Exact, Approximate and Data Stream Counting Methods for Identifying Frequent Letters in Literary Works

Miguel Matos, 103341

***Abstract* – Design, testing, and analysis of three computational algorithms to identify frequent letters in literary works, namely an exact counter, an approximate counter with decreasing counting probability, and space-saving count.**

I. Introduction

In the rapidly evolving field of data analysis, the ability to efficiently process and analyze large volumes of text data is increasingly vital. This report presents the findings of a project aimed at identifying the most frequent letters in text files, specifically sourced from literary works. The project is a part of the coursework for "Algoritmos Avançados" at the Universidade de Aveiro for the academic year 2023/2024.

The primary objective of this project is to explore and evaluate different methodologies for determining the frequency of letters in text data. This task not only serves as a practical application of text data analysis but also as a means to understand the nuances of various counting and estimation techniques. The specific goals are as follows:

- To compute the exact number of occurrences of each letter in the text files.
- To estimate the most frequent letters using a data stream algorithm for varying numbers of top frequent letters (n = 3, n = 5, and n = 10).
- To perform comparative analysis between the exact counts and the estimates provided by approximate counters and data stream algorithms.

To achieve these objectives, three distinct approaches were employed:

- **Exact Counts**: A method that accurately counts the occurrence of each letter in the text.
- **Approximate Counts**: Technique that provides an estimated count of letter frequencies, balancing accuracy with computational efficiency. The conducted work focuses on a decreasing probability counter (*1/sqrt(2)^k*).
- **Data Stream Algorithm**: Specialized algorithm designed to identify frequent items in data streams, useful for processing large or continuous datasets. The conducted work focuses on the space-saving count.

II. Methodology

A. Data Acquisition and Preparation

The foundation of our analysis was text files sourced from literary works, primarily obtained from Project Gutenberg. This data collection consisted of five versions of “The Odyssey”, by Homero, in five different languages, which were English, Portuguese, Spanish, French, and Swedish.

The following steps were taken to prepare the data:

- Removal of Headers: Each text file from Project Gutenberg begins with a standard header. These headers were programmatically removed to ensure that the analysis focused solely on the literary content.
- Elimination of Non-Alphabetic Characters: All stop-words, punctuation marks, and numerals were removed from the text. This step was crucial to focus exclusively on the frequency of letters.
- Case Normalization: All letters in the text were converted to uppercase to maintain consistency in counting letter frequencies, as case differences are irrelevant in this context.

B. Counting Methods

i. Exact Counts

The exact count algorithm iterates through each character in the text and mantains a precise count of each letter. It makes use of a dictionary, where each key represents a letter, and the corresponding value is the count of that letter's occurrences.

The algorithm can be reduced to the following pseudocode:

```
1. Read the text file.
2. For each letter:
   a. Increment its count in the dictionary.
3. Compile the final count of each letter.
```

ii. Approximate Counts

The approximate count algorithm estimates the frequency of each letter using a probabilistic approach to decide whether to increment the count for a letter. As instructed, the probability of the algorithm incrementing the count is *1/sqrt(2)^k*, which means that the higher the count of a letter, the less probable is the increment of its count.

The algorithm can be reduced to the following pseudocode:

```
1. Read the text file.
2. For each letter:
   a. Try to increment the count of that letter, with success probability of 1/sqrt(2)^k, where k is the current count of the letter.
4. Compile the final count of each letter.
```

iii. Data Stream Counts

The implemented data stream algorithm uses the space-saving count strategy, as instructed. It makes use of a fixed-size min-heap (using Python's *heapq*) to keep track of the most frequent letters. As new letters are processed, the algorithm updates the heap based on the frequency of the letter.

The algorithm can be reduced to the following pseudocode:

```
1. Initialize min-heap with given size.
2. For each letter in the data stream:
   a. If the letter in the heap, increment its count and re-heapify the min-heap.
   b. If size of heap < given size, add letter to min-heap with count of 1.
   c. Else, replace least frequent letter in heap with current letter, keeping and incrementing the discarded letter's count.
3. Compile the final count of each letter.
```

C. Computational Experiments

To evaluate and compare these methods, a series of computational experiments was conducted.

