Distinct Word Occurrence Counter with Bloom Filter

Leandro Silva, 93446
leandrosilva12@ua.pt

***Abstract* –The main objective of this article is to analyse the Counting Bloom Filter when utilizing it to obtain an approximate count of distinct words in a file. An experimental analysis is performed to compare this algorithm with an exact counter in terms of number of false positive rate, bit array size, and number of hash functions.**

***Keywords* –Counter, Probability, Bloom Filter, AMQ, Algorithms**

I. INTRODUCTION

A Bloom filter is a probabilistic data structure that is based on hashing. It is extremely space efficient and is typically used to add elements to a set and test if an element is in a set. Though, the elements themselves are not added to a set. Instead, some sort of fingerprint of the elements is added to the set. [1]

A Bloom filter is very much like a hash table in that it will use one or more hash functions to map an element to k indices of a bit array.

The program that was made could utilize a list of 1s or 0s to store the indices that were set when adding an element, but instead it is utilizing a single integer as storage to be as space efficient as possible. The bits in that integer is what will determine whether an element is present or not.

The hash function utilized in the bloom filter will be the hash function built-in python.

To test if an element is in the Bloom filter, false positives are possible, but not false negatives. This means that it is able to tell that an element is definitely not in the set or that it is possible the element is in the set.

II. DESCRIPTION OF THE PROGRAM OPTIONS

The program has two required arguments used by the word retrieval, being the first one the path of the file to count the words, and the second one the path of the file with the stop words. The third argument is the type of counter, which can be 'exact', for a precise counter without errors, or 'bloom', for a counter based on a Bloom filter.

If the bloom counter is specified,

As the program options varies depending on if certain previous options were provided or not, remember to always use the help option as the command is written.

Here are some examples of how to run the program:

1. Run with an exact counter:

```
main.py texts/english.txt
    stopwords/english.txt exact
```

2. Run with a bloom filter based counter that uses a bit array size of 5000 and 5 hash functions:

```
main.py texts/english.txt
    stopwords/english.txt bloom custom -m
    5000 -k 5
```

3. Run with a bloom filter based counter that uses the provided 1000 expected number of words and the 10% false positive probability to calculate the best bit array size and number of hash functions.

```
main.py texts/english.txt
    stopwords/english.txt bloom optimal
    -n 1000 -p 0.1
```

The mode (3) uses the following formulas to calculate the configurations required in order to have a false positive probability p when receiving n different keys. [2]

$$m = \frac{n * ln(\frac{1}{p})}{ln^2(2)} = -\frac{n * ln(p)}{ln^2(2)} \quad (1)$$

From equation 1, it is clear that for a given false positive probability p, the number of bits in a Bloom filter grows linearly with n.

$$k = \frac{m * ln(2)}{n} \quad (2)$$

In the equation 2, $\frac{m}{n}$ is the number of bits per element in the Bloom filter. So, the optimal number of hashes, k, grows linearly with the number of bits per element.

III. RESULTS AND DISCUSSION

In order to analyse the efficacy of the bloom filter, these tests were made with the bible translated in English. It has 12.566 normalized words, meaning words in lower case, without digits and punctuation, and not presented in the standard English stop words list.

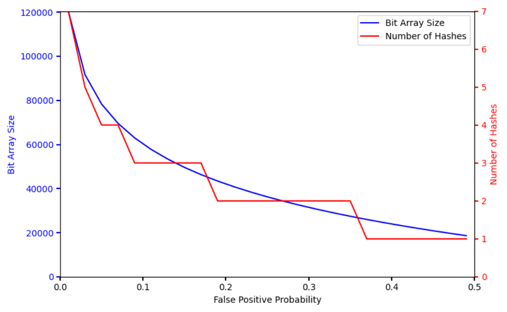

Fig. 1: Variation of the optimal bit array size and number of hashes according to the false positive probability

Figure 1 shows the optimal size of the bit array for a desired fixed false positive rate, as well as the optimal number of hash functions. This is only possible to calculate if having a quite good estimate of the number of distinct words we expect to have. For the making of this graph, the exact 12.566 value was used as the expected number.

