Third AA Assignment - Most Frequent Letters

Artur Romão, 98470

Resumo **- Este trabalho tem como objetivo desenvolver e testar três abordagens diferentes de maneira a encontrar quais as letras mais comuns em textos de diferentes línguas: um contador exato, um contador aproximado e um algoritmo de data stream. De seguida, avaliou-se a performance destas abordagens no que toca à eficiência computacional e à qualidade das estimativas, comparando com o contador exato dos caracteres.**

Abstract **- This work aims to develop and test three different approaches for identifying the most frequent letters in text files of different languages: exact counters, approximate counters, and a data stream algorithm. Then, we evaluated the performance of these approaches in terms of their computational efficiency and the quality of their estimates regarding the exact counts of the letters.**

Keywords **- letters, frequency, frequent, errors, exact, approximate, data stream algorithm.**

I. Introduction

In this work, we explore three different approaches to identify the most frequent letters in text files. The goal was to evaluate the performance of these approaches in terms of their computational efficiency and the quality of their estimates regarding the exact counts of the letters. For this matter, the literary work "*Oliver Twist" by* Charles Dickens extracted from *Project Gutenberg* in Dutch, English, French and German was used. First, we process the text files to remove the Project Gutenberg file headers, stop words, punctuation marks, and convert all letters to uppercase. Then, three different strategies were considered - exact counters, approximate counters, and a data stream algorithm - to compute the exact number of occurrences of each letter, and to estimate the k most frequent letters for different values of k. Afterwards, we compare the results of these approaches, and analyze the computational efficiency and limitations of each.

The main program can be run by executing the following command:

```
$ python3 main.py
```

II. Text Processing

Before even starting to think of the different ways we'll develop to answer this most frequent letters problem, it is necessary to process the text files we'll work with. For that, a python script *text_processing.py* was created in order to help with the logic that addresses this problem.

First off, the *Project Gutenberg* header removing. For this matter, a method *treat_data()* was created. Here we loop through the lines of the file and use a flag to check whether the headers were removed or not. While the flag is False, nothing happens, but when the line that ends the top header is read, the flag is updated to True and we start reading the literary work content. There is also an if statement that checks whether the first line of the bottom was reached, if that's true, the loop breaks.

Now, it's time to normalize our content, also known as, remove punctuation, capitalize the letters and transform accented letters into their normal version. For that, a method *capitalize_line()* was developed. This method receives a line as argument and uses libraries *re* and *unicodedata* to take care of the business. *re.sub()* is used to get rid of punctuation and only consider alphanumeric characters, then the *string* method *upper()* capitalizes the whole line, finally *unicodedata.normalize()* normalizes the line to its non-accented version, for example, transforming ÓRFÃO in ORFAO.

Up next, we have the stop words removal. For this matter, a support method *get_stopw_list()* was created. This function receives the stop words file as an argument, reads it, capitalizes every stop word and stores them in a set. A method *filter_stopwords()* was also created to filter each line by the stop words. Now, coming back to the method *treat_data()*, after reading a line, it is capitalized, filtered by stop words (using *filter_stopwords()*) and then it is *yielded* to the main program (returned without the need of an assignment).

And that's it for text processing!

III. Exact Count and Approximate Count

After having our text processing done, we're good to start counting the most frequent characters.

The first method is exact count. This corresponds to simply counting every occurrence of a character in the file, clear as water. In order to do that, we start by creating a dictionary that will work as a letter counter, *exact_counts*. Then, we iterate through each character of each line retrieved by *treat_data() yield* and and check if the character is already a key of *exact_counts*, if not, the counter is started at 1, if so, the counter is incremented by 1.

And that's it for exact count! Now, for approximate count, the assigned counter was Fixed probability counter:

1/16. Following the same approach as before, we created a dictionary *fixed_prob_counts*, that will receive the character counting for this method.

Now, for each character in the file, we'll need to generate a random number between 0 and 1 by doing *random.random()*. If that number is less than 1/16, we check if the character is already a key of *fixed_prob_counts*, if so, we increment the counter, if not, we start it at 1. Else, if the number is greater than 1/16, we simply discard it and move on to the next character.

In order to obtain more precise results, the explained code was run 30 times and the final result was the average of all of them.

