Data Science Algorithms

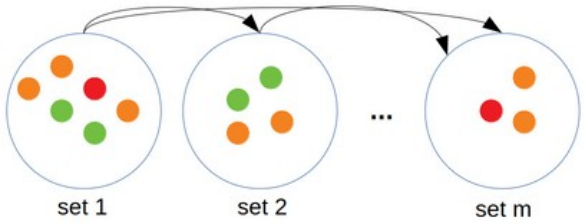
Assignment 1

Full Name – Student ID: Terizi Chrysoula - 430
Date: 09 – 01 – 2020

Part A - Similar objects

(1a)

The assumptions of the problem are that exists a collection of m sets which have at most n elements each one. The total number of combinations is m^2, these are $(set_1, set_2), (set_1, set_3), ..., (set_2, set_1), (set_2, set_3), ..., (set_m, set_{m-1})$. For each of the elements of each set (up to n) will be compared with all the elements of a second set (the comparable set). This means that the worst case requires $O(m^2 n^2)$.



In real conditions the number of sets is large and has consequently that the computation of all pairs is impossible. The comparisons can be reduced to $\frac{m(m-1)}{2}$. To achieve 100 percent accuracy cannot be avoided m^2, nonetheless, n^2 can be improved. A pseudocode is suggested below,

Step 1 For each one of the sets from 1 to m :
Step 2 Apply the *QuickSort* algorithm[1]
Step 3 For each set combination (set_I, set_J) :
Step 4 For each one element in set_I :
Step 5 Apply *Binary Search* algorithm[2]

	QuickSort	Steps 1-2	Binary Search	Steps 3 - 5	Total complexity
Worst case	$O(n^2)$	$O(mn^2)$	$O(\log(n))$	$O(m^2 n \log(n))$	$O(m^2 n^2)$
Average case	$O(n \log(n))$	$O(mn \log(n))$	$O(\log(n))$	$O(m^2 n \log(n))$	$O(m^2 n \log(n))$
Best case	$O(n \log(n))$	$O(mn \log(n))$	$O(1)$	$O(m^2 n)$	$O(m^2 n \log(n))$, if $n \leq 2$ $O(m^2 n)$, if $n > 2$

Table 1: Complexity table for QuickSort and Binary Search algorithms (columns 1 and 3 respectively). Columns 2 and 4 show the complexity time for steps 1-2 and steps 3-5 respectively. In final column, there is the final complexity time of the proposed pseudocode.

Observing Table 1, the proposed algorithm which initially classifies all the sets of objects and then applies binary search for each element of all existing sets of objects succeeds $O(m^2 n^2)$ in the worst case due to the sorting algorithm that has selected. Nevertheless in the average case, $O(m^2 n^2)$ improves on $O(m^2 n \log(n))$ and in the best case can be achieved $O(m^2 n)$. After that analysis, the basic idea is the use of a sorting and efficient search algorithm.

1 C. A. R. Hoare. 1961. Algorithm 64: Quicksort. Commun. ACM 4, 7 (July 1961), 321-402. DOI=http://dx.doi.org/10.1145/366622.366644

2 Thomas N. Hibbard. 1963. An empirical study of minimal storage sorting. Commun. ACM 6, 5 (May 1963), 206-213.

(1b)

Given the four files (*articles_100, articles_1000, articles_2500 and articles_10000, where the number at the end represents the number of users*) to which a user is assigned to an article, similarly users should be identified substantially similar articles. The process of finding similar users is the following:

Step 1: Text representation as a set of integers

Step 2: MinHash Implementation

Step 3: Locality Sensitive Hashing Implementation

Step 1: Text representation as a set of integers

The format of the data is like "*t980 A man was shot … Tuesday.*". initially, each one of the texts is pre-processed to remove symbols, punctuation marks, and stop words[3]. Afterward, σ strings are generated by joining 3 consecutive words from the article, therefore, the representation is transformed into "*t980 = [137860102, 4099950087, … , 1635016701]*". In Figure 1, it is observed that the conversion time (convert terms to hash values, red line) is close to zero regardless of dataset size.

