Algoritmos Avançados
3rd Project - Most Frequent and Less Frequent Words

Miguel Miragaia
Student 108317, DETI
Universidade de Aveiro
Aveiro, Portugal
miguelmiragaia@ua.pt

***Abstract*—The main objective of this article is to identify the most frequent and less frequent words in text files using different methods, and to evaluate the quality of estimates regarding exact counts. Three types of counters were implemented: Exact Counter, Csuros' counter, Frequent Couter.**

***Index Terms*—Countin Algorithms, Exact Counter, Approximate Counter, Frequent Counter**

I. Introduction

Word frequency analysis is a fundamental task in natural language processing and data analysis [1]. It is widely applied in diverse fields such as linguistics, information retrieval, and text mining. Accurately identifying the most frequent and least frequent words in large datasets is crucial for tasks such as keyword extraction, text summarization, and anomaly detection. However, this task often presents significant challenges when dealing with large-scale datasets or data streams, as computational resources and time become critical constraints.

To address these challenges, various counting algorithms have been developed, ranging from exact counters that provide precise results to approximate methods that trade accuracy for efficiency. Approximate methods are particularly useful when working with streaming data, where the dataset is potentially infinite, and it is impractical to store or process all items in memory.

This assignment aims to design, implement, and evaluate counting algorithms that can efficiently and effectively identify the most frequent and least frequent words in text files, while also assessing their computational trade-offs.

A. Objectives

The primary objective of this assignment is to identify the most frequent and least frequent words in text files (books) using different counting methods. Additionally, the assignment evaluates the quality of estimates provided by approximate methods in comparison to exact counts. To achieve this goal, three distinct approaches are developed and tested:

- **Exact Counters:** Algorithms that provide precise word frequency counts.
- **Approximate Counters:** Algorithms that estimate word frequencies with reduced computational and memory overhead, at the expense of some accuracy.
- **Frequent Item Algorithm:** A method specifically designed to identify frequent items in data streams, where storing the entire dataset is infeasible.

An analysis of the computational efficiency, accuracy, and limitations of these approaches is carried out to provide insights into their practical applicability in different scenarios.

II. Counters

A. Exact Counter

The Exact Counter method is the simplest and most intuitive approach. It involves directly counting the occurrences of each word in the text, providing precise results. This method operates by iterating through all words in the input text, incrementing a count for each word encountered [2].

The computational complexity of this algorithm is $\mathcal{O}(n)$ for processing n words, with an additional cost proportional to the number of unique words for maintaining a dictionary of counts. While the Exact Counter guarantees accuracy, it requires storing all unique words and their counts in memory, making it impractical for large datasets or memory-constrained environments.

Advantages:

- Provides exact word counts.
- Straightforward implementation and interpretation.

Disadvantages:

- High memory usage, proportional to the number of unique words.
- Computationally expensive for very large datasets.

B. Csuros' Counter

Csuros' Counter provides an approximation of word frequencies using random sampling. The algorithm selects a subset of the words from the input text based on a pre-defined sampling rate, counts their occurrences, and scales the results to estimate the actual frequencies [3].

Mathematically, let S represent the sampled subset of the text, and $f(w)$ denote the frequency of word w in S. The estimated frequency for w in the original text is given by:

$$\hat{f}(w) = \frac{f(w)}{\text{sampling rate}}.$$

This approach [4] significantly reduces memory usage and computation time, especially for large datasets, at the cost of introducing estimation errors. The accuracy depends on the sampling rate and the distribution of word frequencies in the text.

Advantages:

- Efficient in terms of memory and computation.
- Suitable for large datasets where exact counting is infeasible.

Disadvantages:

- Results are approximate and can be biased, especially for low-frequency words.
- Accuracy is highly dependent on the choice of the sampling rate.

C. Frequent Counter (Misra-Gries Algorithm)

The Misra-Gries algorithm [5] is a streaming algorithm designed to identify frequent items in a data stream. It operates by maintaining a fixed number of counters, where k determines the maximum number of tracked items. As words are processed, their counts are incremented if they are already tracked; otherwise, a new word is added to the counters if there is space. If the counters are full, all counts are decremented, effectively discarding less frequent items.

The algorithm [6] guarantees that any word with a frequency greater than $1/k$ in the text will be included in the counters. While it does not provide exact counts for all words, it efficiently identifies frequent words with high accuracy.

Mathematical Formulation: Let n represent the total number of words in the text, and let $f(w)$ denote the true frequency of word w. The algorithm provides an approximate count, $\hat{f}(w)$, such that:

$$f(w) - \frac{n}{k} \leq \hat{f}(w) \leq f(w).$$

Advantages:

- Highly memory efficient, requiring at most $k-1$ counters.
- Suitable for real-time processing of data streams.

