Most Common Letters

Ricardo Rodriguez (98388)
Universidade de Aveiro - DETI
AA - Algoritmos Avançados
ricardorodriguez@ua.pt

***Abstract*—This document describes the work done on the third and last project of the *Algoritmos Avançados* course, whose objective is to identify the most frequent letters in text files using three different approaches: exact count, approximated count with decreasing probability count and, finally, the space-saving algorithm for data streams.**

***Index Terms*—count, probability, space-saving**

I. Introduction

The ability to efficiently count large numbers of events or data streams is of critical importance for many different areas. Whether it be tracking the number of page views on a website, monitoring network traffic, analyzing social media data or analyzing large datasets of DNA sequences, the ability to quickly and accurately count large volumes of data can have significant real-world applications.

In this paper, we will explore the decreasing probability count and space-saving algorithms that have been developed to tackle this problem, as well as comparing them to the exact count of letters in a text document in terms of results and computational efficiency and limitations.

II. Problem

Exact count refers to the process of accurately counting the number of occurrences of an event or the number of elements in data. Although this is the best methodology, the problem of exact count can be challenging due to the large volumes of data that are often involved. For example, if we have a text file containing billions of characters and we want to determine the exact number of characters in the document, it can be computationally expensive to iterate through the entire file to count them. In addition, the traditional exact count method on large data streams is not well-suited to the continuous and diversified input of data.

To address these challenges, we will use the decreasing probability count and space-saving algorithms algorithms. While these approaches may not provide an exact count, they can provide a good approximation of the total count with a high level of efficiency without the space and time complexity of the exact count algorithm.

III. Data

The data used to simulate data streams for the computational experiences were obtained from the Project Gutenberg [1]. This library contains thousands of free literary books in different languages. For this work, we only need the text file of the book to find the most frequent letters of it.

All text files are processed by removing the Project Gutenberg file headers, the stop-words and punctuation marks according to the language of the book and by converting all letters to uppercase.

The text files used for computational testing were "Romeo and Juliet", in four different languages: english, french, german and dutch.

IV. Count Algorithms

A. Exact counter

The exact count algorithm computes the exact number of occurrences of each letter using the Counter Python library [2]. It iterates through all the text characters and constructs the dictionary with the exact count as values.

Algorithm 1 Exact counter - Pseudo-code

```
c ←
for each item i do
    i ← uppercase(i)
    if i not in c then
        c[i] ← 1
    else
        c[i] ← c[i] + 1
    end if
end for
```

Since there are n characters in a text file, the amount of bits needed to represent the total number of character counts is $\log_2(n)$.

B. Approximate counter with $1/2^k$ decreasing probability

An approximate counter is a data structure that keeps track of the number of times an event has occurred. It's often used when the cost of maintaining an exact count is too high, which is common for large data streams.

To implement the approximate counter, it was proposed to use a decreasing probability factor of $1/2^k$ for each character. This means that if the counter of the current character is 2, k is 2 and the counter will only be incremented with a probability of $1/2^2$, which turns out to be 0.25. Thus, the probability of incrementing a character's count will always be half of its previous probability value.

Algorithm 2 Approximate counter - Pseudo-code

```
c ←[]
for each item i do
  if i not in c then
    c[i] ← 0
  end if
  k ← c[i]
  p ← random.uniform(0, 1)
  if p < 1/2^k then
    c[i] ← c[i] + 1
  end if
end for
```

C. Space-Saving counter

The space-saving count is an algorithm used to find item frequency in data streams that can't store the exact counts of elements in memory, just like the decreasing probability count mentioned in the previous subsection.

The difference between both algorithms is that the decreasing probability count uses the nature of probability to determine whether or not an item gets incremented, while the space-saving count works by keeping a k fixed-size data structure to support the approximate counts of the stored elements.