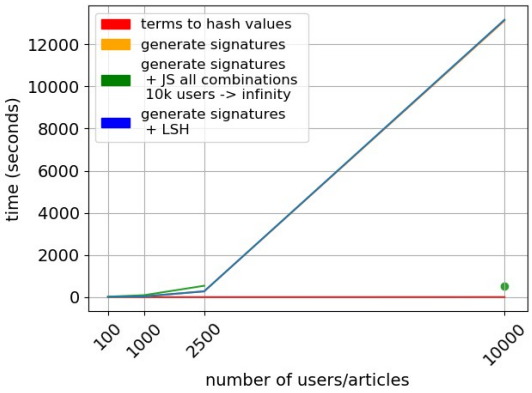
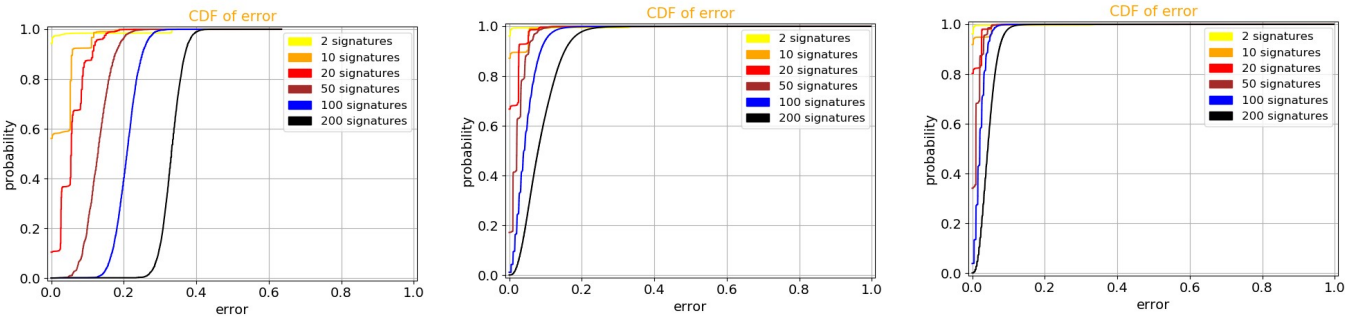


Figure 1: Plot the conversion time in seconds for each one of the four datasets. The red, orange, green and blue lines depict the time of converting a word to a hash value, the time to generate 200 signatures per user, the time to generate signatures and to calculate Jaccard similarity for all possible user comparisons, and finally the time to generate 200 signatures per user and to find potential identical pairs of texts respectively.

Step 2: MinHash Implementation

Six different values for the number of signatures per user were applied, these are 2, 10, 20, 50, 100 and 200. Looking at the Figure 2, the error between the actual Jaccard similarity and this one of signatures for all user combinations depends on the number of signatures and the size of the dataset. As it seems in Figure 2(a), having 2 signatures almost the 95 percent of the pairs have error close to zero and as the number of signatures increases, 20 percent of the pairs have error close to 0.30 for 100 signatures. Observing Figures 2(b) and (c), where the total number of users is big enough compared with this one of 100 users, the error decreases and the percentage of pairs increases slightly. The computation of Jaccard similarity for the dataset of ~10k users is impossible to count. This issue will be addressed in step 3 using LSH.