Disadvantages:

- Does not provide exact counts for all words.
- Low-frequency items are more likely to be excluded from the counters.

D. Comparison of Methods

Each method has its strengths and limitations, making it suitable for different scenarios:

- The Exact Counter is best for small datasets or when accuracy is paramount.
- Csuros' Counter offers a balance between accuracy and efficiency, making it suitable for medium to large datasets.
- The Frequent Counter is ideal for streaming data or when the goal is to identify only the most frequent words.

III. Formal Analysis

A. Exact Counter

The Exact Counter computes word frequencies by processing the text word by word and storing counts in a dictionary. It guarantees precise results at the cost of potentially high memory usage.

Algorithm 1 Exact Counter

1: **Input:** Preprocessed text T
2: Split T into words, $W = T$.split()
3: Initialize an empty dictionary $WORD\_COUNTS$
4: **for all** $WORD \in W$ **do**
5: Increment $WORD\_COUNTS[WORD]$
6: **end for**
7: **Output:** $WORD\_COUNTS$

Complexity:

- **Time:** $\mathcal{O}(n)$, where n is the number of words in the text.
- **Space:** $\mathcal{O}(u)$, where u is the number of unique words.

B. Csuros' Counter

Csuros' Counter estimates word frequencies by sampling a subset of the text. The sampled counts are scaled to approximate the frequencies in the full dataset.

Complexity:

- **Time:** $\mathcal{O}(n)$, as each word is processed once.
- **Space:** $\mathcal{O}(s)$, where $s = r \cdot n$ is the number of sampled words.

Algorithm 2 Csuros' Counter

Input: Preprocessed text T, sampling rate r
Split T into words, $W = T$.split()
Initialize an empty dictionary $SAMPLED\_COUNTS$
for all $WORD \in W$ **do**
  if random() $< r$ **then**
    Increment $SAMPLED\_COUNTS[WORD]$
  end if
end for
Scale $SAMPLED\_COUNTS$ by $1/r$ to estimate Total Counts
Output: $SAMPLED\_COUNTS$

C. Misra-Gries Counter

The Misra-Gries algorithm identifies frequent words in a text by maintaining a fixed number of counters. It provides approximate counts and guarantees inclusion of words above a certain frequency threshold.

Algorithm 3 Misra-Gries Counter

Input: Preprocessed text T, parameter k
Split T into words, $W = T$.split()
Initialize an empty disctionary $COUNTERS$
for all $WORD \in W$ **do**
  if $WORD \in COUNTERS$ **then**
    Increment $COUNTERS[WORD]$
  else if len$(COUNTERS) < k - 1$ **then**
    Add $WORD$ to $COUNTERS$ with count 1
  else
    for all $KEY \in COUNTERS$.keys() **do**
      Decrement $COUNTERS[KEY]$
      if $COUNTERS[KEY] == 0$ **then**
        Remove KEY from $COUNTERS$
      end if
    end for
  end if
end for
Output: $COUNTERS$

Complexity:

- **Time:** $\mathcal{O}(n \cdot k)$, where n is the number of words and k is the parameter determining the number of counters.
- **Space:** $\mathcal{O}(k)$, as only $k - 1$ words are tracked at any time.

D. Comparison of Algorithms

The three counters exhibit distinct trade-offs between computational efficiency, memory usage, and accuracy:

- **Exact Counter:** High accuracy, high memory usage.
- **Csuros' Counter:** Moderate accuracy, reduced memory and time complexity.
- **Misra-Gries Counter:** Approximate results, optimized for streaming and memory efficiency.

These characteristics make each algorithm suitable for different application scenarios, as demonstrated in the subsequent sections.

IV. Development

The development process is divided into three phases: **PreProcessment**, **Main (Run Algorithms)**, and **Analysis**.

A. PreProcessment

The PreProcessment phase is fundamental for ensuring that raw text data is structured and standardized before being analyzed by the counting algorithms. This phase involves the following steps:

1) **Language Detection:** The language of each file is inferred using the `detect_language_from_filename` function, which analyzes the filename to identify keywords such as *english*, *french*, or *german*. This step ensures that appropriate stopwords are used during text cleaning.
2) **Text Cleaning:** The raw text is processed through the `clean_text` function, which includes:
   - Removing Project Gutenberg-specific headers and footers using the `remove_gutenberg_headers` function.
   - Eliminating punctuation and converting the text to lowercase for uniformity.
   - Tokenizing the text into individual words.
   - Removing stopwords based on the detected language using the NLTK library.
3) **File Saving:** The cleaned text is saved in the `PROCESSED_DIR` directory with filenames clearly indicating their processed status. This separation of raw and cleaned data ensures traceability and reproducibility.

