RA - Cardinality Estimation

Dinu Eduard - Adiel

December 2024

1 Introduction

In this assignment, the task of observing and experimenting was the Cardinality Estimation using two well-known algorithms: HyperLogLog and Recordinality.

In the following sections, we will see the results of experiments conducted with the 2 algorithms with synthetic data and publicly available datasets from Project Gutenberg Free eBooks.

2 Repository link

`https://github.com/adieldinu/cardinality-estimation`

3 Synthetic Data

3.1 Generation

To generate the data, I used the zipfian distribution with the following properties. The **Zipfian distribution** is a discrete probability distribution that is often used to model the frequency of words, events, or objects in a variety of natural and social phenomena. It describes the relationship between the rank r of an item and its frequency $f(r)$ in the data set. Zipf's law states that the frequency of the r-th most common element in a dataset is inversely proportional to its rank raised to a power α.

3.1.1 Formula

The probability $P(r)$ that an item has rank r in a Zipfian distribution is given by:

$$P(r) = \frac{1/r^{\alpha}}{\sum_{i=1}^{n} \frac{1}{i^{\alpha}}}$$

Where:

- r is the rank of the element (i.e., $r = 1, 2, 3, \ldots, n$),

- α is the **Zipfian exponent** (typically a positive constant, often between 1 and 2),
- n is the total number of distinct elements.

The denominator, $\sum_{i=1}^{n} \frac{1}{i^\alpha}$, is a normalizing factor that ensures that the probabilities sum to 1.

3.1.2 Example

Consider a Zipfian distribution with $\alpha = 1$ and $n = 5$. The probability of each rank is given by:

$$P(1) = \frac{1/1^\alpha}{1/1^\alpha + 1/2^\alpha + 1/3^\alpha + 1/4^\alpha + 1/5^\alpha}$$

$$P(2) = \frac{1/2^\alpha}{1/1^\alpha + 1/2^\alpha + 1/3^\alpha + 1/4^\alpha + 1/5^\alpha}$$

$$P(3) = \frac{1/3^\alpha}{1/1^\alpha + 1/2^\alpha + 1/3^\alpha + 1/4^\alpha + 1/5^\alpha}$$

In this case, the rank 1 item will have the highest probability, and the probability decreases as the rank increases.

3.1.3 Visualization

For $\alpha = 1$, the distribution resembles a **harmonic series**, and the rank-frequency plot typically shows a straight line on a log-log scale. As α increases, the distribution becomes more skewed, with a steep drop-off in frequency for higher-rank items.

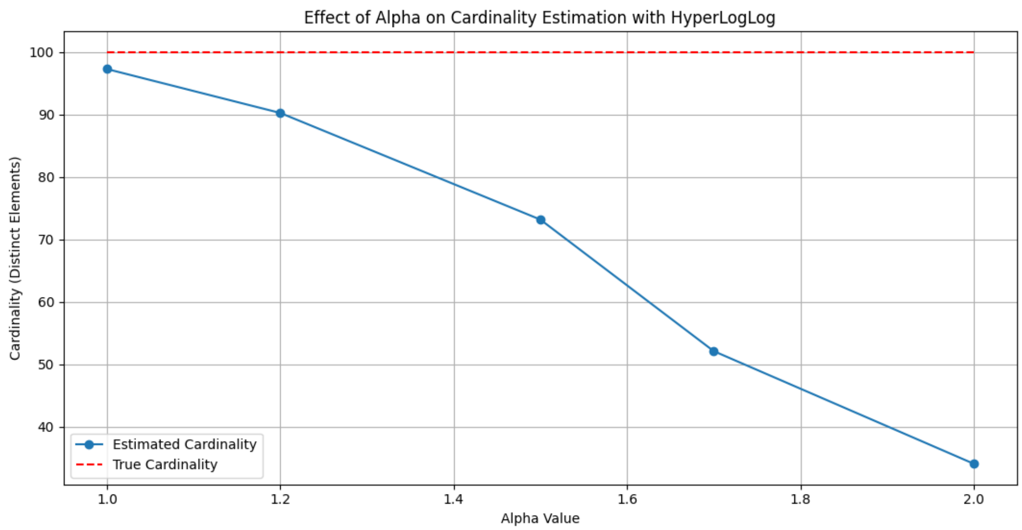
3.2 Experiments with HyperLogLog

In this section, 3 types of experiments have been conducted:

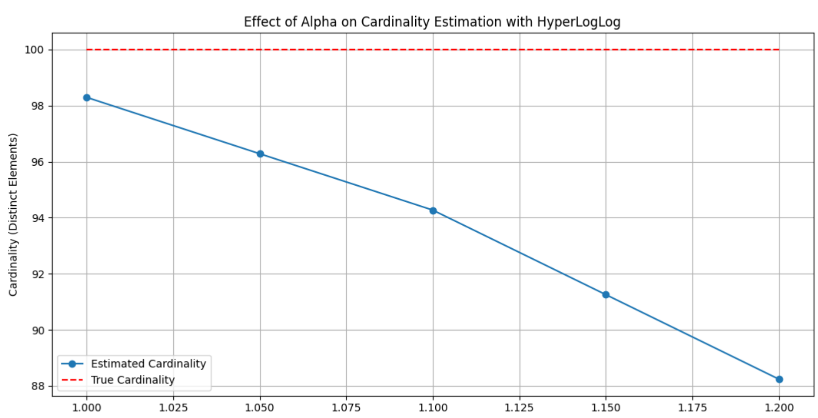
- Experiment 1 $\rightarrow$ Investigating the impact of varying the value of alpha on the algorithm's behavior.
- Experiment 2 $\rightarrow$ Exploring how changing the number of distinct elements, n, affects the algorithm's performance.
- Experiment 3 $\rightarrow$ Analyzing the effect of altering the total number of elements, N, on the accuracy of the algorithm.

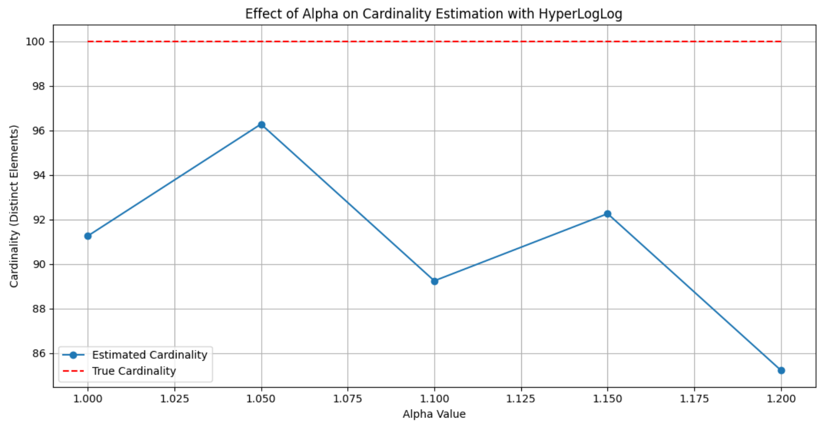
3.2.1 Experiment 1

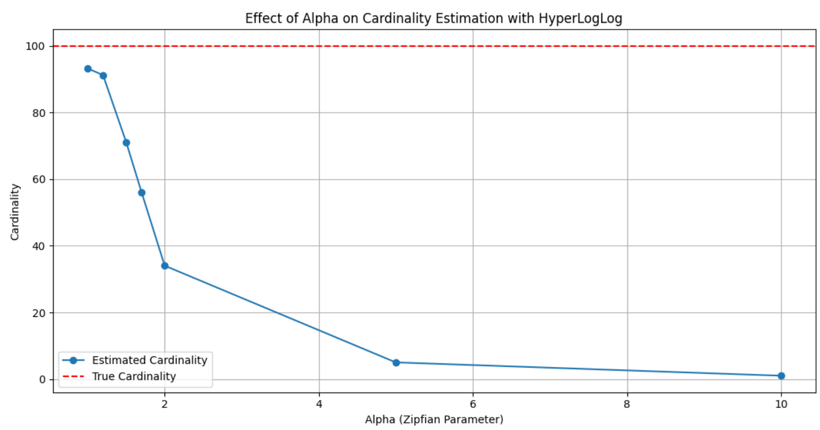
Here are some results of the experiment with the constant $N = 1000$ and $n = 100$, but the alpha passes through these values: $1.0, 1.2, 1.5, 1.7, 2.0$.



From this first experiment, it can clearly be seen that the estimation error drastically increases as alpha increases due to alpha's influence in the occurrence of every term.







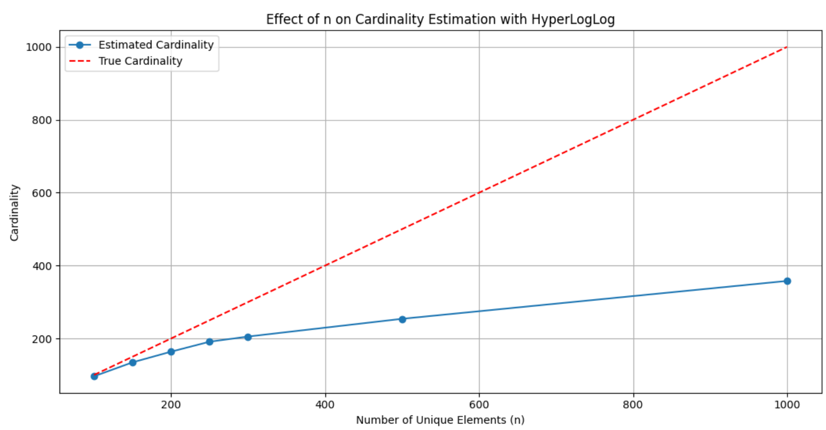
- **Impact on Cardinality Estimation:**
  - Higher values of α lead to fewer distinct elements being observed for a fixed number of total elements, N.
  - A more skewed distribution results in frequent repetition of a small subset of elements, reducing the effective number of distinct elements.
  - For lower values of α, the distribution is more uniform, resulting in a larger number of distinct elements being observed.
- **Algorithm Behavior and Accuracy:**
  - HyperLogLog estimates cardinality by counting distinct elements. It performs better when the data is less skewed (lower α).