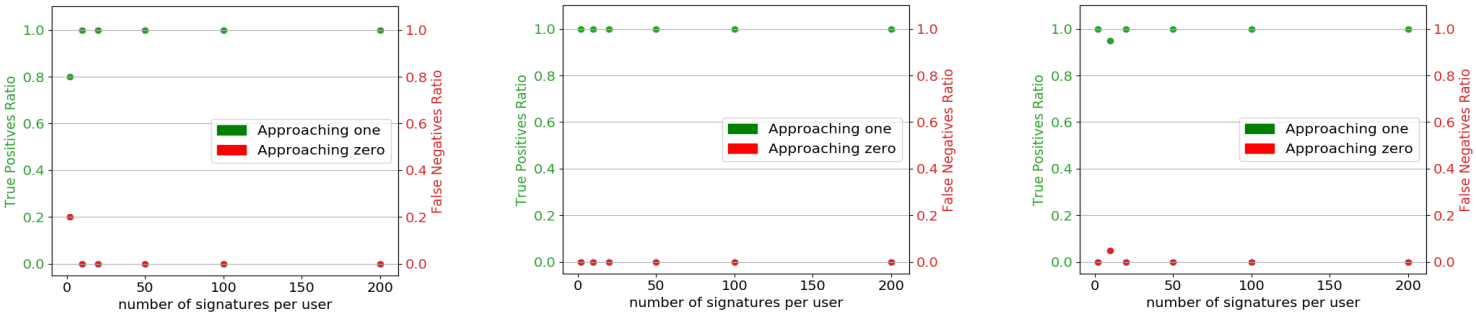


(a) 100 articles **(b) 1000 articles** **(c) 2500 articles**

Figure 2: Cumulative distribution plot for the error between the actual Jaccard similarity and this one of signatures for all user combinations for (a) 100, (b) 1000 and (c) 2500 articles for six different values of number of signatures.

3 Natural Language Toolkit

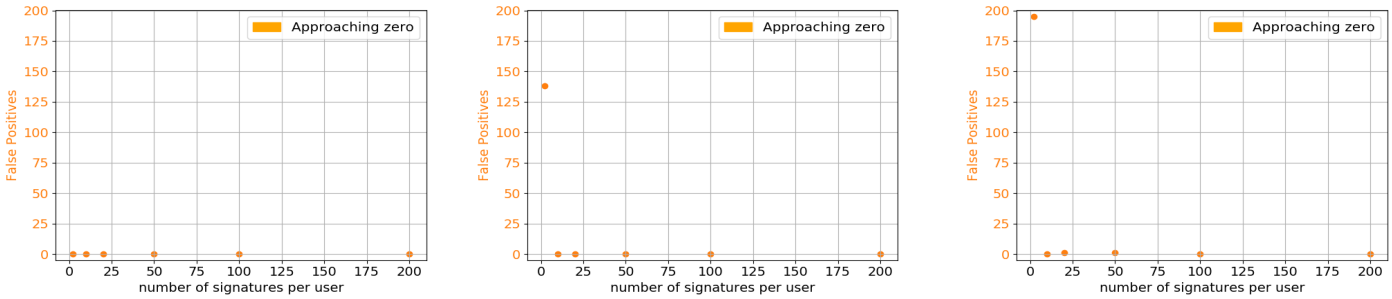
At this point, the effectiveness (convergence to actual Jaccard similarity) of computing the similarity of signatures vectors is studying. The goal is to have high amount of true positives[4] and low amount of false negatives[5]. For relatively small data size and small signature size (see Figure 3(a)), the goal is not achieved. Similar behavior is observed and as the size of the data



(a) 100 articles (b) 1000 articles (c) 2500 articles

Figure 3: True-Positives and False-Negatives ratio plot for (a) 100, (b) 1000 and (c) 2500 articles for the six different values of number of signatures. Similarity threshold = 0.85.

Increases (see Figures 3(b) and (c)). Regardless of the text size, if the size of the signatures is noticeably large, the high true positives ratio and the low false negatives ratio are achieved. About false positives[6] that is a pair of users in not alike, nevertheless incorrectly ends up being predicted as identical. The ideal amount of false positives is to converge to zero. Looking at Figures 4(b) and (c), for a small signature size, a number of incorrect predictions is observed. In the case of the small dataset of 100 articles, false positives not detected.