Below is the pseudocode for the PreProcessment phase:

The functions implemented in this phase ensure that the input data is devoid of irrelevant content, standardized for further processing, and optimized for accurate results.

B. Main

The **Main** phase is the core of the process, where the preprocessed text files are analyzed using various counting algorithms. This phase focuses on running these algorithms efficiently and measuring their performance in terms of execution time and memory usage. The results are stored for subsequent analysis and comparison.

Algorithm 4 PreProcessment Workflow

Input: Raw text files from `RAW_DIR`.
for all files in `RAW_DIR` **do**
  Detect the language using `detect_language_from_filename`.
  Read the raw text from the file.
  Remove headers and footers using `remove_gutenberg_headers`.
  Clean the text using `clean_text`, which removes punctuation, stopwords, and normalizes the text.
  Save the cleaned text to `PROCESSED_DIR`.
end for
Output: Cleaned text files stored in `PROCESSED_DIR`.

Algorithm 5 Main Workflow

Input: Preprocessed text files from `PROCESSED_DIR`.
for all files in `PROCESSED_DIR` **do**
  Load the preprocessed text.
  for all algorithms (Exact Counter, Csuros' Counter, Misra-Gries Counter) **do**
    Measure the performance using `measure_performance`.
    Record the execution time, memory usage, filename, and algorithm.
  end for
end for
Save all performance metrics to `RESULTS_FILE`.
Output: CSV file with performance metrics.

1) **Performance Measurement:** The `measure_performance` function is used to track the execution time and peak memory usage of each algorithm. Memory usage is monitored using Python's `tracemalloc` module, while execution time is recorded using the `time.perf_counter` method.
2) **Algorithms Used:** The following counting algorithms are implemented and evaluated:
   - **Exact Counter:** Provides precise counts of elements in the dataset.
   - **Csuros' Counter:** A probabilistic algorithm that uses sampling techniques to estimate counts with high accuracy.
   - **Misra-Gries Counter:** A streaming algorithm that identifies frequent elements in a memory-efficient manner.
3) **Metrics Recording:** For each algorithm and file, the following metrics are recorded:
   - Execution time (in seconds).
   - Memory usage (in kilobytes).
   - The filename and algorithm name.
4) **Saving Results:** The results are saved to a CSV file using the `save_performance_metrics` function. This file serves as the basis for analyzing and comparing the performance of different algorithms.

The workflow of this phase is as follows:

This phase is crucial for evaluating the computational efficiency and resource utilization of the implemented algorithms, allowing for an informed analysis in the subsequent stages.

C. Data Analysis and Visualization Phase

In this phase, the focus lies on analyzing and visualizing the performance and accuracy of the algorithms used in the project. Key tasks involve generating comparative metrics, analyzing word frequencies, and saving the results in an accessible format for further examination. The following subsections describe the main activities and objectives of this phase.

1) Performance Metrics Visualization: To evaluate the computational efficiency of the algorithms, two primary metrics are considered: execution time and memory usage. Data regarding these metrics is extracted from performance logs and plotted for comparison across different algorithms. The execution time is measured in seconds, while memory usage is recorded in kilobytes (KB).

These visualizations help identify trends and trade-offs between the algorithms, providing insights into their efficiency across various datasets. Graphical outputs are saved as PNG files for detailed review and integration into reports.

2) Word Frequency Analysis: A significant aspect of this phase is the analysis of word frequencies within the datasets. The focus is on identifying the most frequent and least frequent words in the text data, with the goal of uncovering patterns or anomalies. This analysis compares the performance of exact, approximate (Csuros), and streaming methods for word frequency determination.

The results are saved into a structured CSV file, allowing for a detailed comparison of word rankings across the different methods. These comparisons enable the evaluation of how well the approximate and streaming methods align with the exact counts.

3) Comparative Analysis Across Algorithms: This phase involves comparing the outcomes of exact, approximate, and streaming algorithms. By examining the top-ranked words in each method, the consistency and accuracy of the approximate and streaming methods are

assessed relative to the exact approach. The comparative results are printed for detailed evaluation.

4) Translation and Cross-Dataset Comparison: To ensure consistency and enable cross-linguistic analysis, words are translated into a common language, such as English. This standardization facilitates the comparison of word frequencies across different datasets. For each dataset, the most and least frequent words are compared to identify overlaps and similarities.

The comparative analysis results are stored in a CSV file, highlighting shared words between datasets and providing insights into their linguistic patterns.

