Comparative Analysis of Exact and Approximate Algorithms for Word Frequency Estimation in Multilingual Texts

Liliana Ribeiro, 108713

***Resumo* – Este artigo apresenta três métodos distintos para identificar frequências de palavras em diferentes ficheiros de texto. O estudo centra-se na análise de três abordagens: contadores exactos, contadores aproximados utilizando um contador de probabilidade decrescente 1/2^k e um algoritmo Lossy-Count para identificar itens frequentes em fluxos de dados. Os resultados apresentam uma análise comparativa da precisão e eficiência espacial destas técnicas.**

***Abstract* - This article employs three distinct methods to identify word frequencies across different text files. The study focuses on analyzing three approaches: exact counters, approximate counters using a Decreasing Probability Counter 1 / 2^k, and a Lossy-Count algorithm to identify frequent items in data streams. The results have a comparative analysis of the accuracy and space efficiency of these techniques.**

***Keys* – Python, Word frequency analysis, Exact counters, Approximate counters, Decreasing Probability Counter, Lossy-Count algorithm, Data stream processing, Algorihtm Comparation.**

I. INTRODUCTION

This article documents the third project for the Advanced Algorithms class.

In this project the focus is to analyzing word frequencies in text data using different methodologies to accurately and efficiently determine the most and least frequent words in text files. By employing both exact and approximate techniques, the trade-offs between accuracy and space efficiency can be evaluated.

Three approaches are explored in this work:

1. Exact counters: For precise word frequency determination.

2. Aproximate counters: Using the Decreasing Probability Counter 1 / 2^k to provide probabilistic frequency estimations.

3. Lossy-Count algorithm: Designed to handle high-throughput data streams and identify frequent items.

Each method is developed and tested on text data derived from literary works.

This article begins with a general explanation of the data types and computational context. In the subsequently section, each algorithm is detailed. Finally, the efficiency and accuracy of the algorithms are analyzed and compared to draw conclusions.

II. COMPUTATION CONTEXT

A. Data Acquisition and Preparation

The foundation of this analysis lies in text files sourced from Project Gutenberg. The dataset consists of six versions of Alice's Adventures in Wonderland by Lewis Carroll, each in a different language: English, Finnish, French, German, Italian, and Esperanto.

To ensure consistency and focus on the relevant content, the following pre-processing steps were performed:

1. Header and Footer Removal: The Project Gutenberg headers and footers were removed to retain only the main text of each book.
2. Stop-word and Punctuation Removal: Stop-words and punctuation marks were eliminated using the *nltk* Python library to focus solely on meaningful words.
3. Case Normalization: All text was converted to lowercase to ensure uniformity during analysis.

III. ALGORITHMS

A. Exact Counter

The Exact Counter is a straightforward and precise method for counting word frequencies in a text. It maintains an exact counter for each unique word encountered, serving as a baseline for comparison with approximate algorithms. While it provides precise results, it has lower space efficiency for large datasets.

For this implementation, the Exact Counter leverages Python's *Counter* data structure, which internally uses a dictionary where keys are words and values are their frequencies. Each time a word is encountered in the text, its counter is incremented by one. This method guarantees precision.

The Computational Complexity is ***O(n)***, where *n* is the total number of words in the text.

In pseudo-code, the algorithm works as follows:

```
1. Initialize an empty dictionary D to store word counts.
2. Tokenize the input text into a list of words words.
3. For each word in words:
     • If the word exists in D, increment its count.
     • Otherwise, initialize its count to 1.
4. Return D, the dictionary containing the exact counts for all words.
```

B. Decreasing Probability Counter

The Decreasing Probability Counter is a probabilistic algorithm designed to reduce memory usage while maintaining approximate word frequency counts. Instead of incrementing the count for every occurrence of a word, the algorithm decides probabilistically whether to update the counter based on the current count value *k*. The probability of incrementing decreases as *1 / 2^k*, making the algorithm more efficient for high-frequency words.