(a) 100 articles (b) 1000 articles (c) 2500 articles

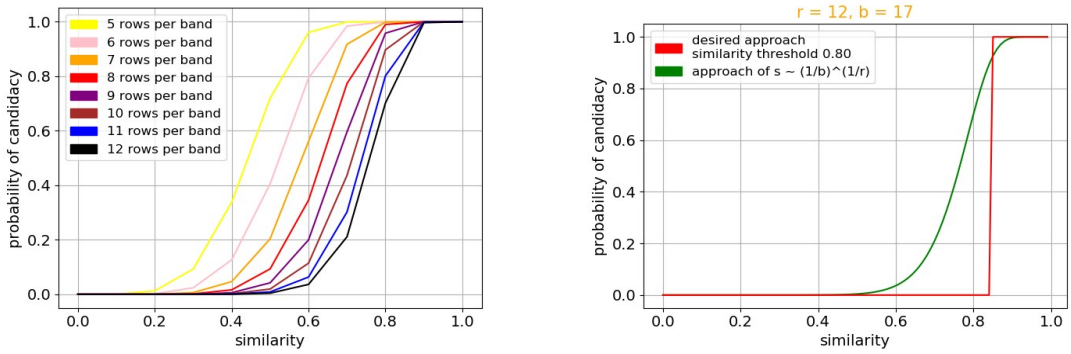
Figure 4: False-Positives plot for (a) 100, (b) 1000 and (c) 2500 articles for the six different values of number of signatures. Similarity threshold = 0.85.

About the conversion time of minhash implementation and the calculation of the actual Jaccard similarity for all user combinations (see Figure 1, orange and green lines), while the minimum time is required for set of 100 articles, from ~1k to ~2k articles the time increases abruptly about ~13 minutes. In case of ~10k users, the calculation converges to infinity due to the inability to calculate ~50m comparisons

Step 3: Locality Sensitive Hashing Implementation

Locality sensitive hashing (LSH) is an algorithmic technique that hashes similar input items into the same buckets with high probability. The number of buckets are much smaller than the universe of possible input items[7]. The technique divides the signatures table into b bands of r rows per band. The maximum number of signatures per user which has been used is two hundred. In Figure 5(a), it is obvious that if $r \in [5,12]$, for $similarity \in [0,1]$ by step 0.10, and focusing on 0.85 similarity value, the probability of candidacy converges one in case of 12 rows per band. Based on Figure 5(b), the particular choice of rows per band achieves low false negative rate and high false positive rate.

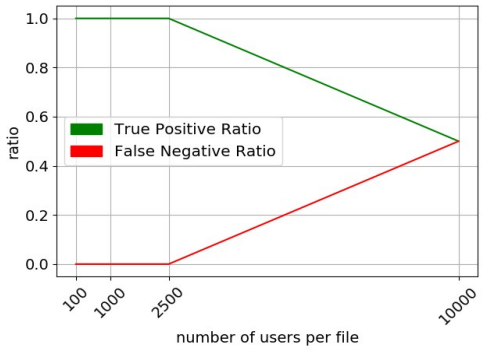
4 The actual class is Yes and the predicted class is Yes
5 The actual class is Yes and the predicted class is No
6 The actual class is No and the predicted class is Yes
7 Rajaraman A. and Ullman J. "Mining of Massive Datasets, Ch. 3", 2010



(a) (b)

Figure 5: (a) Estimate 1 – (1 - s^r)^b for different values of parameter r. (b) Estimate 1 – (1 - s^r)^b for r = 12 and b = 17. The red line declares the desired approach for 0.80 similarity threshold.

After completing the execution of the procedure of local sensitive hashing, the number of possible similar documents is 5, 18, 22 and 80 of 100, 1k, 2.5k and 10k users respectively. It is obvious that the total number of comparisons for calculating users similarity is less than $\frac{m(m-1)}{2}$, where m is the number of users/documents. Based on Figure 6, for small dataset (from 100 to 2.5k documents), the percentage of true positives is 100% without any wrong prediction. Otherwise, for the big dataset, the true positive and false negative ratio is 50% respectively. This means that half of the similar texts were not addressed. This lack of similar documents is attributed to the implementation. After the transformation of terms to numbers, the number of different terms is ~1.2m, an unbelievably large number. For this reason, terms with a frequency greater than or equal to 2 were selected (~213k terms).



About LSH runtime, as can been noticed in Figure 1(blue line), 3 hours are required for the signatures generation and the LSH algorithm.