From this graph, it is clear that in order to minimize the error of the Bloom Filter, we need to have a greater bit array size and number of hashes. In order to make the error has close to zero has possible, the bit array would need to have more than 120.000 bits, about 10 times the number of distinct words, and more than 7 hash functions, which starts to make the program slower.

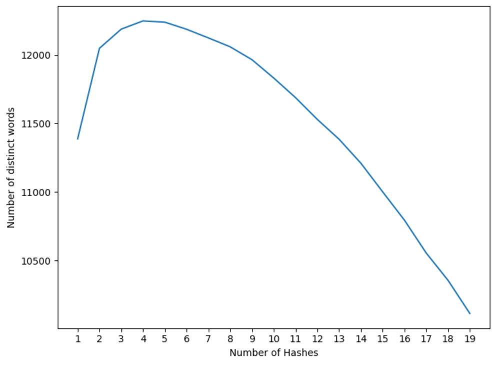

Fig. 2: Variation of the number of distinct word counts in the english text according to the number of hashes

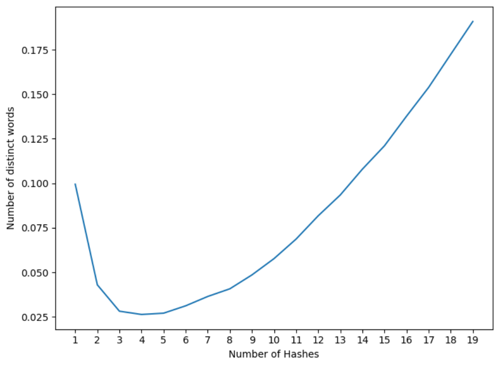

Fig. 3: Variation of the false positive rate in the english text according to the number of hashes

Figure 2 show the resulting count value of the bloom filter counter when specifying a certain number of hashes. In these results, the size of the bit array was fixed to 60.000, which is about the middle value in the interval of the graph on figure 1.

With this, we can confirm that the best number of hash function is between 4 and 5, which is precisely the interval shown on 1 for an array with 60.000 bits. Figure 3 shows the exact same graph, but with a false positive rate instead of distinct words counted.

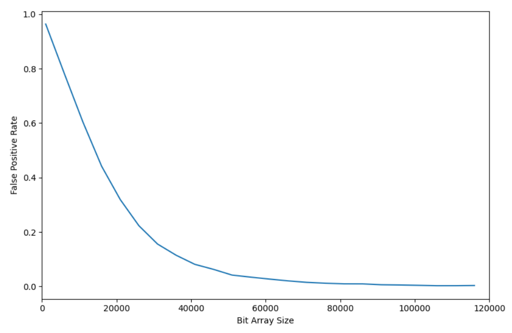

Fig. 4: Variation of the false positive rate in the english text according to the bit array size

When considering uniquely the size of the Bloom filter bit array, it is clear that the greater its number of bits, the greater the number of different keys it will be able to store without collision, and thus, the lower the false positive probability. This is indeed what is represented in figure 4.

IV. CONCLUSION

To conclude this article, I would like to note how remarkable it is that the false positive rate is able to diminish a substantial amount, when increasing the number of hash function till a certain point. It may seem contradictory, as the number of bits occupied is, indeed, greater, thus seeming to lead to a higher false positive rate. However, it is also reasonable to think that, although the probability of one bit being set is higher, the probability of k bits being set is for sure much inferior. Thus, seeking for this optimal number of hashes should be of greater importance when dealing with Bloom filters.

REFERENCES

[1] *Bloom Filter — Brilliant Math Science Wiki.* (2022). Brilliant. `https://brilliant.org/wiki/bloom-filter/`

[2] *corte.si.* (2010). Corte.Si. `https://corte.si/posts/code/bloom-filter-rules-of-thumb/`