Once the processing is complete, the estimated frequency of a word can be calculated as *2^k - 1*, where *k* is the counter value. This algorithm trades precision for scalability and is particularly useful for processing large data streams.

In pseudo-code, the algorithm works as follows:

```
1. Initialize an empty dictionary D to store approximate counters.
2. Tokenize the input text into a list of words words.
3. For each word in words:
     • If the word is not in D, set its counter k to 0.
     • Generate a random number between 0 and 1.
     • If the random number is less than 1 / 2^k :
          • Increment k by 1.
          • Update D with the new value of k.
4. To estimate the frequency of a word:
     • Retrieve k from D.
     • Compute the estimated frequency as 2^k - 1.
5. Return D with the approximate counts for all words.
```

C. Lossy Count Stream

The Lossy Count algorithm is designed for processing data streams while using limited memory. It maintains approximate counters for frequent items and guarantees bounded error.

The algorithm divides the stream into buckets of size $w=\lceil 1/\epsilon \rceil$, where ϵ is a predefined error parameter. Each item in the stream is either incremented in the dictionary or added with an initial count and a delta representing the potential error. At the end of each bucket, a cleanup operation removes items that are unlikely to be frequent. The algorithm guarantees that any item with a true frequency above ϵN (where N is the total number of elements processed) is retained, with an error bounded by ϵN.

In pseudo-code, the algorithm works as follows:

```
1. Initialize:
     • bucket_size = ⌈1/ϵ⌉, where ε is the error tolerance.
     • D, an empty dictionary to store (count, delta) pairs for each item.
     • current_bucket = 1 and N=0, the total items processed.
2. For each item in the stream:
     • Increment N by 1.
     • If the item exists in D, increment its count.
     • Otherwise, initialize its count to 1 and set its delta to current_bucket−1.
3. At the end of each bucket:
     • For each item in D:
          ◦ If count + delta ≤ current_bucket, remove item from D.
     • Increment current_bucket by 1.
4. To retrieve frequent items:
     • Define a threshold support = support_value * N.
     • For each item in D:
     • Include the item if count ≥ threashold - ϵN.
5. Return the list of frequent items.
```

IV. EXPERIMENTAL ANALYSIS

A. Computational efficiency

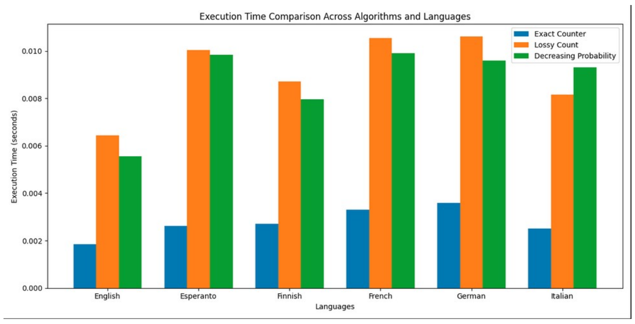

Fig.1 – Execution Time Comparasion Across Algorithms and Languages

The Exact Counter demonstrated significantly faster execution times compared to the other methods. This outcome is expected since the Exact Counter processes each word linearly, with minimal computational overhead. Its straightforward approach of directly incrementing a counter for each word ensures optimal speed, especially for datasets that fit comfortably in memory. However, its computational efficiency will decrease when dealing with very large datasets due to its memory demands, which grow proportionally to the number of unique words.

In contrast, the Decreasing Probability Counter and Lossy Count methods exhibited comparable execution times, with the Lossy Count generally requiring slightly more time. The Lossy Count takes some additional time because of its periodic cleanup phases. These cleanup operations are a trade-off for maintaining memory efficiency, introduce an overhead that slightly impacts execution time.