5) Summary and Output: The final outputs of this phase include:

- Visualizations of execution time and memory usage for performance evaluation.
- CSV files capturing detailed word frequency rankings across exact, approximate, and streaming methods.
- Translated and standardized datasets for cross-dataset comparison.
- Comparative analysis reports, saved in CSV format, highlighting shared linguistic features across datasets.

These results form the basis for evaluating algorithmic trade-offs and the consistency of approximate methods, while also uncovering patterns in word frequencies across different datasets.

V. Results

A. Values (Word Frequencies)

This subsection analyzes the word frequencies obtained from the three algorithms: Exact Counter, Approximate Counter (Csuros), and Streaming Algorithm.

To evaluate the performance of the Csuros counter, it was tested it with 15 different sample rate values ranging from 0.01 to 0.5. After analyzing the results, it was determined that a sample rate of 0.1 provided the best balance between accuracy and computational efficiency. This sample rate was subsequently used, and repeated a few times, for all comparisons involving the Csuros counter in this study.

Similarly, the Misra-Gries algorithm was tested with a wide range of k values to identify the optimal parameter for determining frequent words. Initial experiments with smaller k values, such as $k = 10, 20, 50, 100$, were unable to reliably capture the most frequent words, as their thresholds were too coarse for accurate detection in the dataset.

Subsequently, higher k values, including $k = 500$ and $k = 800$, were evaluated. As shown in Figure 1, $k = 500$ provided accurate results for frequent words. However, $k = 800$, as illustrated in Figure 2, struck the best balance between accuracy and performance. This value allowed the algorithm to effectively identify frequent words while maintaining reasonable computational efficiency. Consequently, $k = 800$ was selected as the default parameter for all analyses involving the Misra-Gries algorithm in this study.

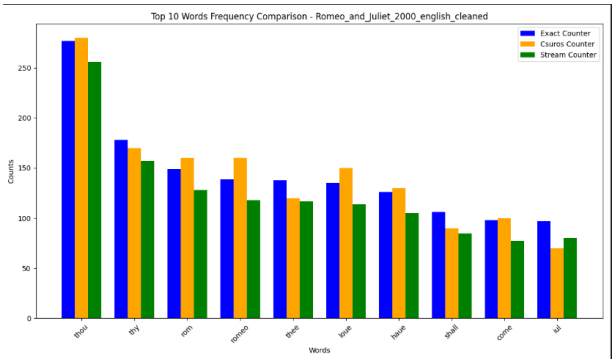

Fig. 1: Top 10 Words Frequency Comparison (k=500) - English 2000

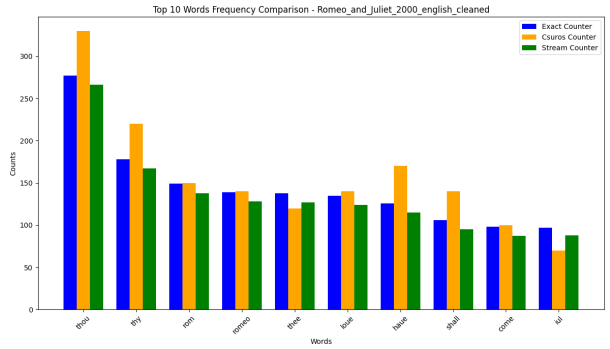

Fig. 2: Top 10 Words Frequency Comparison (k=800) - English 2000

Figure 2 displays the frequency of the top 10 words in the text for the 2000 version of Romeo and Juliet, as computed by each algorithm. The bar chart provides a comparative view of the word counts, with each bar corresponding to one of the algorithms. This visualization allows for an examination of the consistency and differences in word frequency detection among the three methods.

In this example, the most frequent words, such as *"thou"* and *"thy"*, show a high level of agreement across the three algorithms. However, slight variations in frequency counts for certain words, such as *"haue"* and *"shall"*, suggest the potential trade-offs in accuracy for the Approximate Counter and Streaming Algorithm compared to the Exact Counter.

Figure 3 extends the analysis to the 1999 version of Romeo and Juliet. The comparative bar chart highlights similar trends in word usage, with notable differences in the ranking and frequency of certain words such as *"nruse"*. This side-by-side comparison provides insights

into how word frequencies vary slightly across the versions.

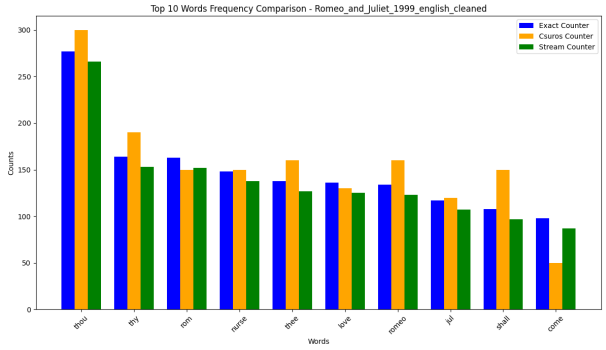

Fig. 3: Top 10 Words Frequency Comparison (k=800) - English 1999

Figures 4 and 5 present the top 10 words detected exclusively by the Approximate Counter (Csuros) and the Streaming Algorithm (Misra-Gries), respectively. These visualizations highlight the frequencies as computed by each algorithm, offering a focused view of their outputs.