Each text file was processed using the exact counter to obtain a definitive frequency count of each letter. The same text files were then processed using both the approximate counter and the data stream algorithm. This was repeated multiple times to assess consistency and reliability. For the data stream algorithm, experiments were conducted for different sizes of the min-heap (3, 5 and 10) to observe the variation in the results based on the number of top frequent letters analysed.

III. RESULTS

The gathered results can be summed up to the following:

- Comparison between Exact and Approximate letter frequencies
- Comparison between Exact and Data Stream letter frequencies
- Comparison between letter frequencies regarding the same work in different languages

The following plots and tables serve to visualize these results.

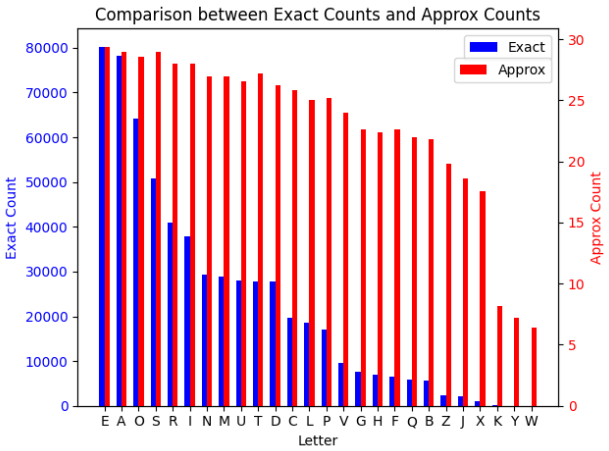