Finally, we save the results in a file to help with the plots with the following structure:

```
order letter count
1 E 83159
2 N 41973
3 O 32484
4 R 29390
5 A 27751
6 T 25606
7 I 25576
8 D 24962
9 G 22114
10 L 21988
11 S 17807
12 K 15282
13 H 13338
14 V 10616
15 U 10022
16 M 9992
17 W 8513
18 B 8394
19 J 6897
20 C 6743
21 P 6550
22 Z 6171
23 F 4324
24 Y 866
25 X 187
26 Q 5
```

Fig. 1 - Text file containing the output of an exact counting. In this case, for the file in Dutch.

IV. Data Stream Algorithm

For the Data Stream Algorithm, the Frequent-Count algorithm was assigned. This algorithm is also known as the Misra & Gries algorithm, used to identify the most frequent items in a data stream. It is designed to identify the k most frequent items in the stream, where k is a parameter of the algorithm. It maintains a list of the k most frequent items and a counter for each item in the list, just like the previous counters. If an item is not in the list, it is added to the list with a count of 1. If the list is full, the counter for the least frequent item is decremented by 1, and if the counter for an item becomes 0, it is removed from the list.

For this algorithm, a method *frequent_count()* was created. A dictionary *counts* is initialized as well as a list *frequent_letters*.

In order for this algorithm to run separately, a dictionary *all_files_letters* was created. This dictionary will contain every single character from each file, so, the key will be the language of the file, (ex.: "ENG", "FR", "GE") and the value will be a list containing every normalized character of the file (stop-words not included, everything capitalized, etc.).

So, method *frequent_count()* receives the list *all_file_letters* and parameter k as arguments (k can either be 3, 5 or 10).

We start by iterating through the list of characters of the file and store the count of each one of them in *counts*. Then, we check if the list *frequent_letters* contains the character. If not, we check if *len(frequent_letters)* is less than k, if so, we append the character to the list, if not, we iterate through the list *frequent_letters* and decrement the dictionary *counts* values for each key. If the counter of that character is equal to 0, we remove it from the list *frequent_letters*.

Finally, just like we did for exact and approximate counts, the output of the data stream algorithm should be stored in a text file, regarding the value of k:

```
k order letter count
3 1 D 4
3 2 O 1
5 1 K 3
5 2 E 2
5 3 D 2
5 4 O 1
10 1 E 27717
10 2 K 5
10 3 O 4
10 4 D 3
10 5 A 2
10 6 S 2
10 7 Z 2
10 8 H 1
10 9 W 1
10 10 L 1
```

Fig. 2 - Text file containing the output of the data stream algorithm. In this case, for the file in Dutch.

V. Results

In this section, a few tables are presented in order to organize the results obtained from the different counting methods and give an insight of the most common letters of literary works in different languages.

Order	Letter	Exact Count	Approximate Count
1	E	83159	5209.8
2	N	41973	2632.0
3	O	32484	2032.77
4	R	29390	1843.13
5	A	27751	1725.9
6	T	25606	1605.27
7	I	25576	1598.43
8	D	24962	1553.87
9	G	22114	1390.07
10	L	21988	1359.77
11	S	17807	1109.7
12	K	15282	954.7
13	H	13338	831.27
14	V	10616	667.1
15	U	10022	628.67
16	M	9992	622.1
17	W	8513	528.6
18	B	8394	523.9
19	J	6897	429.13
20	C	6743	423.33
21	P	6550	411.63
22	Z	6171	383.47
23	F	4324	269.77
24	Y	866	51.7
25	X	187	12.53
26	Q	5	0.2

Table 1 - Table with the Exact Counts and Approximate Counts (Fixed Probability of 1/16) for the Dutch literary work.

Order	Letter	Exact Count	Approximate Count
1	E	50025	3127.97
2	R	28594	1778.17
3	I	27342	1700.43
4	N	26021	1623.43
5	A	25928	1609.03
6	S	25841	1606.5
7	O	25361	1584.7
8	T	25358	1581.17
9	L	21469	1333.33
10	D	20667	1300.47
11	C	12776	795.6
12	H	11203	693.73
13	G	10868	682.93
14	U	10858	671.5
15	P	10319	639.63
16	M	9605	594.73
17	Y	7320	461.13
18	B	6747	419.93
19	F	6202	387.2
20	W	6192	382.07
21	K	4364	277.93
22	V	4156	260.7
23	X	943	58.27
24	J	857	53.4
25	Q	602	38.47
26	Z	158	9.83

Table 2 - Table with the Exact Counts and Approximate Counts (Fixed Probability of 1/16) for the English literary work.

