Lab 11: Bloom Filters

Tommaso Massaglia
Politecnico di Torino ID: s292988
s292988@studenti.polito.it

I. Introduction

The goal of the lab was to analyze and compare the performance of bloom filters in information storage.

II. Assumptions

The following assumptions are made:

- A dataset of words or sentences is considered for each simulation, each dataset is composed of m items
- Each simulation is ran for b bits
- The total number of keys is n and is computed as 2^b
- The number of hashes used in the bloom computation is k
- The analytical probability of false positives for bloom filter storage is given by:

$$p_{falsepositive} = (\frac{\sum_{bloom\_filter} "1" bits}{n})^k \tag{1}$$

- The theoretical probability of false positives for bloom filter storage is given by:

$$p_{falsepositive} = (1 - e^{-\frac{k \cdot m}{n}})^k \tag{2}$$

- The theoretical probability of false positives for bit-string storage is given by:

$$p_{falsepositive} = \frac{\sum_{bit\_string} "1" bits}{n} \tag{3}$$

- The theoretical probability of false positives for a fingerprint set of b bits for a dataset of m elements is given by:

$$p_{falsepositive} = 1 - (1 - \frac{1}{n})^m \tag{4}$$

- As every $char$ takes one byte to store, the memory occupancy of saving just the strings (in MB) is given by:

$$size_{strings} = \frac{\sum_{word} \sum_{char} 1}{8 \cdot 1024^2} MB \tag{5}$$

- The memory required to store the bloom filter is given by:

$$size_{bloom-filter} = \frac{n}{8 \cdot 1024^2} MB \tag{6}$$

- The memory required to store a fingerprint set is given by:

$$size_{fingerprint} = \frac{m \cdot b}{8 \cdot 1024^2} MB \tag{7}$$

- The optimal number of hashes for a given m and b is given by:

$$\lfloor \frac{n}{m} \cdot log(2) \rfloor \quad \forall\, b, k \;/ b + 3 \cdot k > 128 \tag{8}$$

- If the condition on eq. 8 is not satisfied, invoking the function will give as result 32
- Knowing b, the bloom filter and k, we can approximate the size of the original dataset with:

$$m \approx -\frac{n}{k} ln(1 - \frac{\#"1" bits}{2^b}) \tag{9}$$

III. Input Parameter

Given the assumptions, the following input parameters are considered:

- An hashable dataset: for the simulations considered a dataset of 660000 Italian word was taken from here and a list of over 30M movie titles (which is filtered only for Italian movies down to 3M titles) from here were used
- b: the number of bits we want to run the simulation for (from 19 to 26 in our case)
- k: the number of different hash functions we want to use for our filter (either optimal_hashes or from 1 to 32 in our case)

IV. Output Metrics

The following metrics were used:

- The probability of false positives (eq. 1, 2, 3, and 4)
- The total memory occupancy (eq. 5, 6 and 7)
- The Mean Absolute Error between results:

$$MAE = \frac{|res_1 - res_2|}{\#experiments} \tag{10}$$

V. Data structure and Algorithms

Initially, the words are stored in a $pandas$ dataframe for ease of importing; then, to compute the bloom filter, a $numpy$ array of length 2^b is initialized with $int8$ zeros, to minimize as much as possible its size (as we only need to store 1s and 0s inside).

When the simulation is ran for b, at each step we compute k b-hashes for a word in the word list and change the bits in the corresponding positions to a 1.

The use of eq. 1 (the equivalent of generating a list of words such that by computing the bloom filter on them each possible position in the bloom-string is covered) allows us to compute the probability of false positives much quicker than with other methods and to not have to compute the confidence intervals.

