# Assignment № 3

Vasily Apasov

## Introduction

### Program code

The Python code used for assignment presented in this report can be found in the following GitHub repository:

`https://github.com/codemdvd/ra_assignment_3`

Estimating the number of distinct elements (the cardinality) in a large data stream is a fundamental problem in data analysis, with applications in fields such as networking, databases, and web analytics. Traditional methods of counting unique items require memory proportional to the number of distinct elements, which can quickly become infeasible as data volumes grow. This has led to the development of approximate cardinality estimation algorithms that provide accurate estimates while using significantly less memory.

In this report, we explore and experimentally evaluate the performance of two well-known cardinality estimation algorithms: HyperLogLog (HLL) [1] and Recordinality (REC) [2], along with additional methods such as Probabilistic Counting (PCSA) [3] and Adaptive Sampling [4]. We also demonstrate how these algorithms behave on both real-world textual datasets derived from Project Gutenberg eBooks, and on synthetic data streams generated according to a Zipfian distribution. The real datasets have been pre-processed to ensure a consistent format, while the synthetic streams allow full control over parameters such as the number of distinct elements and the length of the stream.

In each experiment, we compare the estimated cardinalities against the ground truth, which can be directly computed for the given data. We study how various parameters—such as the amount of allocated memory (or number of registers for HLL, and the sample size for REC and Adaptive Sampling)—affect the accuracy of the estimations.

## Theory

This section provides a theoretical overview of the primary cardinality estimation algorithms considered in this study, as well as an outline of the methodology used to generate synthetic data streams. We focus on two main algorithms: HyperLogLog (HLL) and Recordinality (REC). Additionally, we mention other algorithms that were also implemented for comparative purposes, such as Probabilistic Counting (PCSA) and Adaptive Sampling.

## Algorithms

### HyperLogLog (HLL)

HyperLogLog [1] is a probabilistic cardinality estimation algorithm that provides a near-optimal trade-off between accuracy and memory usage. Given a precision parameter $p$, the algorithm uses $m = 2^p$ registers. Each incoming item from the data stream is hashed into a 64-bit integer. The first $p$ bits of this hash determine the register index to update, while the remaining bits are used to determine the position of the first set bit (run-length of leading zeros). Each register stores the maximum observed run-length for all items mapped to it.

After processing the entire stream, HLL combines the information from all registers to estimate the cardinality. The expected relative error of HLL is about $1.04/\sqrt{m}$, and by adjusting $p$, one can trade memory for accuracy. HLL has become a widely adopted standard due to its scalability and good accuracy guarantees.

### Recordinality (REC)

Recordinality (REC) [2] is often implemented using the principle of $k$-minimum values (KMV). In this approach:

1. Hash each unique item to a real number in $[0, 1)$. 2. Keep track of only the $k$ smallest hash values. 3. If fewer than $k$ unique items are seen, the true cardinality is simply the number of unique items. Otherwise, if the largest of the $k$ retained hash values is $X_{(k)}$, the cardinality estimate is given by $(k-1)/X_{(k)}$.

By adjusting $k$, one can tune the trade-off between memory consumption and estimation variance. Larger values of $k$ typically yield more accurate estimates but require more memory.

### Bonus: Probabilistic Counting (PCSA)

Probabilistic Counting with Stochastic Averaging (PCSA) [3] was one of the earliest methods proposed for approximate cardinality estimation. PCSA uses bitmaps to record the positions of least significant set bits of hashed items. Although foundational, PCSA is generally less accurate compared to HLL and REC for the same amount of memory.

### Bonus: Adaptive Sampling

Adaptive Sampling [4] dynamically maintains a sample of approximately $k$ smallest hash values. Initially, all items with hash values below a threshold $T$ are included. If more than $k$ items qualify, $T$ is raised to drop the largest values, thus maintaining about $k$ samples. After processing the entire stream, if the largest sampled hash is $T$, the cardinality estimate is $(k-1)/T$. This method adapts to the data distribution and can offer decent accuracy with relatively low memory usage.

## Data Generation

In addition to real-world datasets, we also generate synthetic data streams to allow controlled experiments. We consider a Zipfian distribution with parameter $\alpha \geq 0$ over $n$ distinct elements $\{x_1, \dots, x_n\}$. The probability of selecting element $x_i$ for the $j$-th position in the stream $(z_j)$ is given by:

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

where

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

For $\alpha = 0$, this reduces to a uniform distribution. As $\alpha$ increases, the distribution becomes more skewed, with a small number of elements dominating the stream. By varying $n$, $N$, and $\alpha$, we can thoroughly test how each algorithm responds to different distributions and scales.