.

Order	Letter	Exact Count	Approximate Count
1	E	73657	4605.37
2	I	41376	2587.5
3	R	38727	2425.5
4	T	37643	2370.07
5	A	37522	2349.97
6	N	33625	2105.67
7	S	29837	1866.0
8	O	26255	1647.4
9	U	23402	1462.27
10	L	20764	1286.07
11	M	16169	1009.8
12	C	15327	971.67
13	P	12983	814.1
14	D	12602	787.6
15	V	9390	583.57
16	F	7293	457.83
17	B	6399	404.47
18	G	6127	382.33
19	H	5159	320.9
20	Q	2971	185.1
21	J	2735	171.03
22	X	2115	129.83
23	Y	1899	118.7
24	Z	1286	77.1
25	W	683	41.8
26	K	649	40.0

Table 3 - Table with the Exact Counts and Approximate Counts (Fixed Probability of 1/16) for the French literary work.

Order	Letter	Exact Count	Approximate Count
1	E	67903	4248.6
2	N	36989	2328.8
3	R	28906	1802.8
4	T	27643	1743.17
5	S	24602	1535.73
6	A	22005	1373.63
7	I	20839	1302.3
8	H	20105	1258.13
9	L	18038	1131.07
10	G	17139	1075.97
11	U	16097	1003.2
12	C	12796	793.1
13	O	10588	652.3
14	B	9886	617.73
15	D	9824	609.97
16	M	9048	569.43
17	F	8879	561.1
18	K	7180	447.8
19	W	6304	387.87
20	Z	4337	265.03
21	V	3318	207.97
22	P	2670	166.03
23	J	805	50.47
24	Y	744	49.33
25	Q	48	2.73
26	X	36	2.17

Table 4 - Table with the Exact Counts and Approximate Counts (Fixed Probability of 1/16) for the German literary work.

By analyzing these tables, we can conclude that Fixed Probability 1/16 gives us a good insight about the most frequent letters occurring in a document, with the calculated approximate count being around 1/16 of the exact count. The order of most frequent letters match for both count approaches in every file.

Regarding the data stream algorithm, Frequent Count, I concluded that there was no reason to show tables for this counting method, because the plots in section VII will speak for themselves. Although, it is important to note that all these values can be consulted in the .txt files in the /results folder.

VI. Absolute and Relative Errors

As we know, there is always an error associated with these algorithms and approximations. And since we have the exact counts, we can calculate both absolute and relative errors, for each approach (fixed probability and frequent count).

The metrics calculated and printed were:

- Average absolute error
- Minimum absolute error
- Maximum absolute error
- Average relative error
- Minimum relative error
- Maximum relative error

```
STATISTICS FOR FIXED PROBABILITY 1/16 oliver_twist_DU.txt:
Average absolute error: 17092.47
Minimum absolute error: 4
Maximum absolute error: 78060
Average relative error: 0.94
Minimum relative error: 0.80
Maximum relative error: 0.97


STATISTICS FOR FIXED PROBABILITY 1/16 oliver_twist_ENG.txt:
Average absolute error: 13693.59
Minimum absolute error: 141
Maximum absolute error: 46984
Average relative error: 0.94
Minimum relative error: 0.89
Maximum relative error: 0.98


STATISTICS FOR FIXED PROBABILITY 1/16 oliver_twist_FR.txt:
Average absolute error: 16823.68
Minimum absolute error: 599
Maximum absolute error: 69181
Average relative error: 0.94
Minimum relative error: 0.92
Maximum relative error: 0.97


STATISTICS FOR FIXED PROBABILITY 1/16 oliver_twist_GE.txt:
Average absolute error: 14017.91
Minimum absolute error: 30
Maximum absolute error: 63831
Average relative error: 0.94
Minimum relative error: 0.83
Maximum relative error: 0.98
```

Fig. 3 - Fixed Probability 1/16 error statistics for each file.