VI. Results

Firstly, the optimal number of hash functions was calculated for m words and b bits using eq. 8

b	optimal k
19	1
20	1
21	2
22	4
23	8
24	17
25	32
26	32

Using the optimal number of hash functions, we then computed and compared the analytical probability of false positives with the theoretical one:

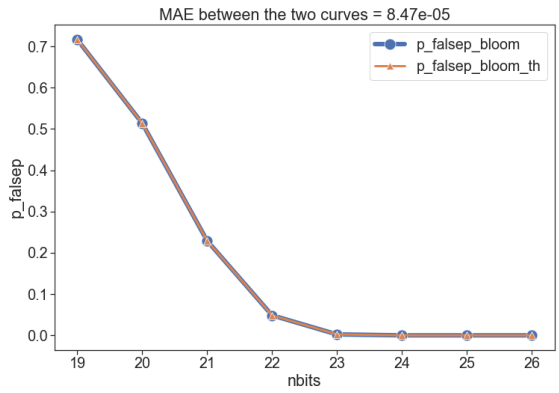

Fig. 1. p_falsep bloom, theoretical vs analytical

Considering the mean absolute error between the two curves, as well as the visual representation of them, we can assert with confidence that the theoretical and analytical results are the same. As expected, as the number of bits increases as the probability of false positives decreases, becoming ≈ 0 when over 23 bits.

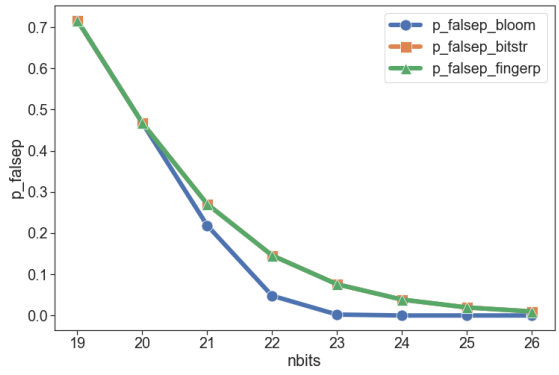

Fig. 2. p_falsep bloom vs fingerprint vs bit-string

Comparing the results saw in fig. 1 with other hashing methods we can notice the increase in performance gained by using bloom filters, as we get the same p of false positives with fewer bits; overall bloom filtering performed the same or better, getting close to no collisions as early as b=2.

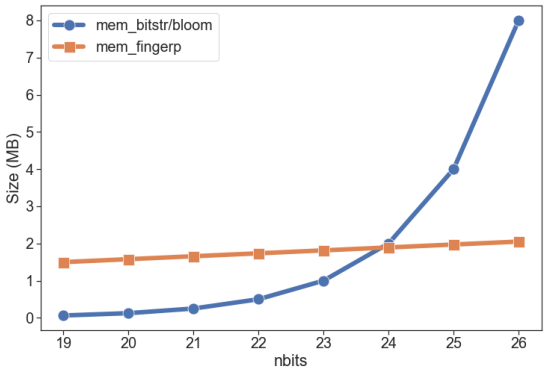

Fig. 3. memory requirements for bloom vs fingerprint vs bit-string. Memory baseline of $7.23MB$ (computed using eq. 5)

Similarly to *bit_string hashing*, the memory requirement for bloom hashing is dependent on the number of bits we decide to use as per eq. 6; figure 3 shows the memory requirements depending on the number of bits used to store our information, but in a sense "lie" as bloom filters would require less bits to reach the same probability of false positives as the other two methods, as seen in fig. 1.

For large values of b, fingerprinting performs better as its requirements are more dependent on the size of the dataset rather then the number of bits used, while bloom filters and bit-strings are suggested whenever we have less restrictions on the probability of false positives or extremely large datasets.

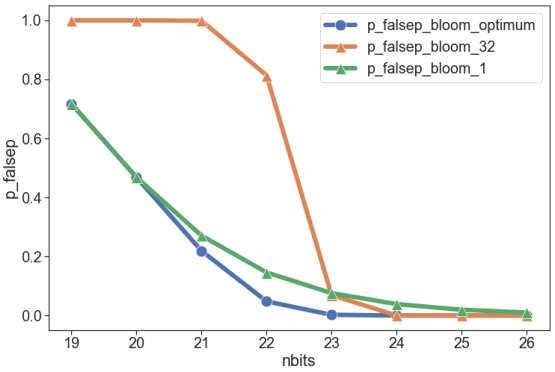

Fig. 4. Probability of false positives, optimum vs k=32 vs k=1

Figure 4 shows the average probability of false positives for h at the optimum, equal to 1 and equal to 32; the two extremes show the different impact of having a non optimum number of hashes when storing values:

- A lower than optimum number of hashes causes the probability of false positives to be higher with higher

values of b, and behaves exactly as bit-string hashing when =1, removing the advantages of bloom filters

- A higher than optimum number of hashes causes the string in which we save the computed hashes to be ”filled up“, leading to awful performances until the string becomes long enough to make this phenomenon not happen

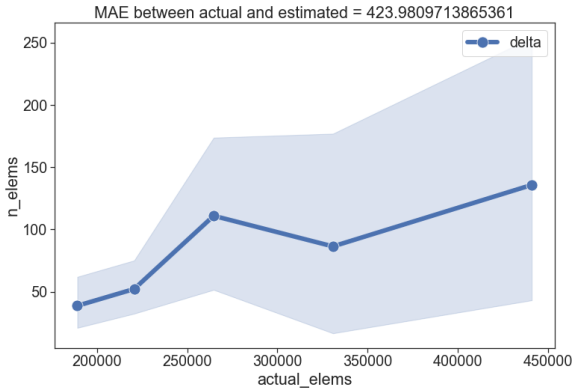

Fig. 5. Delta between actual m and predicted m, ci=95

Lastly, figure 5 shows the difference between the approximated number of elements in the dataset using eq. 9 as the actual size of the dataset increases; considering we start with 200k elements and end up with 600k, a MAE of 423 can be considered a great result and eq. 9 a great approximation as such a difference is meaningless with respect to the size of the dataset.

A. *Movies Dataset*

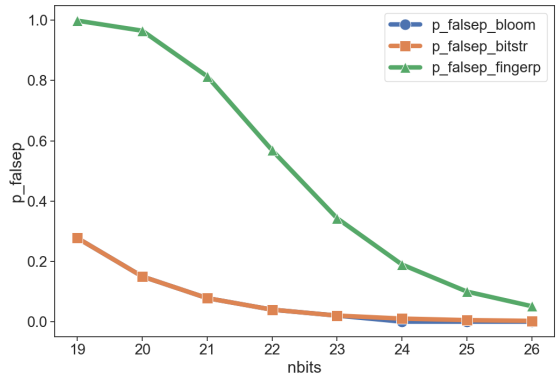

Fig. 6. p_falsep bloom, theoretical vs analytical

Increasing the amount of information we want to store shows even more the advantage of bloom filters and bit-string hashing in storing information; with a bit count as low as 19 we already have under 40% chanche of collisions despite the 3M elements stored.

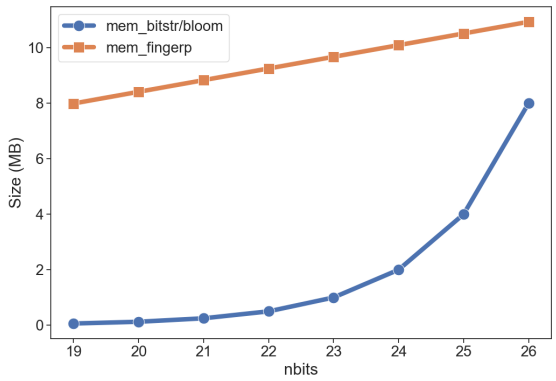

Fig. 7. memory requirements for bloom vs fingerprint vs bit-string. Memory baseline of $64.42MB$ (computed using eq. 5)

The weakness of fingerprinting is shown here as well; for large datasets the dependency of fingerprinting on the size of the dataset makes it not as efficient as the other methods (even if eventually it will be).

VII. Conclusions

Bloom filters represent a very efficient way to store information in short bit strings while keeping a relatively low chance of false positives; what has to be considered though is the time it requires to compute and write on the bit-string for high values of k. As the length of the string and the number of hashes we have to compute increases, the time required to store (or check if present) each new entry gets increasingly high.

Whenever we take into consideration this way of storing information, we have to consider the trade-off between memory and speed, which varies depending on the task at hand.