The theory and principles described in this section form the foundation for the experiments and evaluations presented in the subsequent sections of this report.

### Hash Function Choice: xxhash

In all the algorithms it is crucial to use a hash function that produces a uniform and unbiased distribution of hash values over the space of possible items. This ensures that the probabilistic sampling or partitioning steps of the algorithms are not influenced by any non-random patterns in the data.

In this work, we use `xxhash`, a non-cryptographic hash function known for its high speed and good distributional properties. Unlike traditional cryptographic hash functions (e.g., SHA-1 or SHA-256), `xxhash` is optimized for performance and streaming scenarios. Its main advantages are:

1. Speed: `xxhash` is engineered for efficiency, making it well-suited for high-throughput data streams where computation time must be minimized.

2. Uniformity: While not cryptographically secure, `xxhash` provides a uniform distribution of hash values. This uniformity is essential for algorithms like HLL or REC, which rely on the assumption that hash values are approximately i.i.d. uniform random variables.

3. Randomization: Although `xxhash` is deterministic given a fixed seed, we introduce randomness by choosing different seeds for each run. This approach allows us to obtain multiple, independent estimates from the same dataset and average them, providing insights into the variance and stability of the estimator under changing hash functions.

By ensuring efficient and well-distributed hashing, `xxhash` helps the considered algorithms maintain their theoretical guarantees and keeps overheads minimal, leading to more reliable cardinality estimates with reduced computation time.

## Experiments

In this section, we first describe the parameter settings for the considered algorithms, then present the experimental results. To improve clarity, we provide separate tables for each algorithm across all datasets. This layout allows a direct comparison of how each algorithm performs on different data sources without mixing multiple estimators in the same table.

### Parameter Settings

- **HyperLogLog (HLL):** We choose a precision parameter $p(14)$ that determines $m = 2^p$ registers. Standard error is approximately $1.04/\sqrt{m}$.
- **Recordinality (REC):** We fix $k(1024)$ for the $k$-minimum values method. Adjusting $k$ balances memory and accuracy.
- **Probabilistic Counting (PCSA):** We select a number of registers $m(3000)$. PCSA is included for historical and comparative purposes.
- **Adaptive Sampling:** Similar to REC, we use a sample size $k(1024)$ and adapt the threshold to maintain about $k$ samples.

All experiments are performed over 5 runs, and results are averaged. Relative Error (RE) is computed as $|C_{\text{est}} - C_{\text{true}}|/C_{\text{true}}$.

### Experiment 1: Real Datasets Comparison

For each dataset, we create a table listing its True Cardinality, the Estimated Cardinality (average over 5 runs), and the Relative Error for each algorithm.

**crusoe (True Card. = 6245)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 6230.89 | 0.0023 |
| REC | 6346.14 | 0.0162 |
| PCSA | 5249.34 | 0.1594 |
| Adaptive Sampling | 6225.98 | 0.0030 |

Table 1: Results on crusoe dataset.

**dracula (True Card. = 9425)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 9417.59 | 0.0008 |
| REC | 9328.84 | 0.0102 |
| PCSA | 5411.91 | 0.4258 |
| Adaptive Sampling | 9361.21 | 0.0068 |

Table 2: Results on dracula dataset.

**iliad (True Card. = 8925)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 8933.68 | 0.0010 |
| REC | 8941.65 | 0.0019 |
| PCSA | 5398.17 | 0.3952 |
| Adaptive Sampling | 8995.25 | 0.0079 |

Table 3: Results on iliad dataset.

**mare-balena (True Card. = 5670)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 5649.50 | 0.0036 |
| REC | 5723.48 | 0.0094 |
| PCSA | 5214.29 | 0.0804 |
| Adaptive Sampling | 5575.71 | 0.0166 |

Table 4: Results on mare-balena dataset.

**midsummer-nights-dream (True Card. = 3136)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 3147.48 | 0.0037 |
| REC | 3168.03 | 0.0102 |
| PCSA | 4869.60 | 0.5528 |
| Adaptive Sampling | 3172.88 | 0.0118 |

Table 5: Results on midsummer-nights-dream dataset.