Figure 1 - Portuguese Exact-Approx Comparison

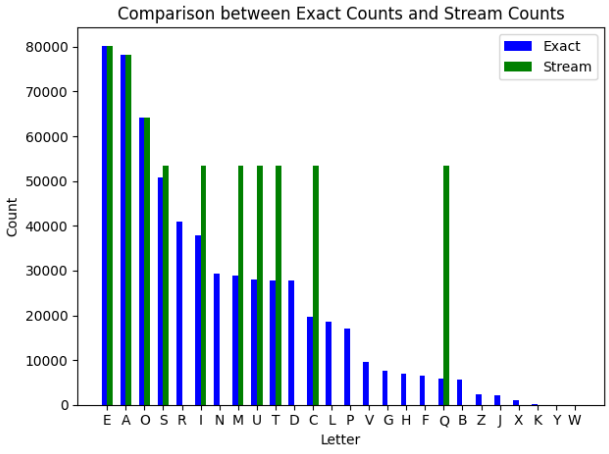

Figure 2 - Portuguese Exact-Stream Comparison

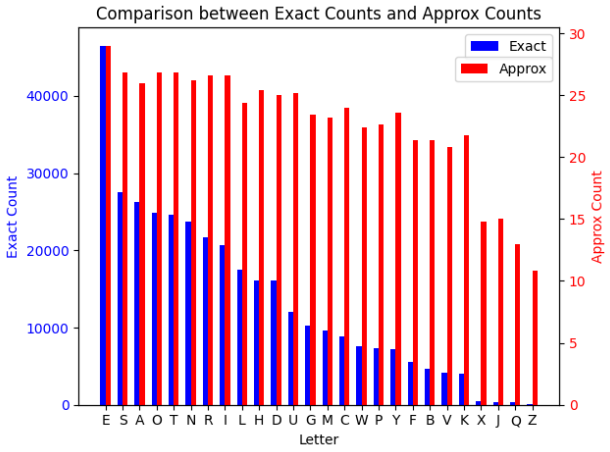

Figure 3 - English Exact-Approx Comparison

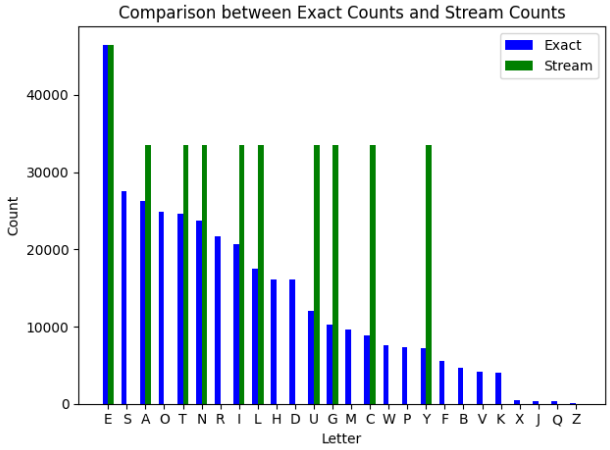

Figure 4 - English Exact-Stream Comparison

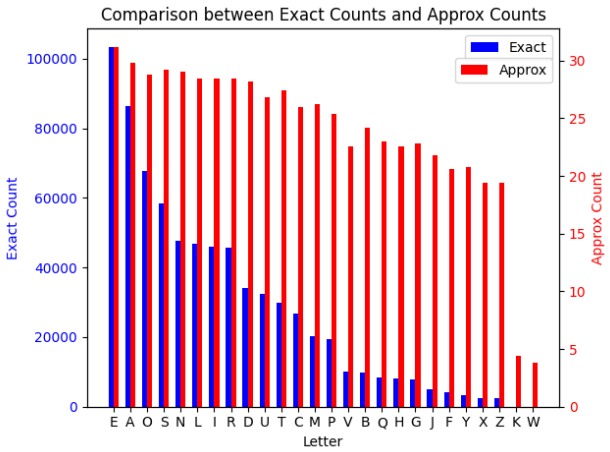

Figure 5 - Spanish Exact-Approx Comparison

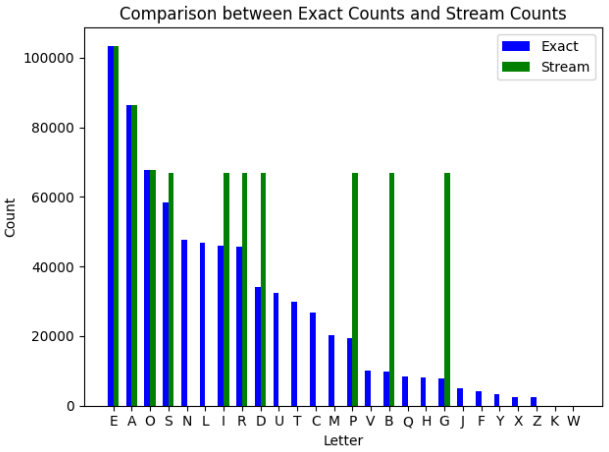

Figure 6 - Spanish Exact-Stream Comparison

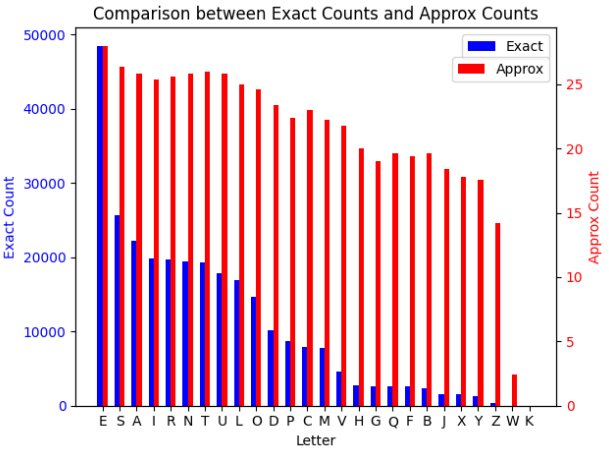

Figure 7 - French Exact-Approx Comparison

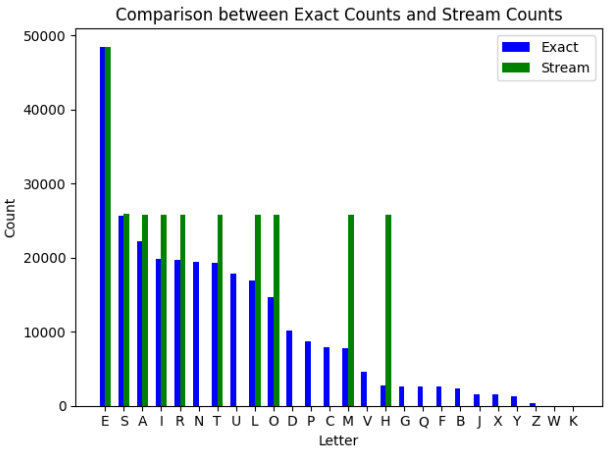

Figure 8 - French Exact-Stream Comparison

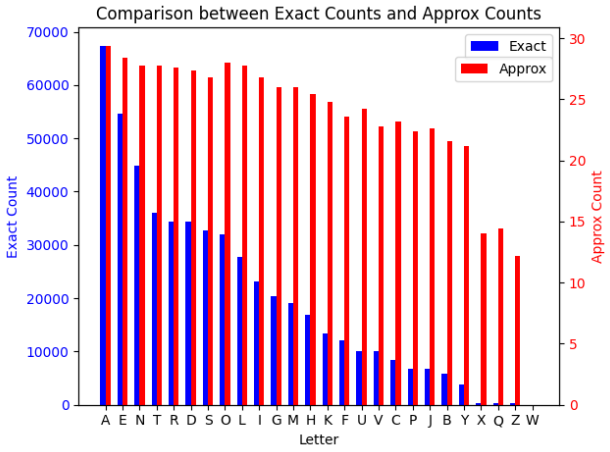

Figure 9 - Swedish Exact-Approximate Comparison

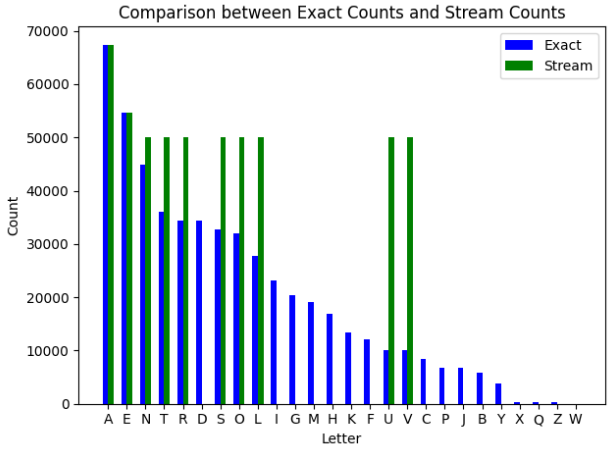

Figure 10 - Swedish Exact-Stream Comparison

Table 1 - Top 10 frequent letters for the Exact Counter

