HW1 DMT

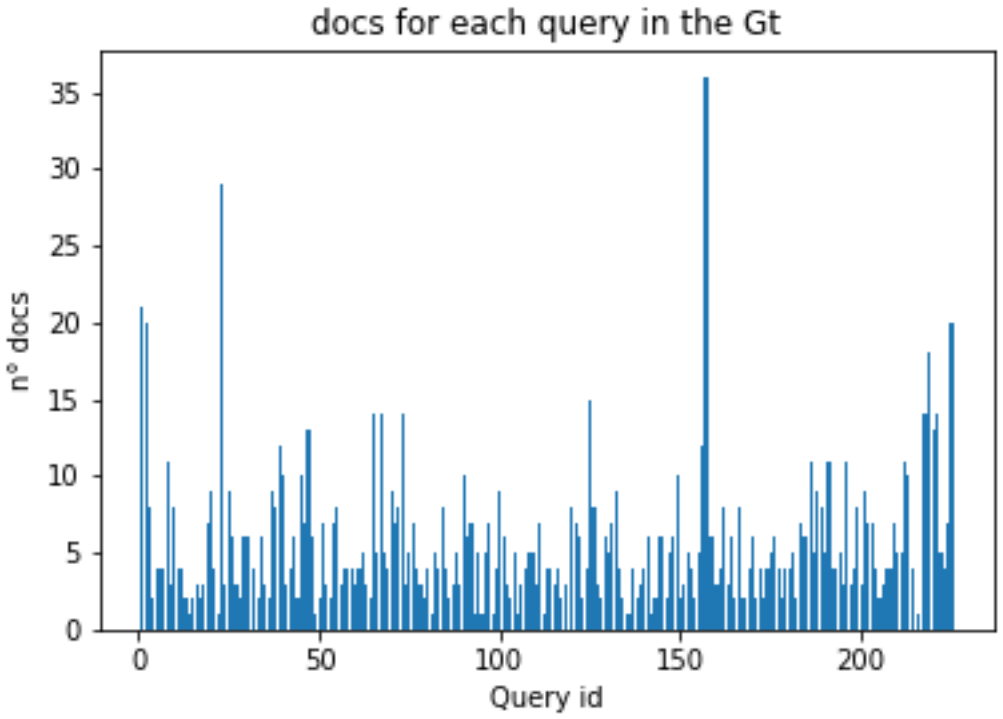
13/4/2018

1658170 Alfonso D'Amelio

1669084 Dario Stagnitto

Ex 1.1

We're going to show some statistics from our data. Next plot shows how many **docs** for each query are in the **ground truth**.



Evaluation Measure for Search Engine

- **P@K**

Search engine	mean(P@1)	mean(P@3)	mean(P@5)	mean(P@10)
SE_1	0.03153	0.03303	0.03303	0.04968
SE_2	0.30180	0.31981	0.35675	0.40153
SE_3	0.23873	0.22672	0.25848	0.31690

- **R-Precision**

Search engine	Mean R-Precision	min R-Precision	1° quartile R-Precision	MEDIAN R-Precision	3° quartile R-Precision	MAX R-Precision
SE_1	0.02256	0.0	0.0	0.0	0.0	0.66666
SE_2	0.25494	0.0	0.0	0.25	0.42857	1.0
SE_3	0.17932	0.0	0.0	0.14285	0.33333	1.0

- **MRR**

Search engine	MRR
SE_1	0.08141
SE_2	0.48630
SE_2	0.39511

- **nDCG**

Search engine	mean (nDCG@1)	mean (nDCG@3)	mean (nDCG@5)	mean (nDCG@10)
SE_1	0.03153	0.03045	0.02874	0.02705
SE_2	0.30180	0.29684	0.27540	0.22221
SE_3	0.23873	0.20865	0.19576	0.16557