These additional visualizations complement the comparative analysis and provide further insights into the word frequencies detected by the approximate algorithms.

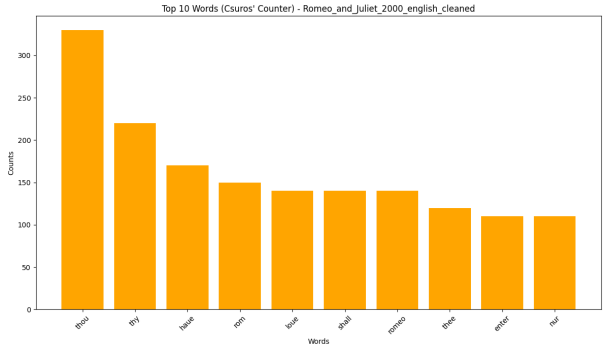

Fig. 4: Top 10 Words Frequency Comparison - Csuros Counter - English 2000

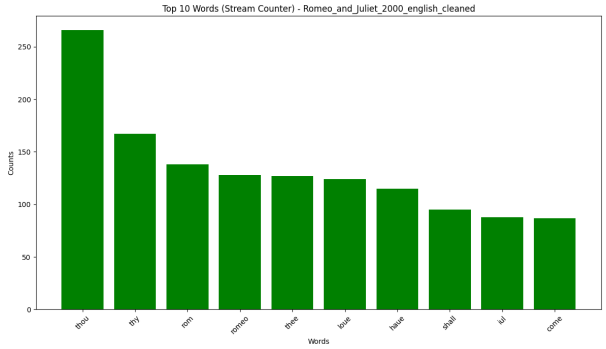

Fig. 5: Top 10 Words Frequency Comparison - Misra Gries Counter - English 2000

These results demonstrate the effectiveness of each algorithm in capturing frequent words while highlighting minor discrepancies. Further exploration of the implications of these differences will be presented in subsequent sections.

B. Relative Order Analysis

Table I presents the word frequency rankings for the 2000 English version of *Romeo and Juliet*. The rankings were computed using three methods: Exact Rank, Csüros Rank, and Stream Rank, applied to both the most frequent and least frequent words.

TABLE I: Summary of Word Ranks for Csuros' and Streaming Algorithms

Word	Exact Rank	Csuros Rank	Frequent Rank	Type
thou	1	1	1	Most Frequent
thy	2	2	2	Most Frequent
rom	3	4	3	Most Frequent
romeo	4	7	4	Most Frequent
thee	5	8	5	Most Frequent
loue	6	5	6	Most Frequent
haue	7	3	7	Most Frequent
shall	8	6	8	Most Frequent
come	9	-	10	Most Frequent
iul	10	-	9	Most Frequent
bucklers	1	-	-	Least Frequent
carry	2	-	-	Least Frequent
coales	3	-	-	Least Frequent
colliars	4	-	-	Least Frequent
mean	5	2	-	Least Frequent
choller	6	-	-	Least Frequent
oth	7	-	-	Least Frequent
collar	8	-	-	Least Frequent
moues	9	-	-	Least Frequent
runst	10	-	-	Least Frequent

Observations:

- **Most Frequent Words:** Words such as *thou*, *thy*, and *Romeo* consistently rank highly, reflecting their thematic importance. Notable updates include:
  - *Thou* retains its position as the most frequent word across all ranking methods.
  - *Rom* ranks 3rd overall, with some variability observed in Csüros Rank.
  - *Loue* and *Shall* emerge as significant narrative-relevant words, with almost consistent rankings across methods.
- **Least Frequent Words:** Words like *bucklers*, *carry*, and *coales* dominate the least frequent list but show significant discrepancies across Csüros and Stream Ranks compared to Exact Rank. These challenges highlight the difficulty of accurately capturing rare terms with probabilistic methods.
  - For example, the word *mean* appears in Csüros Rank but is absent in Misra Gries Rank, illustrating the inconsistencies for low-frequency predictions.
- **Language-Specific Insights:** Interestingly, analysis of the 2004 German version of *Romeo and Juliet*,

presented in Table II, suggests higher accuracy in predicting least frequent words with Csüros and Stream Ranks. Notable examples include:

- Words like *august*, *escalus* and *schlegel* were correctly ranked by Csüros with relatively close alignment to their Exact Ranks.