Initially, the data structure gets filled by the items and their exact counts until it reaches the maximum fixed-size, or k. When the next item being iterated is being tracked by the data structure, it's count is simply incremented. However, when the item is not being monitored, the pair of item and count with the smallest count value is replaced by the new item, from the current iteration, with an incremented count.

Algorithm 3 Space-Saving counter for each k - Pseudo-code

```
c ←[]
for each item i do
  if i ∈ c then
    c[i] ← c[i] + 1
  else
    l ← length(c)
    if l >= k then
      smallest_char ← character with least count in c
      smallest_value ← c[smallest_char]
      c ← c − smallest_char
      c[i] ← smallest_value + 1
    else
      c[i] ← 1
    end if
  end if
end for
```

As observed, if the item is not in the data structure, it is added to the summary and one of the existing items is removed to make room, based on the frequency of the existing items. Thus, approximate frequencies of the items in the summary are updated as new items are processed using a fixed amount of memory.

V. Results

The results of decreasing probability counter and space-saving counter for different languages of the same book are presented in this chapter.

A. English

The following table presents the 10 characters with the highest count in the three different algorithms: exact count, decreasing probability count with a factor of $1/2^k$ and space-saving count.

Exact count		Approximate count		Space-saving count	
E	9429	E	13	E	9429
T	5760	T	13	O	7223
O	5618	A	13	T	7219
A	5509	G	13	N	7219
R	5258	C	13	G	7219
S	5059	L	12	B	7219
I	4347	I	12	R	7219
N	4342	S	12	J	7219
L	4282	H	12	K	7218
H	3474	N	12	M	7218

TABLE I
10 MOST FREQUENT LETTERS IN ENGLISH FOR DIFFERENT COUNT METHODS (EXACT COUNT, APPROXIMATE COUNT, SPACE-SAVING COUNT

The absolute and relative errors for the approximate count, using a decreasing probability factor of $1/2^k$, are presented in the following graphs:

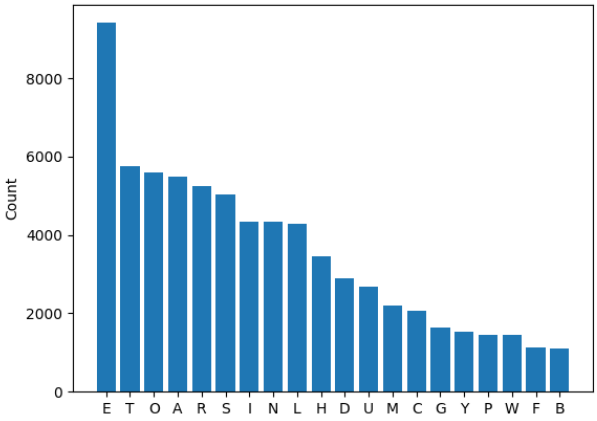

Fig. 1. Absolute errors for the decreasing probability count with a $1/2^k$ factor

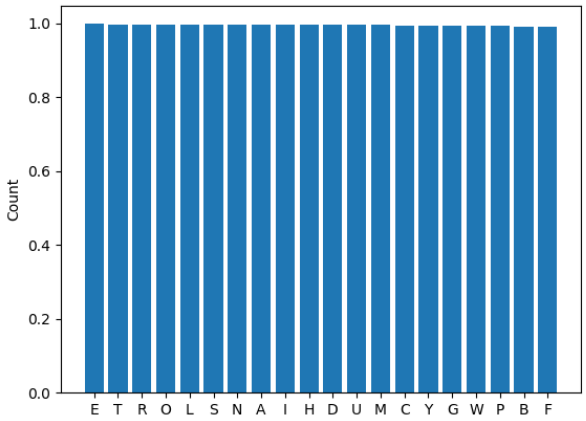

Fig. 2. Relative errors for the decreasing probability count with a $1/2^k$ factor