B. Space Efficiency

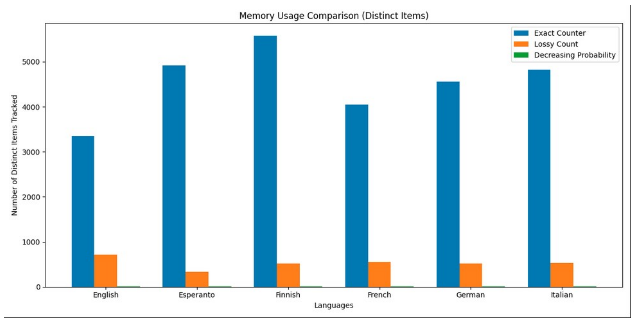

Fig. 2 – Memory Usage Comparasion (Distinct Items)

The Exact Counter consistently tracks the highest number of distinct items for all languages. This is because it maintains an exact count for every unique word encountered in the text. While the exact approach guarantees precise results, its memory usage scales directly with the number of unique words in the dataset. As shown, Finnish, with a higher lexical diversity in the dataset, results in the largest number of tracked items.

In contrast, the Lossy Count algorithm exhibits substantially lower memory usage. By periodically removing low-frequency words during the cleanup phase, it reduces the total number of items tracked. This behavior is evident in the Fig2 graph, where the number of distinct items tracked by Lossy Count is consistently smaller than that of the Exact Counter. Also, the relatively stable memory usage across languages suggests that the algorithm's performance is less affected by the linguistic diversity of the dataset, as it is primarily driven by its error-bound parameter.

The Decreasing Probability Counter shows the lowest memory usage among the presented algorithms. Its probabilistic nature allows it to limit the number of items it tracks by reducing the likelihood of updates as counters grow. As seen in the graph, the memory usage is nearly negligible compared to the Exact Counter and even smaller than the Lossy Count.

C. Error Percentage

To measure the error percentage the exact 10 most common words were used. Each graph evaluates the accuracy of these algorithms by comparing their estimated frequencies to the actual counts.

The Fig. 3 and Fig. 4 show the Error Percentage for German language. Other languages follow a similar pattern.

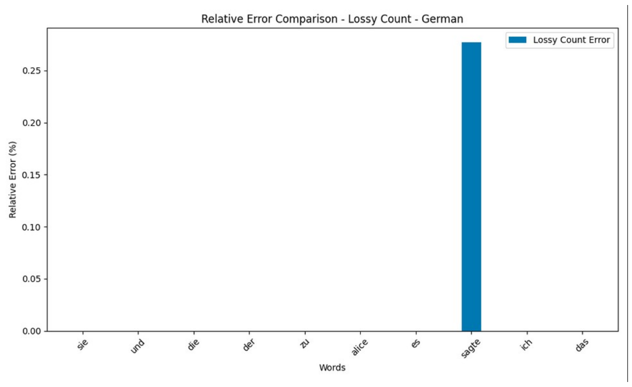

Fig. 3 – Relative Error Comparasion for Lossy Count Algorithm in German

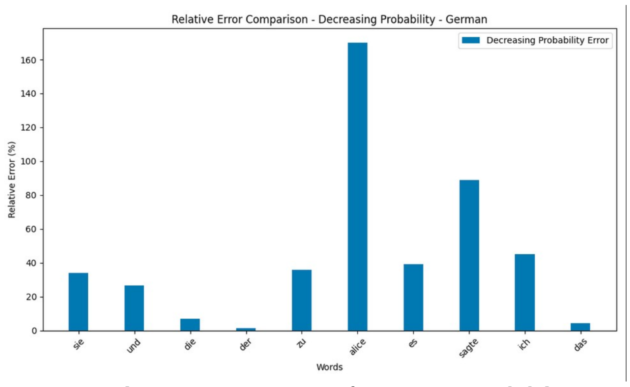

Fig. 4 – Relative Error Comparasion for Decreasing Probability in German

The Decreasing Probability Counter exhibits a wide range of errors across the words analyzed. Common words such as sie, und, and es show moderate relative errors, ranging between 20% and 60%, indicating the algorithm's capability to handle frequently occurring words reasonably well. However, for the word alice, the error surpasses 160%, revealing a significant limitation. This high error arises because the probabilistic nature of the algorithm struggles to maintain accuracy, especially because the probability of incrementing the counter decreases sharply as the frequency grows. .