TABLE II: Comparison of Exact, Csüros, and Stream Ranks for Least Frequent Words in the 2004 German Version of *Romeo and Juliet*

Word	Exact Rank	Csüros Rank	Frequent Rank
william	1		
übersetzt	2		
august	3	1	
wilhelm	4		
schlegel	5	2	
escalus	6	3	
miteinander	7		
vorigen	8		
onkel	9		
franziskaner	10		

Conclusions:

- The most frequent words in the 2000 English version demonstrate consistency across ranking methods, underscoring their thematic prominence in the text.
- Probabilistic methods like Csüros and Misra-Gries counters face significant challenges in accurately ranking least frequent words, particularly in heavy datasets.
- German texts exhibit better alignment for least frequent word predictions, suggesting that language-specific features may influence the effectiveness of ranking algorithms.

C. Word Frequencies Across Different Book Versions and Languages

The word frequency analysis across multiple versions and translations of *Romeo and Juliet* (Table III and IV) reveals insights into linguistic variations and algorithm performance. The analyzed texts include English editions from 1999 [8] and 2000 [9], a French translation from 2006 [10], and a German translation from 2004 [11].

TABLE III: Most Frequent Words Across Versions of *Romeo and Juliet*

File1	File2	Common Words
English-2000	English-1999	thy, come, thee, shall, romeo, thou, rom
English-2000	French-2006	romeo
English-2000	German-2004	romeo
English-1999	French-2006	romeo
English-1999	German-2004	romeo
French-2006	German-2004	romeo, and

TABLE IV: Least Frequent Words Across Versions of *Romeo and Juliet*

File1	File2	Common Words
English-2000	English-1999	bucklers, collar
English-2000	French-2006	
English-2000	German-2004	
English-1999	French-2006	
English-1999	German-2004	
French-2006	German-2004	

Observations:

- **Most Frequent Words:**
  - For the English editions (1999 and 2000), common Shakespearean words [7] such as *thy*, *thou*, *thee*, *shall*, *romeo*, and *come* consistently appeared as most frequent. These results confirm the preservation of key thematic elements across editions.
  - Across translations, the word *romeo* was a consistent frequent term, appearing prominently in all versions, including French and German.
  - Notably, the French and German translations shared the word *and*, suggesting structural similarities between these languages for conjunction usage in translation.
- **Least Frequent Words:**
  - Rare words like *collar* and *bucklers* were observed exclusively in the English editions, specifically the 1999 and 2000 versions. These terms likely stem from unique narrative descriptions or scene-specific content.
  - No significant least frequent words were identified in the French and German translations, indicating a lack of overlapping rare vocabulary across languages.
- **Cross-Language Similarities:**
  - The word *romeo* was universally frequent across all versions, demonstrating its central narrative importance.
  - Shared conjunctions, such as *and*, were observed in the French and German texts, highlighting syntactic overlaps in translations.
- **Algorithm Performance:**
  - The algorithms effectively identified shared frequent words like *romeo* across all versions and languages.
  - Challenges were noted in identifying least frequent words across translations, particularly due to the lack of overlap and the influence of metadata in English editions.

Conclusions:

- The consistent identification of narrative-critical words like *romeo* validates the reliability of the

frequency analysis methods in detecting core textual elements.

- Metadata and linguistic differences, such as archaic terms or untranslated proper nouns, impact least frequent word analysis, emphasizing the need for preprocessing.
- The algorithms demonstrate robust performance in multilingual contexts, with notable accuracy in identifying frequent words across translations.

D. Performance

This section evaluates the performance of the word frequency algorithms used in terms of execution time and memory consumption. The results are measured across different versions and languages of the book Romeo and Juliet. Three algorithms were analyzed: the *Exact Counter*, *Csuros' Counter*, and *Misra-Gries Counter*. The metrics include execution time (in seconds) and memory usage (in kilobytes). Line charts are used to visualize the performance of each algorithm for all book versions.

1) Execution Time: The execution time of each algorithm varies depending on the version of the text being processed. Figure 6 shows that the *Csuros' Counter* consistently outperforms the other algorithms in terms of speed, taking the least amount of time to process the text across all book versions. On the other hand, the *Exact Counter*, as expected, has the highest execution time due to its more computationally intensive approach. The *Misra-Gries Counter* provides a balance, with execution times closer to *Csuros' Counter* but slightly slower.