The performance of the space-saving count for different k values (3,5,10) can be seen in the table blow, regarding absolute and relative errors:

	k = 3	k = 5	k = 10
Minimum Absolute Error	15371	5452	0
Average Absolute Error	19688.66	10318.8	3799.5
Maximum Absolute Error	24512	14591	6930
Minimum Relative Error	1.63	0.57	0
Average Relative Error	29.95	12.05	4.56
Maximum Relative Error	84.81	50.48	23.97

The absolute and relative errors for the space-saving count are presented in the following graphs:

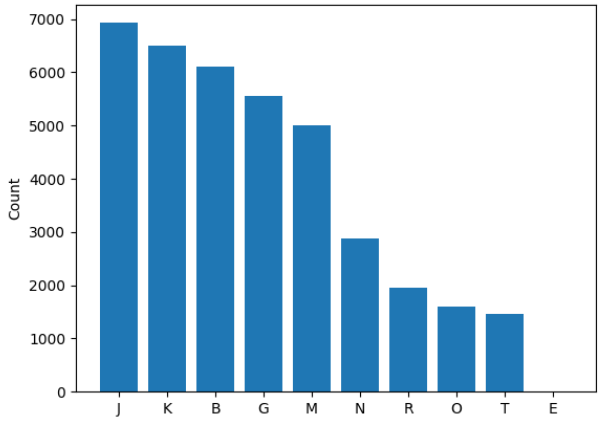

Fig. 3. Absolute errors for the space-saving count

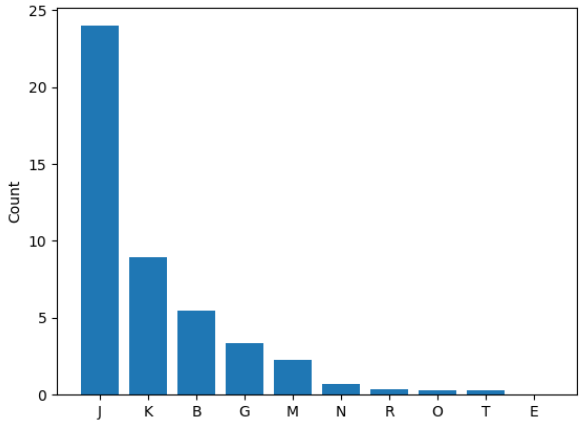

Fig. 4. Relative errors for the space-saving count

B. French

The following table presents the 10 characters with the highest count in the three different algorithms: exact count, decreasing probability count with a factor of $1/2^k$ and space-saving count.

Exact count		Approximate count		Space-saving count	
E	16181	U	14	E	16181
R	9852	I	14	O	11136
I	8942	E	14	R	11134
T	8150	O	14	B	11134
O	8106	L	13	É	11134
N	7931	T	13	J	11134
A	7755	R	13	U	11133
S	7753	A	13	N	11133
U	6410	N	13	K	11133
L	5607	S	13	M	11133

TABLE II
10 MOST FREQUENT LETTERS IN FRENCH FOR DIFFERENT COUNT METHODS (EXACT COUNT, APPROXIMATE COUNT, SPACE-SAVING COUNT

The absolute and relative errors for the approximate count, using a decreasing probability factor of $1/2^k$, are presented in the following graphs:

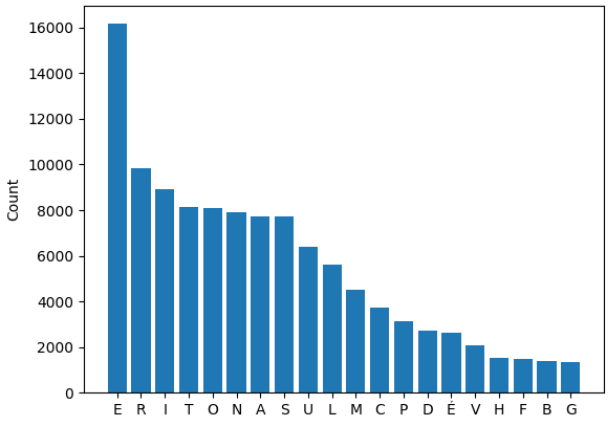

Fig. 5. Absolute errors for the decreasing probability count with a $1/2^k$ factor