As we can see, for fixed probability, we have high averages for both absolute and relative errors. That is sort of expected, since we're using a 1/16 probability, the obtained results are not going to be similar to the exact count.

Now, we'll analyze the errors for the data stream algorithm having the Dutch file as an example:

```
STATISTICS FOR FREQUENT COUNT (k = 3) oliver_twist_DU.txt:
Average absolute error: 28720.50
Minimum absolute error: 24958
Maximum absolute error: 32483
Average relative error: 1.00
Minimum relative error: 1.00
Maximum relative error: 1.00


STATISTICS FOR FREQUENT COUNT (k = 5) oliver_twist_DU.txt:
Average absolute error: 38969.75
Minimum absolute error: 15279
Maximum absolute error: 83157
Average relative error: 1.00
Minimum relative error: 1.00
Maximum relative error: 1.00


STATISTICS FOR FREQUENT COUNT (k = 10) oliver_twist_DU.txt:
Average absolute error: 22371.70
Minimum absolute error: 6169
Maximum absolute error: 55442
Average relative error: 0.97
Minimum relative error: 0.67
Maximum relative error: 1.00
```

Fig. 4 - Error statistics regarding the Frequent Count Data Stream Algorithm, for the Dutch file and different values of k.

In this file, for k = 3 and k = 5, all the relative errors are rounded to 1.0, which allows us to infer that this algorithm is not precise at all. Although, for k = 10, we find an interesting minimum relative error of 0.67. This error concerns the most frequent letter in that text. As we can see, in Figure 2, the letter "E" was counted 27717 times in the Dutch file, while the next most frequent letter, "K", was only counted 5 times, which is an enormous difference. If we look at Figure 1, we can see that letter "E" was counted 83159 times for exact count and if we compare both values, we find the 0.67 relative error. This tendency also happens with the other files.

These results can be obtained by running the *main.py* program.

With that being said, we can agree that Frequent Count, for k >= 10 gives a good insight about the most frequent letter in the text, but does not guarantee the same conclusions for the other frequent letters. That will be further analyzed in the plot section.

VII. Plots and Analysis

In this section, we'll analyze the plots created for the results of the different counting methods.

A total of 20 plots were created, for each file:

- 1 plot for the exact count
- 1 plot for the approximated count
- 3 plots for the data stream algorithm (for the 3 different values of k)

Here, we'll present the 4 plots regarding the Exact Count for each file, the 4 plots regarding the Approximated Count for each file and, regarding Frequent Count, in order to simplify the analysis, we'll be only presenting the 3 plots for each k (3, 5, 10) for the Dutch file. The rest of the plots can be consulted in the folder /results/plots.

Dutch File Results

Fig. 5 - Plot of the Exact Count results for the Dutch file.

Fig. 6 - Plot of the Approximate Count results for the Dutch file.

Comparing both Exact and Approximate Count results, we can denote that they pretty much look alike in terms of bar height and order of the most frequent letters. Although, the counts for Approximate Count are significantly lower (≃ 1/16 of the Exact Count). The top 5 letters are: E, N, O, R and A.

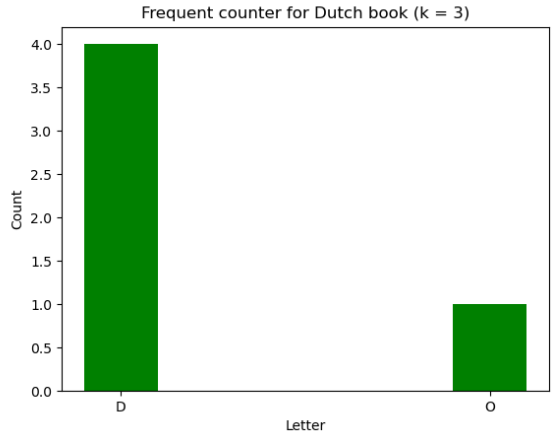

Fig. 7 - Plot of the Frequent Count results (k = 3) for the Dutch file.