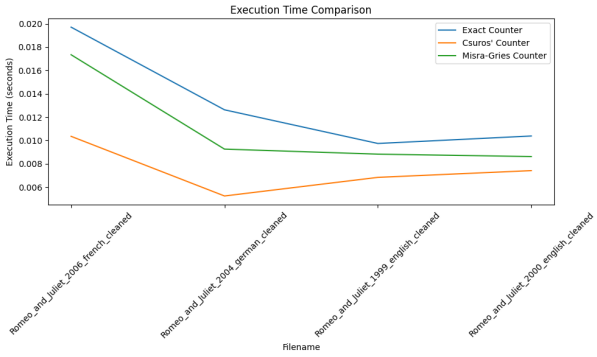

Fig. 6: Execution Time comparison of Counters for each file

2) Memory Usage: In terms of memory usage, as illustrated in Figure 7, the *Misra-Gries Counter* is the most memory-efficient algorithm, requiring significantly less memory compared to the *Exact Counter*. The *Csuros' Counter* also performs well in this metric, with memory consumption slightly higher than *Misra-Gries Counter* but still notably lower than the *Exact Counter*. This aligns with the design principles of these algorithms, where approximate counters prioritize memory efficiency.

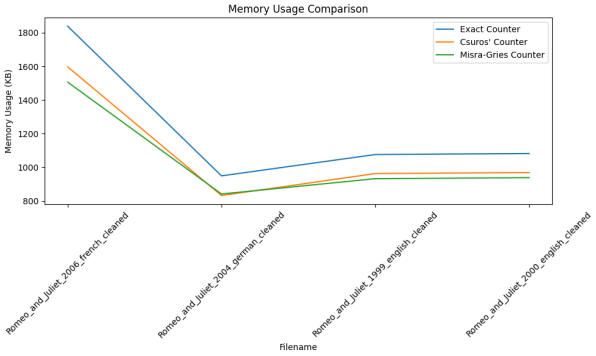

Fig. 7: Memory Usage comparison of Counters for each file

3) Key Observations: Table V summarizes the performance metrics for each algorithm and book version. The results confirm that approximation-based algorithms (*Csuros' Counter* and *Misra-Gries Counter*) offer significant improvements in both execution time and memory consumption compared to the *Exact Counter*. Among the approximation algorithms, the choice between *Csuros' Counter* and *Misra-Gries Counter* depends on whether execution speed or memory efficiency is more critical for the application.

TABLE V: Algorithm Performance for Different Versions of *Romeo and Juliet*

Filename	Algorithm	Execution Time (s)	Memory Usage (KB)
French-2006	Exact Counter	0.0196	1838.74
French-2006	Csuros' Counter	0.0103	1597.28
French-2006	Misra-Gries Counter	0.0148	1506.31
German-2004	Exact Counter	0.0104	949.08
German-2004	Csuros' Counter	0.0050	833.25
German-2004	Misra-Gries Counter	0.0088	841.21
English-1999	Exact Counter	0.0092	1075.89
English-1999	Csuros' Counter	0.0072	962.72
English-1999	Misra-Gries Counter	0.0082	932.31
English-2000	Exact Counter	0.0107	1081.91
English-2000	Csuros' Counter	0.0063	968.76
English-2000	Misra-Gries Counter	0.0102	938.32

E. Errors (Relative and Absolute)

The performance of the algorithms was further evaluated based on their absolute and relative errors across various datasets. Tables were constructed to analyze the errors using metrics such as the lowest value, highest value, average value, and total value. The results are summarized below:

- **Csuros' Algorithm:** This algorithm demonstrates higher absolute error values compared to the Stream algorithm. The maximum absolute error values reached up to 68 in the dataset *English-2000*. The average absolute error values for Csuros' algorithm range between approximately 2.90 and 3.52, with

total errors exceeding 21,000 for some datasets. The relative error for Csuros' algorithm remains higher, with averages around 1.55–1.69 and maximum relative errors of 9.0 across all datasets.

- **Stream Algorithm:** The Stream algorithm shows comparatively lower absolute errors, with maximum values reaching up to 21. Its average absolute error values range between 1.84 and 2.43, with total errors consistently around 8,000–16,800. The relative error for the Stream algorithm is significantly lower than Csuros', with averages of approximately 0.90–0.95 and maximum relative errors of 1.0 across all datasets.

Table VI presents a summary of the errors for both algorithms across different datasets.