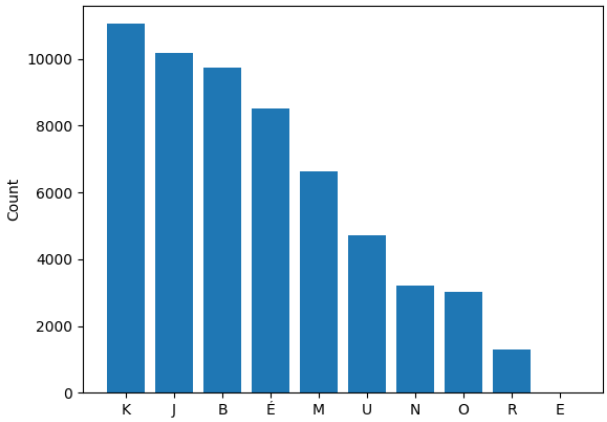

Fig. 7. Absolute errors for the space-saving count

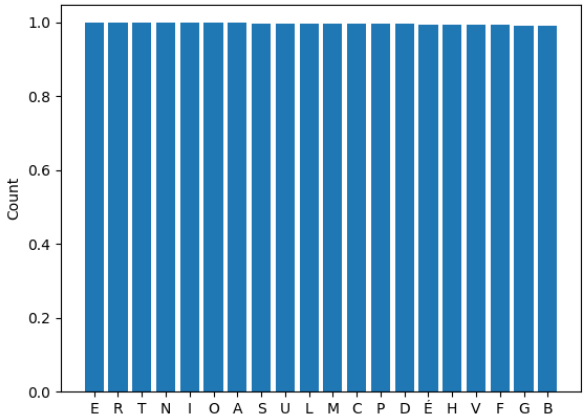

Fig. 6. Relative errors for the decreasing probability count with a $1/2^k$ factor

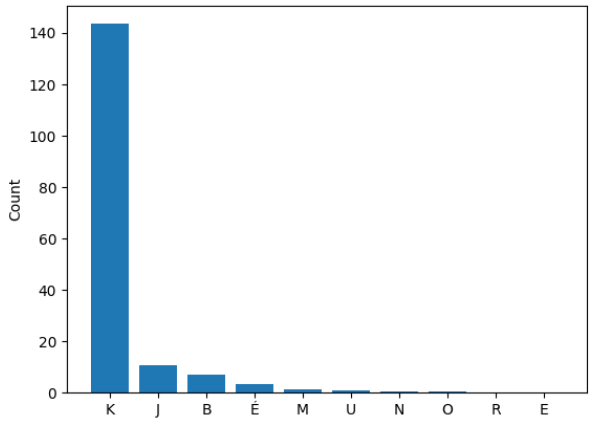

Fig. 8. Relative errors for the space-saving count

The performance of the space-saving count for different k values (3,5,10) can be seen in the table blow, regarding absolute and relative errors:

	k = 3	k = 5	k = 10
Minimum Absolute Error	30690	13424	0
Average Absolute Error	34902	18069.8	5835.5
Maximum Absolute Error	37842	22324	11056
Minimum Relative Error	3.79	1.36	0.0
Average Relative Error	19.10	7.74	16.76
Maximum Relative Error	39.71	23.42	143.58

The absolute and relative errors for the space-saving count are presented in the following graphs:

C. German

The following table presents the 10 characters with the highest count in the three different algorithms: exact count, decreasing probability count with a factor of $1/2^k$ and space-saving count.