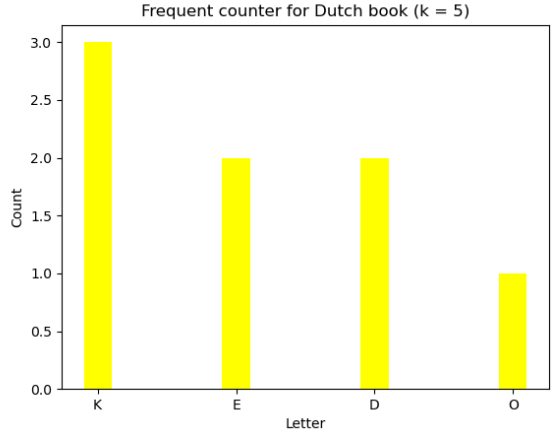

Fig. 8 - Plot of the Frequent Count results (k = 5) for the Dutch file.

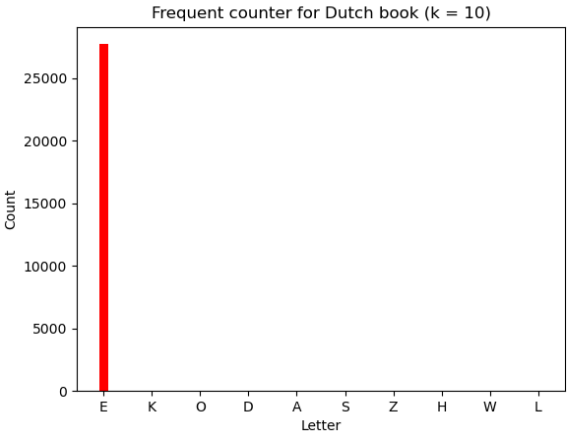

Fig. 9 - Plot of the Frequent Count results (k = 10) for the Dutch file.

Now, looking at the Frequent Count results, it is clear that this count method does not retrieve positive results.

We can see that the order of the most frequent letters does not correspond to the one achieved in Exact Count, for any value of k used. Although, for k = 10, we can see that the letter "E" bar reaches 27717 counts, as registered in Figure 2, of section IV and this corresponds to the most frequent letter of the file, the rest of the bars are not visible because the values are really small: 5 counts for letter "K", 4 counts for letter "O", 3 counts for letter "D", and so on…

This allows us to conclude that even though the data stream algorithm does not provide good results for the order of the most frequent letters (specially for small values of k), from a certain value of k, it starts giving us a good insight about the top frequent letters. If we used, for example k = 20, we could see some more bars in the plot and the order of them would most likely be correct.

This pattern was also observed for the rest of the files, whose plots can be consulted in results/plots.

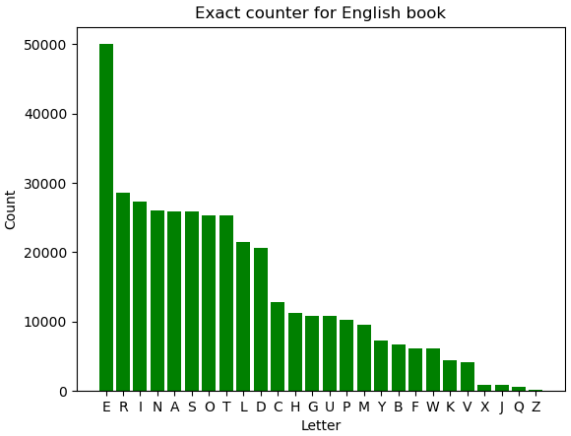

Fig. 10 - Plot of the Exact Count results for the English file.

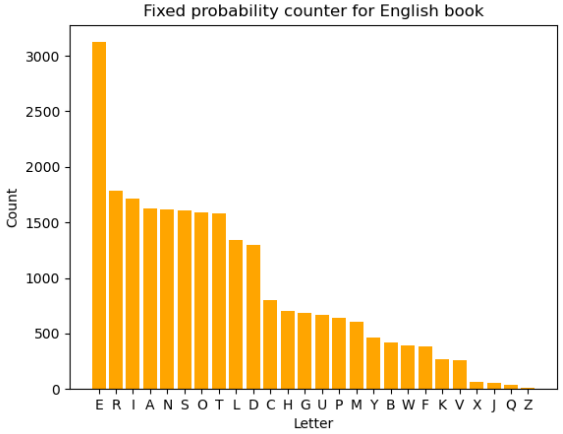

Fig. 11 - Plot of the Approximate Count results for the English file.