TABLE VI: Summary of Errors for Csuros' and Stream Algorithms

Dataset	Error Type	Lowest Value	Highest Value	Average Value	Total Value
English-2000	Csuros Absolute	0	68	3.43	13,134
English-2000	Csuros Relative	0.0	9.0	1.61	6,166.21
English-2000	Stream Absolute	0	11	2.30	8,800
English-2000	Stream Relative	0.0	1.0	0.90	3,459.51
English-1999	Csuros Absolute	0	62	3.52	12,743
English-1999	Csuros Relative	0.0	9.0	1.56	5,633.73
English-1999	Stream Absolute	0	11	2.43	8,800
English-1999	Stream Relative	0.0	1.0	0.92	3,345.73
French-2006	Csuros Absolute	0	63	3.02	21,663
French-2006	Csuros Relative	0.0	9.0	1.61	11,588.98
French-2006	Stream Absolute	0	21	2.34	16,800
French-2006	Stream Relative	0.0	1.0	0.95	6,850.47
German-2004	Csuros Absolute	0	42	2.90	12,613
German-2004	Csuros Relative	0.0	9.0	1.69	7,332.31
German-2004	Stream Absolute	0	10	1.84	8,000
German-2004	Stream Relative	0.0	1.0	0.91	3,953.90

From the data, it is evident that while both algorithms produce errors, the Stream algorithm achieves lower absolute and relative errors across all datasets, making it more consistent and accurate.

VI. Conclusion

This study compared Csuros' algorithm and the Stream algorithm using datasets from various versions of *Romeo and Juliet* in different languages. The Stream algorithm outperformed Csuros' algorithm in accuracy, particularly for frequent words, while both algorithms performed well in identifying high-frequency patterns. However, their effectiveness decreased for less frequent words, with greater error variability.

Csuros' algorithm showed higher error values but was more scalable for larger datasets, whereas the Stream algorithm was more stable and accurate for frequent word detection. These results highlight the importance of algorithm selection based on specific task requirements, especially for handling word frequency distributions.

VII. Code Organization

To go along with this report which results were developed in a GitHub repository, there will be a set of files organized in a set of folders, namely:

- **data:** contains the csv result files, graphics folder, raw and processed text folders, and the counters results folders.
  - **graphics:** contains the files with the different visualizations.
  - **raw:** contains the raw files of the different versions/languages of the books.
  - **processed:** contains the processed/cleaned files of the different versions/languages of the book.
  - **results:** contains the folders of each counter algorithm and their JSON files with the word frequencies.
- **src:** contains the utils.py, preprocess.py file, and analysis.py, also contains the counters folder with the different algorithm counters.
  - **utils.py:** file that contains all the utility functions needed.
  - **analysis.py:** file that performs the different needed visualizations.
  - **preprocess.py:** file that performs the preprocessing of the text files.
- **main.py:** main file that runs and measures the performance of the different counters.

References

[1] "Text Analysis Methods," George Washington University LibGuides. Retrieved from: https://libguides.gwu.edu/textanalysis/methods

[2] Byte Insights, "Exploring Fascinating Insights with Word Frequency Analysis," Byte Insights, Medium, 2019. Retrieved from: https://medium.com/@ByteInsights/exploring-fascinating-insights-with-word-frequency-analysis-5da2113df864

[3] Miklós Csűrös, "Approximate Counting with a Floating-Point Counter," Conference Paper. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14031-0_39

[4] C. S. Csuros, "Probabilistic Counting Algorithms for Estimating Frequencies of Items in Large Data Streams," 2002. [Online]. Available: http://www.iro.umontreal.ca/~csuros/papers/probcount.pdf

[5] CyberNotes, "Misra-Gries Algorithm," Medium, 2021. Retrieved from: https://cybernotes.medium.com/misra-gries-algorithm-02d01bc6a3ec

[6] Wikipedia, "Misra–Gries summary," 2024. Retrieved from: https://en.wikipedia.org/wiki/Misra%E2%80%93Gries_summary

[7] Admin Academia, "Basics of Shakespeare Pronouns," Academia.com.sg. Retrieved from: https://academia.com.sg/basics-of-shakespeare-pronouns/

[8] William Shakespeare, "Romeo and Juliet", 1999. Retrieved from: https://www.gutenberg.org/ebooks/1777

[9] William Shakespeare, "Romeo and Juliet", 2000. Retrieved from: https://www.gutenberg.org/ebooks/2261

[10] William Shakespeare, "Roméo et Juliette", 2006. Retrieved from: https://www.gutenberg.org/ebooks/18143

[11] William Shakespeare, "Romeo und Julia", 2004. Retrieved from: https://www.gutenberg.org/ebooks/6996

(a) English 1999 - Top 10 (b) English 1999 - Csüros (c) English 1999 - Frequent

(d) English 2000 - Top 10 (e) English 2000 - Csüros (f) English 2000 - Frequent

(g) German 2004 - Top 10 (h) German 2004 - Csüros (i) German 2004 - Frequent

(j) French 2006 - Top 10 (k) French 2006 - Csüros (l) French 2006 - Frequent

Fig. 8: Comparison of Top 10 Words Frequency Across Different Versions and Methods