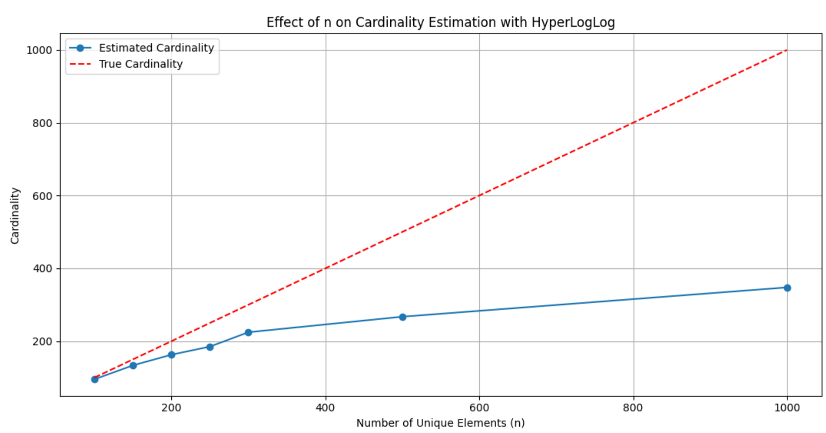
  - As α increases, the algorithm processes fewer distinct elements, potentially leading to slightly less accurate estimations, as seen from the increasing value of α.

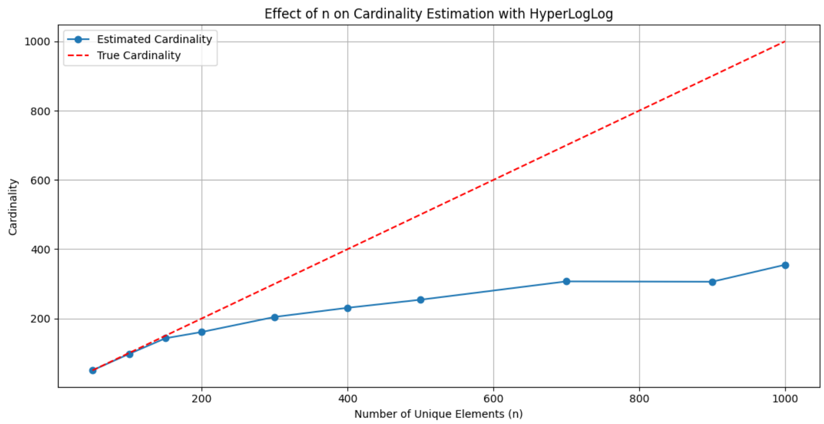
- **Application Insights:**
  - For highly skewed data distributions (e.g., web traffic, social media interactions), setting an appropriate α during experiments helps mimic real-world scenarios.
  - Analyzing different values of α provides insights into the robustness of HyperLogLog in handling varied data distributions.

3.2.2 Experiment 2

Here the variable parameter is n, the number of distinct elements, the idea here is to see what happens from $n = 100$ to $n = N$.







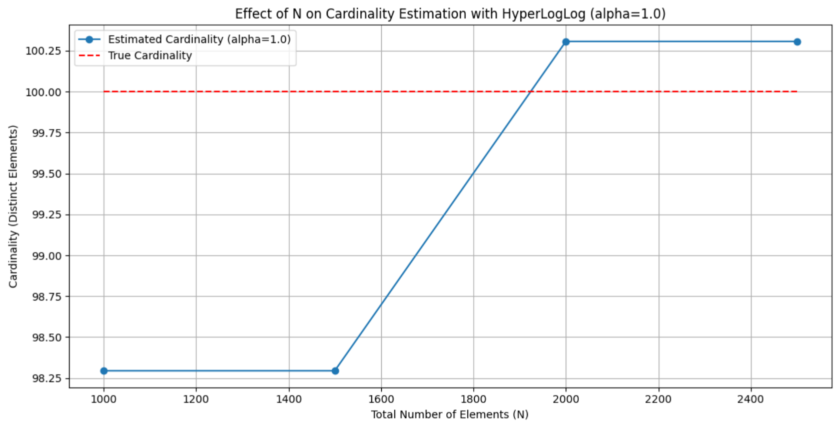
Varying the number of distinct elements (n) highlights the following:

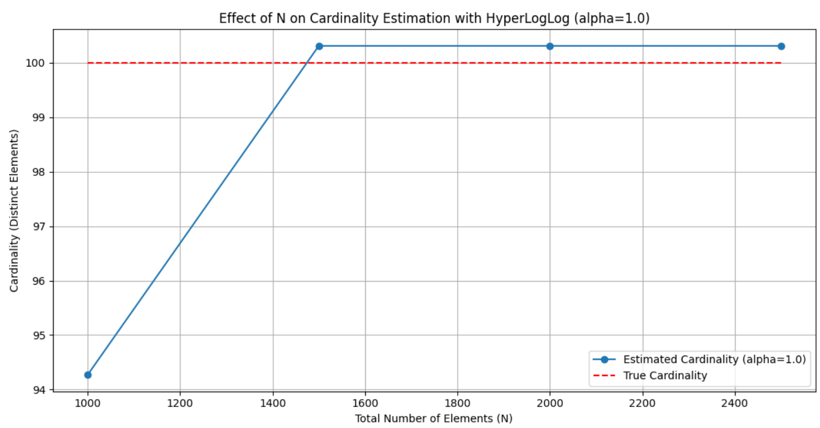
- **Accuracy and Scalability:** HLL maintains high accuracy across a wide range of n, demonstrating its scalability for large datasets.
- **Efficiency:** The algorithm remains computationally efficient, as it only processes hash values instead of storing elements.
- **Limits at Extremes:** Small n can lead to fluctuations due to fewer data points, while extremely large n may reveal minor underestimations due to register limitations.

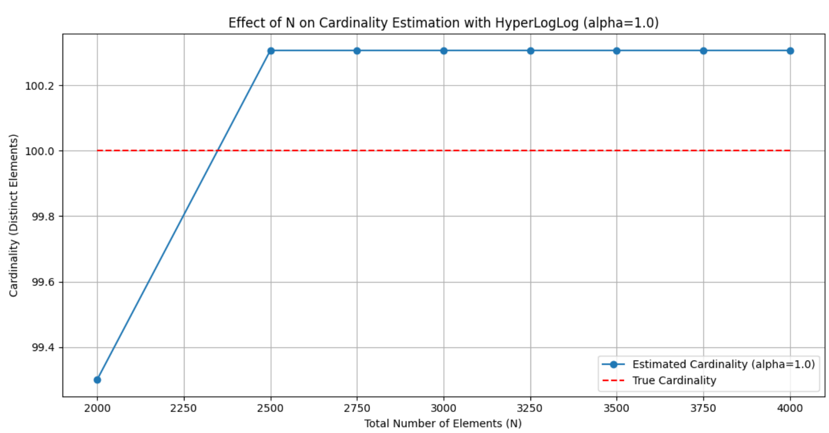
3.2.3 Experiment 3

In this subsection, $n = 100$ and $\alpha = 1$ and N will vary from:

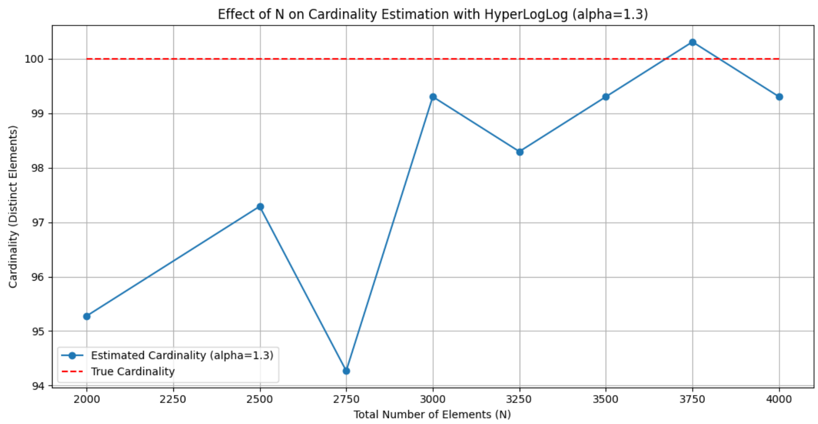
- 1000 to 2500
- 2000 to 4000

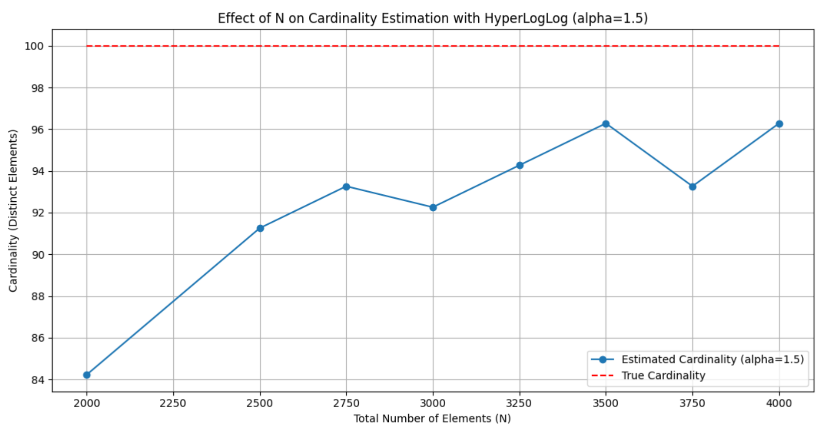

Effect of N on Cardinality Estimation with HyperLogLog (alpha=1.0)
Estimated Cardinality (alpha=1.0)
True Cardinality
Cardinality (Distinct Elements)
Total Number of Elements (N)
100.25
100.00
99.75
99.50
99.25
99.00
98.75
98.50
98.25
1000
1200
1400
1600
1800
2000
2200
2400