On the other hand, the Lossy Count algorithm demonstrates remarkable accuracy across the same set of words. Most words, in this case the high-frequency terms like sie, und, and alice, show relative errors of 0%. The Lossy Count’s design provides bounded error guarantees by periodically cleaning up low-frequency items while maintaining accurate counts for more frequent terms. The only notable exception is sagte, which has a slight error of approximately 0.25%. This minor deviation could be attributed to the algorithm's cleanup phase, where words close to the frequency threshold may experience minor inaccuracies.

D. Most common words for each algorithm and different languages

ALG.	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Exact	said	alice	little	“i	one	went	like	could	would	thought
Lossy	said	alice	little	“i	one	went	like	could	would	thought
Prob	said	alice	would	thought	see	went	“i	tell	little	mock

Table 1 – 10 most common words for the English book

ALG.	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Exact	la	kaj	sxi	mi	ne	vi	alicio	diris	al	en
Lossy	la	kaj	sxi	mi	ne	vi	alicio	diris	al	en
Prob	la	kaj	sxi	mi	vi	diris	de	alicio	en	sed

Table 2 – 10 most common words for Esperanto book

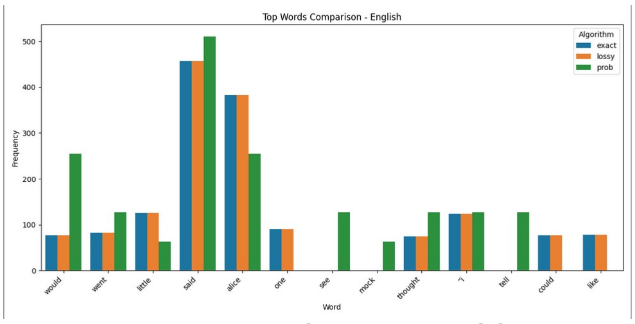

Fig. 5 – Top 10 Words Comparasion English

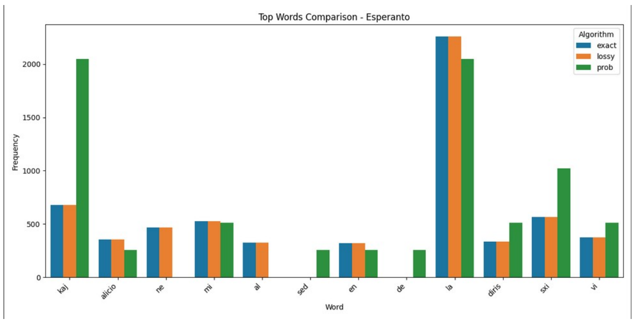

Fig. 6 – Top 10 Words Comparasion Esperanto

The Lossy Count algorithm is almost always align with the Exact Counter, particularly for high-frequency words. The frequencies calculated by Lossy Count are nearly identical to those from the Exact Counter.

The Decreasing Probability Counter exhibits a more variable alignment with the Exact Counter. While the calculated frequencies for specific words deviate significantly it is notable that the top 10 words identified by the Decreasing Probability Counter largely overlap with those from the Exact Counter. This indicates that while the frequencies may not always be accurate, the algorithm is effective in identifying the most frequent words in the dataset. For instance, in Table 1, words like alice, said, would and little appear in both top 10 lists, demonstrating that Prob maintains a reasonable understanding of word importance, even if the rankings differ.

In this analysis the most frequent words reflect the grammatical structure of the respective languages. Articles, prepositions, and common conjunctions, such as ‘la’ and ‘il’ in Italian or ‘and’ and ‘und’ in English and German, consistently dominate the top positions across all languages. Additionally, the name Alice, the main character in the book is consistently present in the top 10 for all languages, despite slight variations in their rankings between algorithms.

