.004	5264
32	8126	0.187	1590	1891	8908	0.109	3732
64	8752	0.125	1259	1317	8324	0.168	2489
128	8629	0.137	819	924	8092	0.191	1324
256	8520	0.148	459	649	8057	0.194	957
512	8419	0.158	319	459	7903	0.210	470

Table 4: Sample table for HLL and REC with synthetic data created following a Zipfian law of parameter $\alpha = 1$ with 100000 words and cardinality 10000.

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	5706	0.429	4734	5530	5348	0.465	8449
8	6479	0.352	2753	3910	4656	0.534	2629
16	5614	0.439	1088	2765	5498	0.450	2806
32	6032	0.397	1348	1891	6114	0.389	2555
64	5890	0.411	840	1317	5918	0.408	1278
128	5992	0.401	549	924	5711	0.429	721
256	5980	0.402	364	649	5710	0.429	496
512	5914	0.409	277	459	5435	0.456	301

Table 5: Sample table for HLL and REC with synthetic data created following a Zipfian law of parameter $\alpha = 1.2$ with 100000 words and cardinality 10000.

We can see that for the two previous datasets the cardinality estimation methods worked quite badly, that may be because the frequency of the words is really variant. So some words appear too frequently and some others too often.

k	HyperLogLog				Recordinality		
	Avg	Error	σ	Expected σ	Avg	Error	σ
4	11538	0.154	8172	5530	7602	0.240	12656
8	10843	0.084	5745	3910	7564	0.244	4481
16	10425	0.043	2385	2765	9080	0.092	3813
32	9974	0.003	1607	1891	9303	0.070	2938
64	9895	0.011	1324	1317	9219	0.078	2197
128	9569	0.043	811	924	9260	0.074	1530
256	9990	0.001	547	649	9791	0.021	1108
512	9849	0.015	448	459	9543	0.046	634

Table 6: Sample table for HLL and REC with synthetic data created following a Zipfian law of parameter $\alpha = 0.6$ with 100000 words and cardinality 10000.

We can see that for this value of the parameter α, both methods determine the cardinality of the set with really good precision. For instance for the hyperloglog method for k equal to 32 or 256 and for the recordinality for k equal to 256. The variance works as expected, getting reduced when k grows and with similar values than the ones calculated for the hyperloglog method.