**quijote (True Card. = 23034)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 23026.57 | 0.0003 |
| REC | 23248.69 | 0.0093 |
| PCSA | 5483.65 | 0.7619 |
| Adaptive Sampling | 23528.09 | 0.0215 |

Table 6: Results on quijote dataset.

**valley-fear (True Card. = 5829)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 5839.77 | 0.0018 |
| REC | 5901.73 | 0.0125 |
| PCSA | 5197.45 | 0.1083 |
| Adaptive Sampling | 5725.19 | 0.0178 |

Table 7: Results on valley-fear dataset.

**war-peace (True Card. = 17475)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 17388.14 | 0.0050 |
| REC | 17925.14 | 0.0258 |
| PCSA | 5479.85 | 0.6864 |
| Adaptive Sampling | 17992.86 | 0.0296 |

Table 8: Results on war-peace dataset.

**Analysis**

Overall, the results demonstrate that the HyperLogLog (HLL) algorithm achieves notably low relative errors (RE) across all datasets. For instance, the *war-peace* dataset shows an RE of 0.0050 with HLL, indicating a high level of accuracy. Similarly, Adaptive Sampling also maintains relatively low errors, with most datasets exhibiting RE values under 3% and the *war-peace* dataset at about 2.96%.

Recordinality (REC) provides moderate accuracy. In most cases, it performs better than PCSA, though it generally shows higher RE than HLL and Adaptive Sampling. For example, the *quijote* dataset yields an RE of 0.0093 with REC, which is almost three times larger than HLL's 0.0003% for the same dataset.

In contrast, the Probabilistic Counting with Stochastic Averaging (PCSA) method exhibits significantly larger errors. Several datasets (e.g., *quijote* and *war-peace*) show RE values exceeding 68%. Even the more moderate cases often present RE values well above 10%.

In summary, HLL and Adaptive Sampling emerge as the most accurate techniques, consistently providing low RE values. REC stands in a middle ground, performing acceptably though not as accurately. PCSA, on the other hand, shows substantial deviations from the true cardinalities, making it less suitable for scenarios where precision is a key requirement. The specified parameter m with a value of 3000 in the PCSA algorithm was not effective in all cases. It is worth carefully choosing the values of this parameter for each task.

## Experiment 2: Synthetic Datasets Comparison

We now consider three synthetic datasets generated with Zipfian distributions.

For the three synthetic datasets, the parameters are as follows:

- $dataset_1$: $n = 9999$, $N = 1000$, $\alpha = 0.7$
- $dataset_2$: $n = 99999$, $N = 6666$, $\alpha = 0.8$
- $dataset_3$: $n = 999999$, $N = 45323$, $\alpha = 0.9$

**Results on $dataset_1$ (True Card. = 789)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 788.47 | 0.0007 |
| REC | 789.00 | 0.0000 |
| PCSA | 4198.30 | 4.3210 |
| Adaptive Sampling | 789.00 | 0.0000 |

Table 9: Results on $dataset_1$.

**Results on $dataset_2$ (True Card. = 4829)**

| **Algorithm** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| HLL | 4826.85 | 0.0004 |
| REC | 4902.83 | 0.0153 |
| PCSA | 5083.44 | 0.0527 |
| Adaptive Sampling | 4734.76 | 0.0195 |

Table 10: Results on $dataset_2$.

**Results on $dataset_3$ (True Card. = 26975)**

| Algorithm | Est. Card. (avg) | RE |
|---|---:|---:|
| HLL | 26985.78 | 0.0004 |
| REC | 27218.02 | 0.0090 |
| PCSA | 5484.92 | 0.7967 |
| Adaptive Sampling | 27216.17 | 0.0089 |

Table 11: Results on $dataset_3$.

### Analysis of Synthetic Results

Overall, HLL and Adaptive Sampling deliver very accurate estimates. For example, both methods achieve nearly perfect estimations on $dataset_1$ (RE close to 0.0000 for Adaptive Sampling and 0.0007 for HLL), and maintain errors under 1% even for the larger datasets. REC also performs well on $dataset_1$ with zero error, but exhibits moderate increases in RE for $dataset_2$ and $dataset_3$ (up to about 1.53% and 0.90%, respectively).

In contrast, PCSA shows substantial deviations, especially for $dataset_1$ and $dataset_3$. For $dataset_1$, the error exceeds 400%, and for $dataset_3$, it approaches 80%. Even though $dataset_2$ errors for PCSA are somewhat lower (around 5%), these results are still less reliable compared to the other techniques.