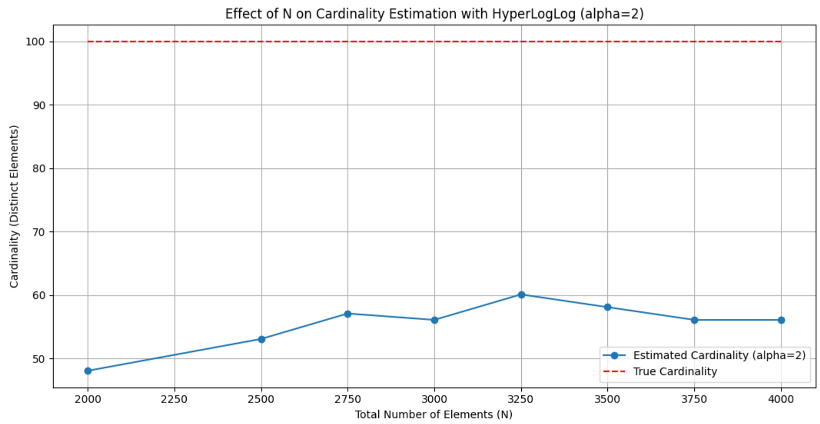

Effect of N on Cardinality Estimation with HyperLogLog (alpha=1.0)
Estimated Cardinality (alpha=1.0)
True Cardinality
Cardinality (Distinct Elements)
Total Number of Elements (N)
100
99
98
97
96
95
94
1000
1200
1400
1600
1800
2000
2200
2400


Effect of N on Cardinality Estimation with HyperLogLog (alpha=1.0)
Estimated Cardinality (alpha=1.0)
True Cardinality
Cardinality (Distinct Elements)
Total Number of Elements (N)
100.2
100.0
99.8
99.6
99.4
2000
2250
2500
2750
3000
3250
3500
3750
4000

Next experiments change the α to 1.3, 1.5 and 2.





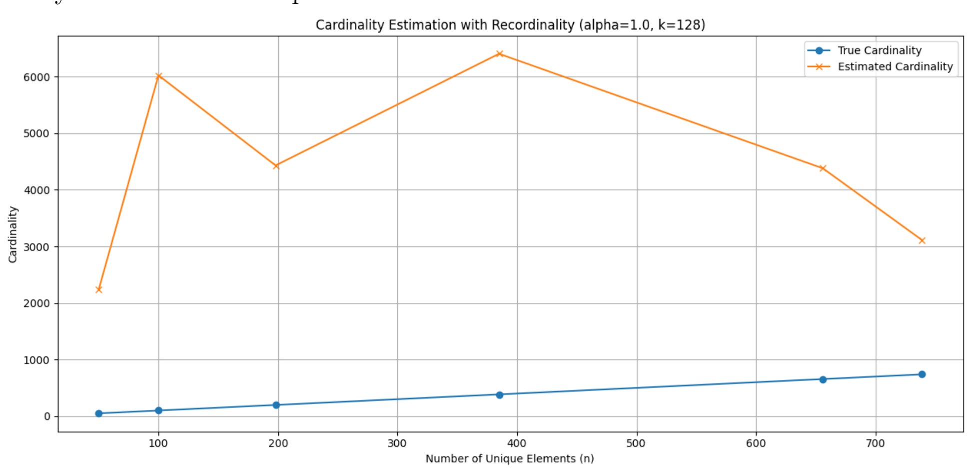


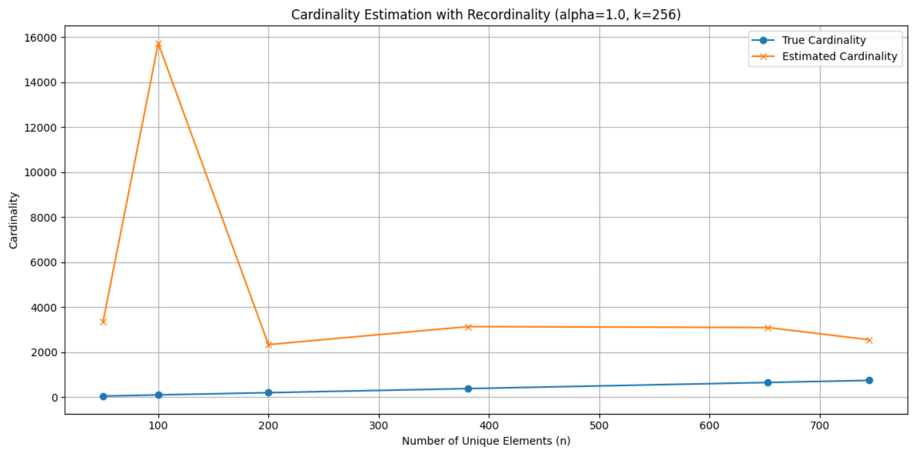
Varying the total number of elements (N) shows that:

- The accuracy of cardinality estimates remains stable as N increases.
- HLL efficiently handles large datasets, ignoring duplicates.

3.3 Experiments with Recordinality

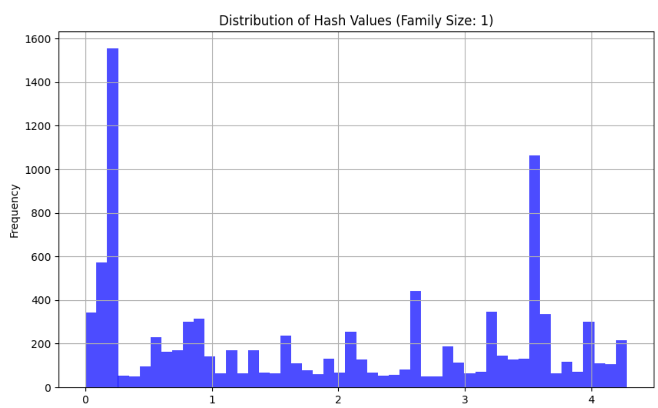
In this subsection, an overall view of the Recordinality will be observed. In the first few runs of the algorithm, the value k was between 128 and 256. It can clearly be seen a small improvement in the estimation due to the increase in k.

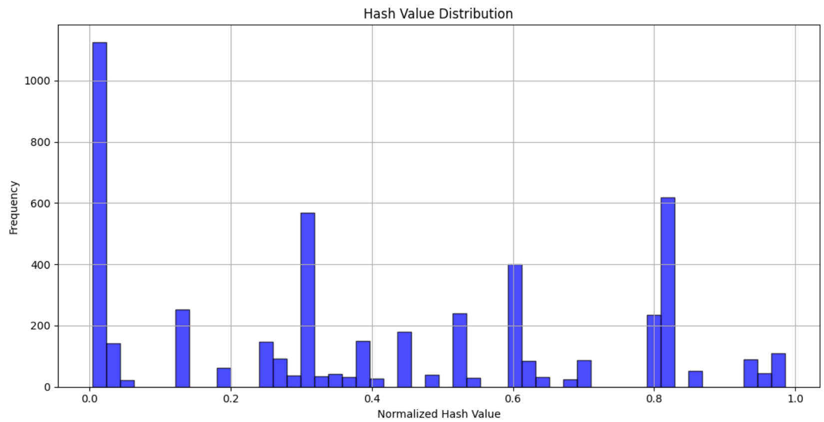




3.3.1 Hash Distribution of the data

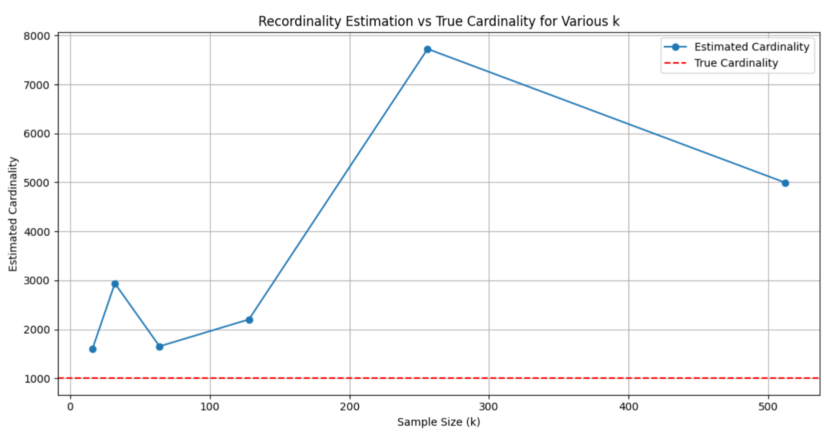
Due to usage of hash functions, I decided to explore a little bit the distribution of the hashes. A uniform hash distribution is crucial for accurate cardinality estimation.
Inaccurate estimations occur if hash values are not evenly distributed.
The effectiveness of the algorithm depends on a good distribution of hash values.
Random hash functions ensure a better spread of values across the hash space.
For high accuracy, the hash values should be well-distributed and independent.