E. Lossy Count with different values of n

The analysis of the Lossy Count algorithm across different values of n (5, 10, 15, and 20) reveals that the top 5 most frequent words and their corresponding frequencies remain unchanged. For both English and Esperanto texts, the rankings and counts of the most common words, such as ‘said’ and ‘la’, are consistent regardless of the n value. This indicates that increasing n only affects the algorithm's ability to track less frequent words beyond the most common ones, without impacting the accuracy or order of the top frequent terms.

F. Analysis Conclusions

This analysis highlights the trade-offs between accuracy, computational efficiency, and memory usage across the three algorithms—Exact Counter, Lossy Count, and Decreasing Probability Counter.

The Exact Counter algorithm delivers precise results. Its high accuracy makes it the preferred choice for applications where precision is critical, such as text analytics or linguistic research on smaller datasets. However, the Exact Counter’s memory usage scales directly with the number of unique words, making it less practical for large datasets or real-time applications with significant memory constraints.

The Lossy Count algorithm demonstrates exceptional performance in balancing accuracy and memory efficiency. It closely aligns with the Exact Counter, when looking for the high-frequency words, and maintains consistent rankings across languages. Its memory efficiency, achieved through periodic cleanup of low-frequency items, makes it an excellent choice for applications where bounded error is acceptable, such as detecting frequent patterns in large-scale data streams or identifying trends in real-time systems. Lossy Count’s ability to track frequent words with near-zero error ensures reliability without excessive memory consumption.

The Decreasing Probability Counter stands out for its unparalleled memory efficiency, using a probabilistic approach to drastically reduce the number of items tracked. While its accuracy is lower than Lossy Count, it effectively identifies the most frequent words, with significant overlap in the top 10 lists compared to the Exact Counter. This makes it suitable for scenarios where memory constraints are critical, such as IoT systems, edge computing, or streaming data applications with extremely large datasets. However, its probabilistic nature introduces variability in frequency estimation, making it less suitable for tasks requiring precise rankings or exact counts.

In resume, the choice of algorithm depends on the specific requirements of the task. For small datasets requiring high precision, the Exact Counter is the optimal choice. For large-scale or real-time systems where a balance between accuracy and memory efficiency is needed, Lossy Count offers the best trade-off. For applications with extreme memory constraints, the Decreasing Probability Counter provides an effective solution, althought with reduced precision.

VI. CONCLUSIONS

This study provides a comprehensive evaluation of three algorithms: Exact Counter, Lossy Count, and Decreasing Probability Counter, for identifying word frequencies in text data across multiple languages. Each algorithm offers distinct advantages, making them suitable for different scenarios based on task requirements and constraints.

We draw an analysis and have taken conclusions about the selection of an algorithm for specific requirements of a task. The Exact Counter is optimal for tasks emphasizing precision, Lossy Count is ideal for applications requiring a balance of efficiency and accuracy, and the Decreasing Probability Counter excels in resource-constrained environments where scalability is paramount. These algorithms demonstrate their versatility and applicability across diverse use cases, highlighting their potential for addressing challenges in modern text analysis and data stream processing.

REFERENCES

[1] University of Aveiro "Advanced Algorithms Course Materials."

[2] Wikipedia. "Lossy Count Algorithm" Accessed January 1, 2025. https://en.wikipedia.org/wiki/Lossy_Count_Algorithm

[3] Wikipedia. "Approximate Counting Algorithm." Accessed January 1, 2025. https://en.wikipedia.org/wiki/Approximate_counting_algorithm

[4] Wikipedia. "Brute-Force Search." Accessed January 1, 2025. https://www.nltk.org/

[5] NTLK Documentation "Natural Language Toolkit" Accessed January 1, 2025. https://chatgpt.com/