PT		EN		ES		FR		SE	
E	80274	E	46514	E	103445	E	48490	A	67408
A	78148	S	27593	A	86267	S	25616	E	54657
O	64103	A	26265	O	67621	A	22193	N	44932
S	50710	O	24846	S	58374	I	19799	T	36115
R	41022	T	24555	N	47680	R	19759	R	34465
I	37983	N	23721	L	46770	N	19470	D	34361
N	29319	R	21717	I	45939	T	19352	S	32677
M	28775	I	20737	R	45674	U	17830	O	31905
U	27926	L	17443	D	34186	L	16870	L	27741
T	27826	H	16069	U	32460	O	14691	I	23110

Table 2 - Top 10 average frequent letters for 5 runs of the Approximate Counter

PT		EN		ES		FR		SE	
E	29.4	E	29	E	30.0	E	28	A	28.8
A	29	S	26.8	A	29.8	S	26.4	E	28.6
S	29	O	26.8	O	29.4	T	26	O	28
O	28.6	T	26.8	S	29	A	25.8	N	27.8
R	28	R	26.6	L	27.8	N	25.8	T	27.8
I	28	I	26.6	R	27.8	U	25.8	L	27.8
T	27.2	N	26.2	T	27.8	R	25.6	R	27.6
N	27	A	26	N	27.6	I	25.4	D	27.4
M	27	H	25.4	I	27.6	L	25	S	26.8
U	26.6	U	25.2	D	26.8	O	24.6	I	26.8