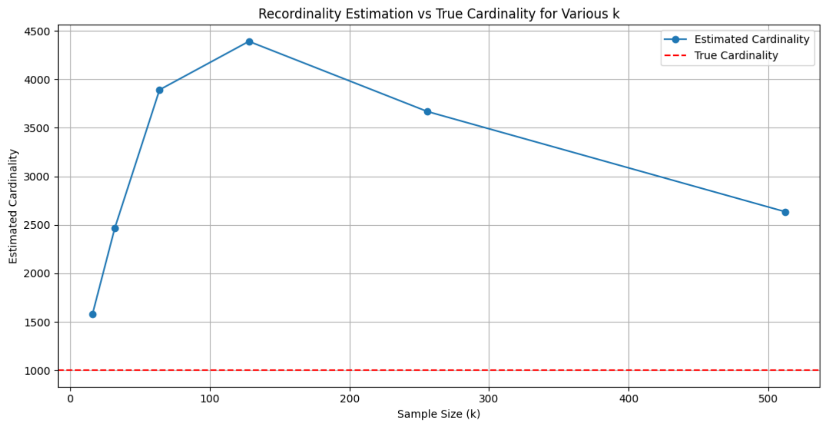


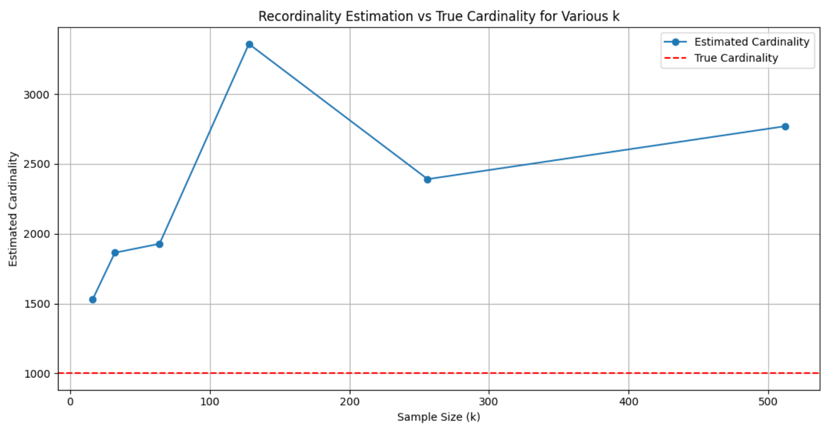


3.3.2 Experiments with k

In cardinality estimation algorithms, the parameter k represents the sample size used in the estimation process. Varying k has a significant impact on the accuracy and efficiency of the estimation. As k increases, the accuracy of the estimated cardinality typically improves, as a larger sample provides a better approximation of the underlying data distribution. However, increasing k also increases memory usage and computation time. Therefore, selecting an appropriate value for k is crucial for balancing between estimation accuracy and computational resources. Below are some experiments with different values of k and the true cardinality.







4 Real World-Datasets

In the following subsection, we will explore the behavior of the HyperLogLog and Recordinality on a few datasets that have been preprocessed.

4.1 HyperLogLog

Here is a table with the results of the estimation, the true cardinality and the difference. I included even the computation time for a better understanding of the whole process.

Book	True Cardinality	Estimated Cardinality	Absolute Difference
crusoe.txt	6245	6277.745300249111	32.7453002491111
dracula.txt	9425	9464.142082880519	39.1420828805185
iliad.txt	8925	8970.943898592186	45.94389859218609
mare-balena.txt	5670	5572.871257042959	97.12874295704114
midsummer-nights-dream	3136	3137.0398799605596	1.039879960559574
quijote.txt	23034	23196.09827543521	162.0982754352117
valley-fear.txt	5830	5824.91598211731	5.084017882690205
war-peace.txt	17476	17741.705157091525	265.7051570915246

Book	Total Words	True Cardinality	Estimated Cardinality	Absolute Difference	Computation Time (s)
crusoe.txt	91813	6245	6277.745300249111	32.7453002491111	0.1193084716796875
dracula.txt	124249	9425	9464.142082880519	39.1420828805185	0.1620798110961914
iliad.txt	124944	8925	8970.943898592186	45.94389859218609	0.16013312339782715
mare-balena.txt	18474	5670	5572.871257042959	97.12874295704114	0.023644208908081055
midsummer-nights-dream	13609	3136	3137.0398799605596	1.039879960559574	0.01851511001586914
quijote.txt	264853	23034	23196.09827543521	162.0982754352117	0.34557557106018066
valley-fear.txt	47064	5830	5824.91598211731	5.084017882690205	0.06769084930419922
war-peace.txt	458701	17476	17741.705157091525	265.7051570915246	0.5958025455474854