Exact count		Approximate count		Space-saving count	
E	10082	E	14	E	10083
N	5627	H	13	R	6322
T	5367	N	13	O	6321
R	5362	S	13	T	6319
S	4248	R	13	G	6319
I	4086	I	12	B	6319
H	3700	A	12	M	6319
A	3449	T	12	U	6318
L	3247	G	12	N	6318
U	2510	U	11	K	6318

TABLE III
10 MOST FREQUENT LETTERS IN GERMAN FOR DIFFERENT COUNT METHODS (EXACT COUNT, APPROXIMATE COUNT, SPACE-SAVING COUNT

The absolute and relative errors for the approximate count, using a decreasing probability factor of $1/2^k$, are presented in the following graphs:

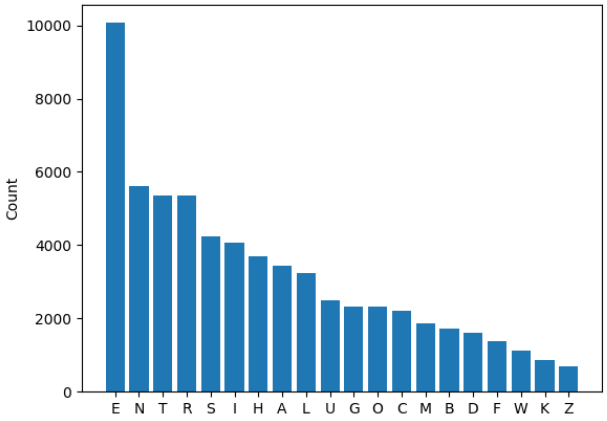

Fig. 9. Absolute errors for the decreasing probability count with a $1/2^k$ factor

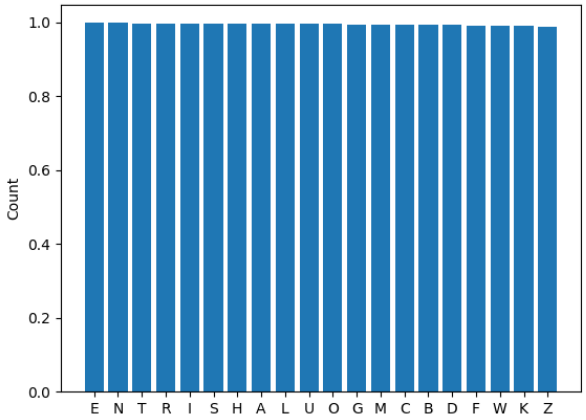

Fig. 10. Relative errors for the decreasing probability count with a $1/2^k$ factor

The performance of the space-saving count for different k values (3,5,10) can be seen in the table blow, regarding absolute and relative errors:

	k = 3	k = 5	k = 10
Minimum Absolute Error	12236	3311	1
Average Absolute Error	17552.33	9285.8	2885.8
Maximum Absolute Error	20432	12524	5452
Minimum Relative Error	1.21	0.32	9.91e-
Average Relative Error	6.87	5.42	1.67
Maximum Relative Error	10.83	14.46	6.29

The absolute and relative errors for the space-saving count are presented in the following graphs:

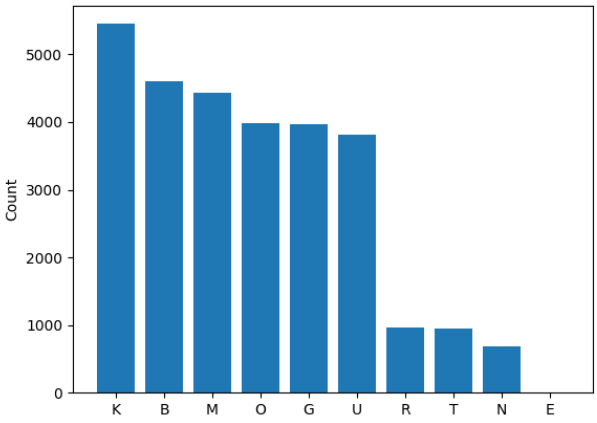

Fig. 11. Absolute errors for the space-saving count

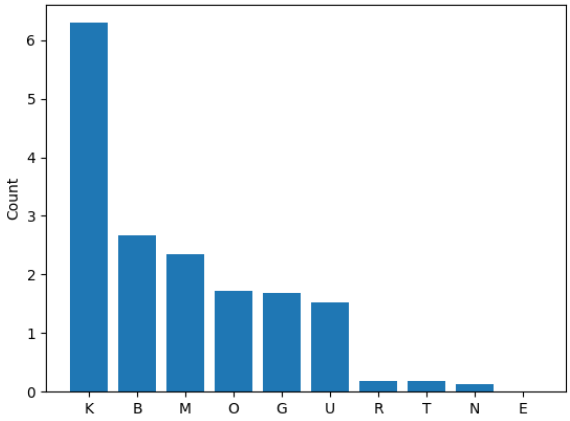

Fig. 12. Relative errors for the space-saving count

D. Dutch

The following table presents the 10 characters with the highest count in the three different algorithms: exact count, decreasing probability count with a factor of $1/2^k$ and space-saving count.

Exact count		Approximate count		Space-saving count	
E	15445	E	14	E	15446
O	7506	I	13	O	8958
N	7049	A	13	T	8956
A	6691	R	13	G	8956
T	6550	O	13	B	8956
R	6546	L	12	R	8956
I	5467	D	12	J	8956
D	5062	C	12	N	8955
L	4438	N	12	K	8955
G	3863	G	12	M	8955

TABLE IV
10 MOST FREQUENT LETTERS IN DUTCH FOR DIFFERENT COUNT METHODS (EXACT COUNT, APPROXIMATE COUNT, SPACE-SAVING COUNT

The absolute and relative errors for the approximate count,

using a decreasing probability factor of $1/2^k$, are presented in the following graphs:

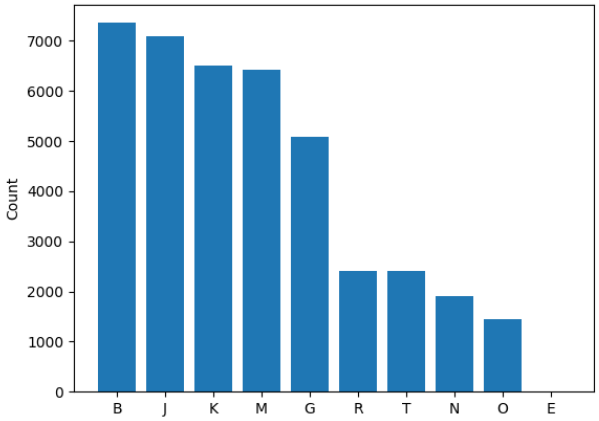

Fig. 13. Absolute errors for the space-saving count

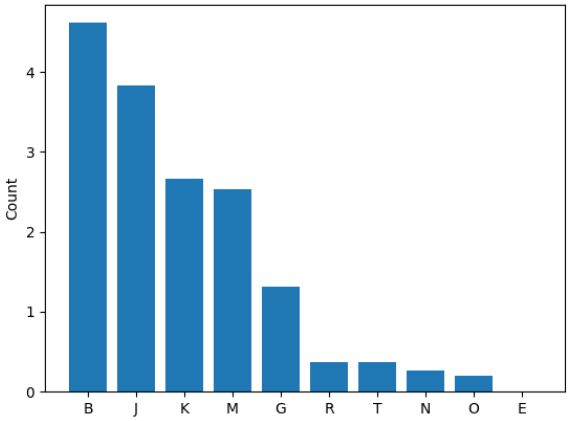

Fig. 14. Relative errors for the space-saving count

The performance of the space-saving count for different k values (3,5,10) can be seen in the table blow, regarding absolute and relative errors:

	k = 3	k = 5	k = 10
Minimum Absolute Error	16571	3765	1
Average Absolute Error	23748.33	13252.4	4065.9
Maximum Absolute Error	30163	17357	7361
Minimum Relative Error	1.07	0.24	6.47e-
Average Relative Error	6.87	4.91	1.61
Maximum Relative Error	16.27	9.36	4.61