By analyzing these plots, we can see that, just like for the Dutch file, the heights of the bars and the order of the most frequent letters are almost coincidental for this file. With the top 5 letters being the same for both count methods: E, R, I, N, A. With a trade on the order of A and N for Approximate Count, because the Exact Counts are really close, as we can see in Table 2 of section V (26021 and 25928).

Fig. 12 - Plot of the Exact Count results for the French file.

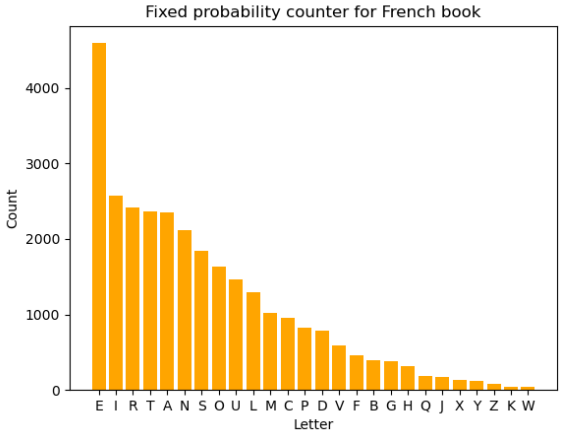

Fig. 13 - Plot of the Approximate Count results for the French file.

Just like the Dutch file, for the French file, the order is the same in both count methods and the bar height is identical. Top 5 letters being: E, I, R, T, A.

Fig. 14 - Plot of the Exact Count results for the German file.

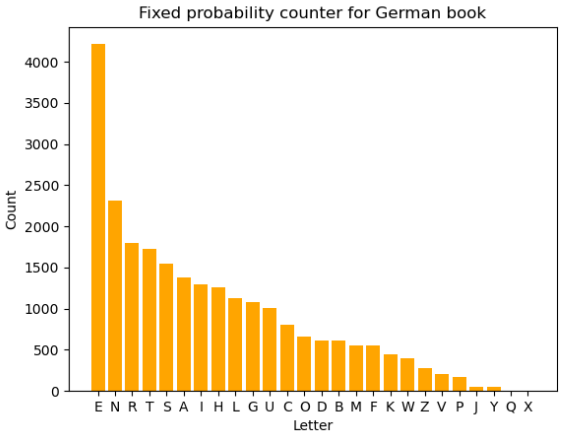

Fig. 15 - Plot of the Approximate Count results for the German file.

Following the flow, the results for the German file are coincidental for both count approaches. The order matches in both Exact and Approximate Count and the height of the bars look just alike. The top 5 letters are: E, N, R, T, and S.

VI. Conclusion

In conclusion, we can say that for a number of tests = 30, Fixed Probability: 1/16 gives us a pretty good insight about the most frequent letters of the literary work “Oliver Twist” in any of the analyzed languages, even though the absolute and relative errors may be significant. The same can’t be said about the data stream algorithm, Frequent Count, for the studied small values of k (3, 5 and 10). Only k = 10 gave us a correct insight about the most frequent letter in every file, but both the order and the counting have bad results. This could be improved by maxing up the value of k.

References

[1] French Stopwords https://github.com/stopwords-iso/stopwords-fr/blob/master/stopwords-fr.txt

[2] German Stopwords https://github.com/stopwords-iso/stopwords-de/blob/master/stopwords-de.txt

[3] Dutch Stopwords https://github.com/stopwords-iso/stopwords-nl/blob/master/stopwords-nl.txt

[4] English Stopwords https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt

[5] Project Gutenberg - Oliver Twist in English https://www.gutenberg.org/cache/epub/730/pg730.txt

[6] Project Gutenberg - Oliver Twist in French https://www.gutenberg.org/cache/epub/16023/pg16023.txt

[7] Project Gutenberg - Oliver Twist in Dutch https://www.gutenberg.org/cache/epub/37093/pg37093.txt

[8] Project Gutenberg - Oliver Twist in German https://www.gutenberg.org/files/56586/56586-0.txt