- **Overhead of HyperLogLog:** The HyperLogLog algorithm, while efficient for large-scale data streams and approximate counting, incurs some overhead due to its use of internal hash functions and multiple registers for approximation.However, in large datasets where the memory usage is limited and the accuracy of the estimation does not have to be perfect, the HyperLogLog is a very good choice, with high scalability.
- **Time Complexity:** The time complexity of the direct word count operation is linear, $O(N)$, where N is the number of words. In contrast, HyperLogLog operates with constant time complexity per element for adding words to the internal data structure, but the overall runtime is affected by factors such as the error rate and precision parameters.
- **Trade-off Between Time and Accuracy:** While HyperLogLog offers significant time and memory savings for very large datasets by providing approximate results, its estimation is generally slower due to the constant overhead. However, for smaller datasets, the direct word count method is faster and provides an exact result. This highlights the trade-off between the speed of HyperLogLog and the accuracy of direct counting methods.
- **Error Rate and Performance:** The error rate parameter in the HyperLogLog algorithm can be tuned to balance between accuracy and performance. A higher error rate can speed up the algorithm, but this might introduce some loss in accuracy, which needs to be considered depending on the use case.

4.2 Recordinality

Book	True Cardinality	Estimated Cardinality	Computation Time (s)
crusoe.txt	6245	27931.710770257236	0.09768033027648926
dracula.txt	9425	55193.9451941717	0.12291741371154785
iliad.txt	8925	53368.13839709339	0.12406158447265625
mare-balena.txt	5670	24298.173149757327	0.01801919937133789
midsummer-nights-dream	3136	10210.36175905131	0.013078689575195312
quijote.txt	23034	371803.0610196933	0.2601137161254883
valley-fear.txt	5830	22886.57139215075	0.04707694053649902
war-peace.txt	17476	193314.74362744993	0.46083569526672363

Book	True Cardinality	Estimated Cardinality	Computation Time (s)
crusoe.txt	6245	75435.16800259941	0.09284687042236328
dracula.txt	9425	120297.04552442831	0.12185359001159668
iliad.txt	8925	93914.07587357104	0.12463688850402832
mare-balena.txt	5670	18158.00937237335	0.019034862518310547
midsummer-nights-dream	3136	6565.725988775934	0.013521432876586914
quijote.txt	23034	76311.41305282679	0.2639899253845215
valley-fear.txt	5830	149573.8011743766	0.04604744911193848
war-peace.txt	17476	170595.19604291598	0.4459640979766845

5 Comparison HyperLogLog and Recordinality

	Book	True Cardinality	HLL Estimate	Recordinality Estimate
0	crusoe.txt	6245	6277.745300	6839.113857
1	dracula.txt	9425	9464.142083	9622.607409
2	iliad.txt	8925	8970.943899	8748.431567
3	mare-balena.txt	5670	5572.871257	5451.598871
4	midsummer-nights-dream.txt	3136	3137.039880	3406.347829
5	quijote.txt	23034	23196.098275	23665.999243
6	valley-fear.txt	5830	5824.915982	6579.514805
7	war-peace.txt	17476	17741.705157	19126.841150

	Book	True Cardinality	HLL Estimate	Recordinality Estimate
0	crusoe.txt	6245	6277.745300	6683.341845
1	dracula.txt	9425	9464.142083	10262.246273
2	iliad.txt	8925	8970.943899	9170.276546
3	mare-balena.txt	5670	5572.871257	5999.200288
4	midsummer-nights-dream.txt	3136	3137.039880	2775.290301
5	quijote.txt	23034	23196.098275	21448.927683
6	valley-fear.txt	5830	5824.915982	6814.432205
7	war-peace.txt	17476	17741.705157	18374.381403

	Book	True Cardinality	HLL Estimate	HLL Time	Recordinality Estimate	Recordinality Time
0	crusoe.txt	6245	6277.745300	0.009048	6478.494835	0.006318
1	dracula.txt	9425	9464.142083	0.012985	9589.458874	0.010176
2	iliad.txt	8925	8970.943899	0.011718	9884.163217	0.009983
3	mare-balena.txt	5670	5572.871257	0.008025	6548.830427	0.006110
4	midsummer-nights-dream.txt	3136	3137.039880	0.003586	3381.299679	0.004126
5	quijote.txt	23034	23196.098275	0.032444	25078.249876	0.026807
6	valley-fear.txt	5830	5824.915982	0.007956	6223.418320	0.007302
7	war-peace.txt	17476	17741.705157	0.023751	17649.576820	0.019148

Feature	HyperLogLog (HLL)	Recordinality
Accuracy	High (with adjustable error rate)	Adjustable based on k
Time Efficiency	Moderate (due to multiple registers)	Generally faster due to simpler structure
Space Usage	Space-efficient (adjustable precision)	Space-efficient (depends on k)
Implementation Complexity	High (due to hash functions and registers)	Moderate (simpler structure)
Error Rate Control	Precise error control via precision (p)	Adjustable based on k, no strict error control
Use Case	Estimation of cardinality in large datasets	Estimation of cardinality with a tunable trade-off between accuracy and space

Table 1: Comparison of HyperLogLog and Recordinality Algorithms