The absolute and relative errors for the space-saving count are presented in the following graphs:

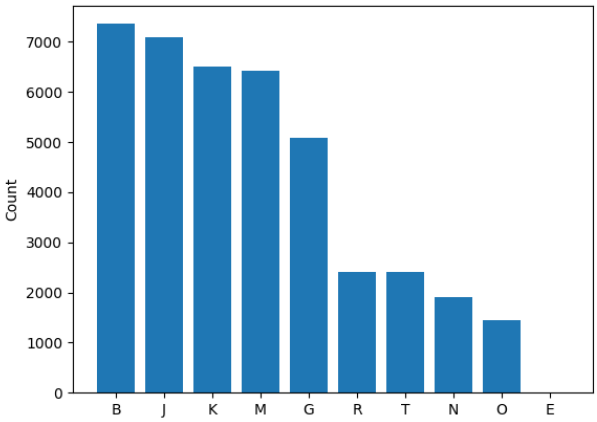

Fig. 15. Absolute errors for the space-saving count

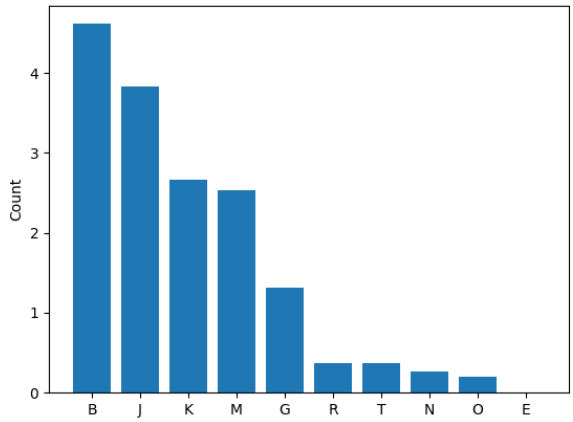

Fig. 16. Relative errors for the space-saving count

VI. Conclusion

According to the obtained results, we can verify that approximate count with decreasing probability of $1/2^k$ is very inaccurate and provides items with very low counts in comparison to the exact count. Approximate counting is a trade-off between accuracy and performance and, in this case, a fixed probability factor would probably be more accurate on providing better results without spending too much memory.

However, this method can be useful since it's somehow precise on returning the most frequent characters in the book, since the it's highly scalable for growing data streams and occupies way less space in memory when compared to exact count.

On the other hand, the space-saving count is more precise than the approximate count algorithm, since the counts of characters are closer to their exact count. Additionally, the sum of the space-saving counts results in the number of elements iterated through the data stream, which did not happen in the approximated count mentioned before, and its data-structure

with fixed-size reduces the amount of memory needed to know the most frequent items.

Ultimately, the size of the data structure determines the accuracy of the estimates. A larger fixed-size will result in more accurate estimates but it will also require more space. A smaller fixed-size will result in less accurate estimates, but will require less space. In this case, even the k=10 fixed-size data structure for space-saving count is not sufficient to provide a good estimate of character counts, although it's better than space-saving count with k=3 or k=5, as seen in the absolute and relative errors for the space-saving count with different k values.

REFERENCES

[1] Project gutenberg (no date) Project Gutenberg. Available at: https://www.gutenberg.org/.

[2] Collections - container datatypes (no date) Python documentation. Available at: https://docs.python.org/3/library/collections.htmlcollections.Counter.

[3] Finding the frequent items in streams of data - Rutgers University (no date). Available at: http://dimacs.rutgers.edu/ graham/pubs/papers/freq-cacm.pdf (Accessed: January 9, 2023).