In summary, both HLL and Adaptive Sampling display consistently low relative errors across all three synthetic datasets, with REC performing moderately well, and PCSA showing significantly larger estimation errors.

## Experiment 3: Memory/Parameter Impact Study

We now vary the parameters $m$ in HLL and $k$ in REC to study their impact on estimation accuracy. For each algorithm, we provide a table listing results for different parameter settings.

### HLL (Memory Variation) on dracula dataset

True Card. = 9425

| Param ($m$) | Est. Card. (avg) | RE |
|---|---:|---:|
| 16 | 688.98 | 0.9269 |
| 1024 | 2601.99 | 0.7239 |
| 8192 | 9371.67 | 0.0057 |
| 262144 | 9427.71 | 0.0003 |

Table 12: HLL and dracula.

**HLL (Memory Variation) on war-peace dataset**

True Card. = 17475

| **Param ($m$)** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| 16 | 1170.85 | 0.9330 |
| 1024 | 1484.94 | 0.9150 |
| 8192 | 17469.07 | 0.0003 |
| 262144 | 17478.99 | 0.0002 |

Table 13: HLL and war-peace.

**REC (Parameter Variation) on dracula dataset**

True Card. = 9425

| **Param ($k$)** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| 100 | 11048.07 | 0.1722 |
| 500 | 9698.34 | 0.0085 |
| 1500 | 9516.66 | 0.0097 |
| 4000 | 9499.30 | 0.0079 |

Table 14: REC and dracula.

**REC (Parameter Variation) on war-peace dataset**

True Card. = 17475

| **Param ($k$)** | **Est. Card. (avg)** | **RE** |
|---|---:|---:|
| 100 | 15703.77 | 0.1014 |
| 500 | 17134.36 | 0.0195 |
| 1500 | 17192.71 | 0.0162 |
| 4000 | 17454.22 | 0.0012 |

Table 15: REC and war-peace.

**Analysis of Parameter Variation**

The results show a clear relationship between the chosen parameters (which control memory or sampling size) and the accuracy of both HLL and REC estimates.

For HLL, varying the memory parameter $m$ significantly affects the accuracy of the estimation. At very low memory settings, as seen with $m = 16$ on both the *dracula* and *war-peace* datasets, the relative error (RE) is extremely high (over 90%). As $m$ increases, the estimates approach the true cardinality. For example, on the *dracula* dataset, increasing $m$ to 8192 reduces the RE to 0.57%, and further increasing it to 262144 reduces the error to just 0.03%. A similar pattern is observed for the *war-peace* dataset, where increasing $m$ results in dramatic improvements, finally achieving near-perfect estimates with RE values as low as 0.02%.

For REC, adjusting the parameter $k$ affects the accuracy in a similar way. Initially, small values of $k$ lead to higher relative errors. For the *dracula* dataset, $k = 100$ yields a 17.22% error, which steadily decreases as $k$ increases, eventually dropping below 1% at $k = 4000$. The *war-peace* dataset follows the same trend: starting from approximately 10% error at $k = 100$ and improving to around 0.12% at $k = 4000$.

In summary, both HLL and REC exhibit considerable sensitivity to their respective memory and parameter choices. Increasing the available memory or the parameter size (i.e., $m$ for HLL and $k$ for REC) leads to substantially more accurate cardinality estimates, demonstrating that these algorithms can be tuned to balance resource usage and estimation precision.

### Conclusion

In conclusion, our comprehensive evaluation of various cardinality estimation algorithms (HLL, REC, PCSA, and Adaptive Sampling) across both real and synthetic datasets demonstrates that HLL and Adaptive Sampling consistently deliver low relative errors and reliable estimates. REC shows moderate but acceptable performance, while PCSA's accuracy is generally lower. Moreover, our parameter variation experiments for HLL and REC confirm that increasing memory or parameter size significantly improves estimation accuracy. These findings highlight the importance of both algorithm selection and parameter tuning in achieving precise and resource-efficient cardinality estimation.

## References

[1] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics and Theoretical Computer Science*, Proceedings of AofA 2007.

[2] Sasu Tarkoma, Boris Mutnny, and Yu Xiao. Recordinality: Efficiently counting distinct data using a hierarchy of registers. Technical Report TKK-ICS-TR-8, Department of Communications and Networking, Aalto University, 2018.

[3] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[4] Philippe Flajolet and G. Nigel Martin. Adaptive sampling and the approximation of zero frequencies. Research Report 0193, INRIA Rocquencourt, 1985.