Table 3 - Top 10 frequent letters for the Space Saving Counter

PT		EN		ES		FR		SE	
E	80274	E	46514	E	103445	E	48490	A	67408
A	78148	L	33531	A	86267	S	25864	E	54667
O	64105	N	33530	O	67728	A	25849	L	49982
S	53544	A	33529	I	66896	R	25826	T	49981
I	53523	I	33528	D	66876	L	25807	S	49981
C	53522	U	33528	S	66875	I	25806	N	49980
T	53521	G	33528	R	66875	O	25806	U	49980
M	53521	T	33527	B	66875	T	25805	O	49979
U	53520	O	33527	G	66875	M	25805	R	49979
Q	53520	Y	33527	P	66874	H	25805	V	49979

Table 4: Results for the Space Saving Counter with different values of n (heap size) for the portuguese version

N = 3		N = 5		N = 10	
C	199066	I	119440	E	80274
A	199066	A	119440	A	78148
S	199066	S	119440	O	64105
		M	119439	S	53544
		T	119439	I	53523
				C	53522
				M	53521
				T	53521
				Q	53520
				U	53520

IV. DISCUSSION

A. Comparison of Algorithms

The exact counter displayed unparalleled accuracy in determining the frequency of letters, as they provide the actual count without any estimation. However, for larger datasets, this algorithm is more computationally expensive, especially in processing time and memory usage, since it keeps track of the exact letter counts.

The approximate counter does not match the exact counter in precision, since, by definition, it only keeps track of approximate values for the frequency of each letter. However, they did provide a good estimate of the frequency of each letter, with reduced computational load. This type of counter is more suitable for situations where slight inaccuracies in the results are tolerable, in exchange for faster processing and lower memory usage.

The data stream algorithm/space saving algorithm is tailored for scenarios involving large, continuous data streams. The accuracy of this algorithm for identifying the most frequent letters was impressive, particularly for larger values of the heap size. However, it could only count with precision at most the top 2 frequent letters, for a heap size of 10. As with the approximate counters, it also becomes victim of the speed-accuracy trade-off, since reducing the heap size to obtain faster times would result in less accurate results, as seen in Table 4.

With this, we can conclude that each algorithm has distinct advantages and is suited for specific types of tasks. Exact counters are the gold standard for accuracy but are the heaviest in terms of resources; approximate counters provide a practical middle ground; data stream algorithms excel in scenarios requiring processing of large datasets.

B. Algorithms Accuracy

The results of the exact counter were used as the ground truth for the analysis of the accuracy of the other two algorithms.

The approximate counters, by design, trade some accuracy for performance. The largest absolute error obtained was of 5 positions, and the lowest was of 0 positions. The average absolute error was of 2.4 positions.

The exact counters did not perform well for the lower values of the heap size. However, for a heap with 10 maximum letters, it almost always provided the exact results for the top 2 letters, while the rest of the letters did not show much accuracy.

C. Consistency Across Different Languages

This study also served to assess whether the most frequent letters remain consistent across the same literary works translated into different languages.

The study was conducted across five different translations of the same work, and revealed that letters such as ‘E’ and ‘A’ were the most frequent. This is probably due the fact the most of the chosen translations shared the same linguistic roots and structures, such as Portuguese, Spanish, French and English.

V. CONCLUSION

This project compared exact counters, approximate counters, and data stream algorithms for identifying the most frequent letters in literary texts. Exact counters were highly accurate but less efficient with large datasets, highlighting a trade-off between accuracy and computational resources. Approximate counters offered improved efficiency but with variable precision. The data stream algorithm struck a balance between speed and accuracy, adapting well to different data sizes and real-time analysis requirements.

The comparative analysis revealed insights into letter frequency across languages, demonstrating the algorithms' robustness across diverse datasets. Overall, this study emphasizes the importance of selecting an appropriate method based on specific needs for accuracy, efficiency, and data scalability.

REFERENCES

[1]https://chat.openai.com/
[2]https://algo.inria.fr/flajolet/Publications/Flajolet85c.pdf
[3]https://romania.amazon.com/techon/presentations/DataStreamsAlgorithms_FlorinManolache.pdf