Ex 1.2

Considering the problem we have to face on, the best way to evaluate the app is to test our Search Engine through the measures implemented before. Of course not all of them are useful for our evaluation. What do we know about our app? Well, it provides us the output in a **random way**. In according to this, we can not use the nDCG and MRR measure because they strictly depend on the position of the documents. Let's see what we're saying:

$$nDCG(q,k) = \frac{DCG(q,k)}{IDCG(q,k)}$$

$$DCG(q,k) = relevance(doc_1, q) + \sum_{pos=1}^{k} \frac{relevance(doc_{pos}, q)}{log_2(pos)}$$

$$relevance(doc_{id}, q) = \begin{cases} 1 & doc_{id} \in GT_q \\ 0 & otherwise \end{cases}$$

$$MRR(Q) = \frac{1}{|Q|} \sum_{\forall q \in Q} \frac{1}{index_q(FirstRelevantResult)}$$

The *P@K* measure is the best choice for this kind of evaluation because it's the only one which doesn't take into account the position of the relevant documents among the top k (4 in our case). For what concern the *R-precision* also it's not good for our purpose cause it strictly depends on the cardinality of each queries of the Ground Truth.

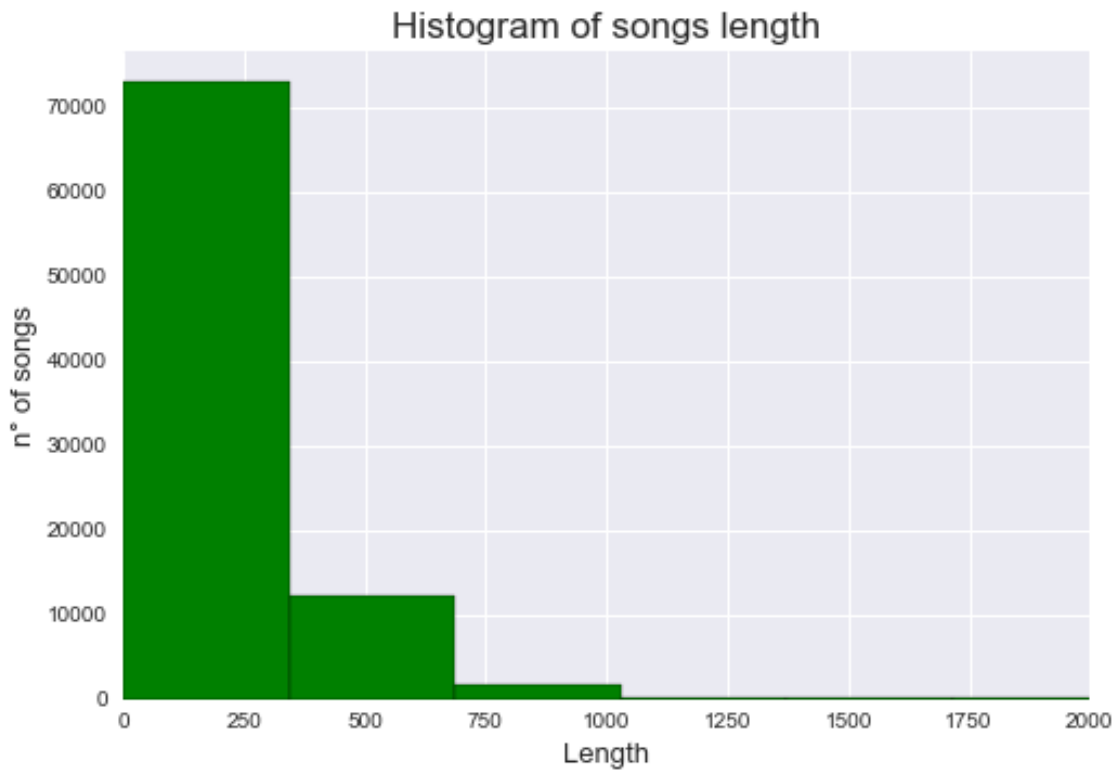
$$R - Precision(q) = \frac{\text{n°of relevant docs in the |GT(q)| positions}}{|GT(q)|}$$

	SE_1	SE_2	SE_3
mean(P@4)	0.3476	0.2529	0.3438

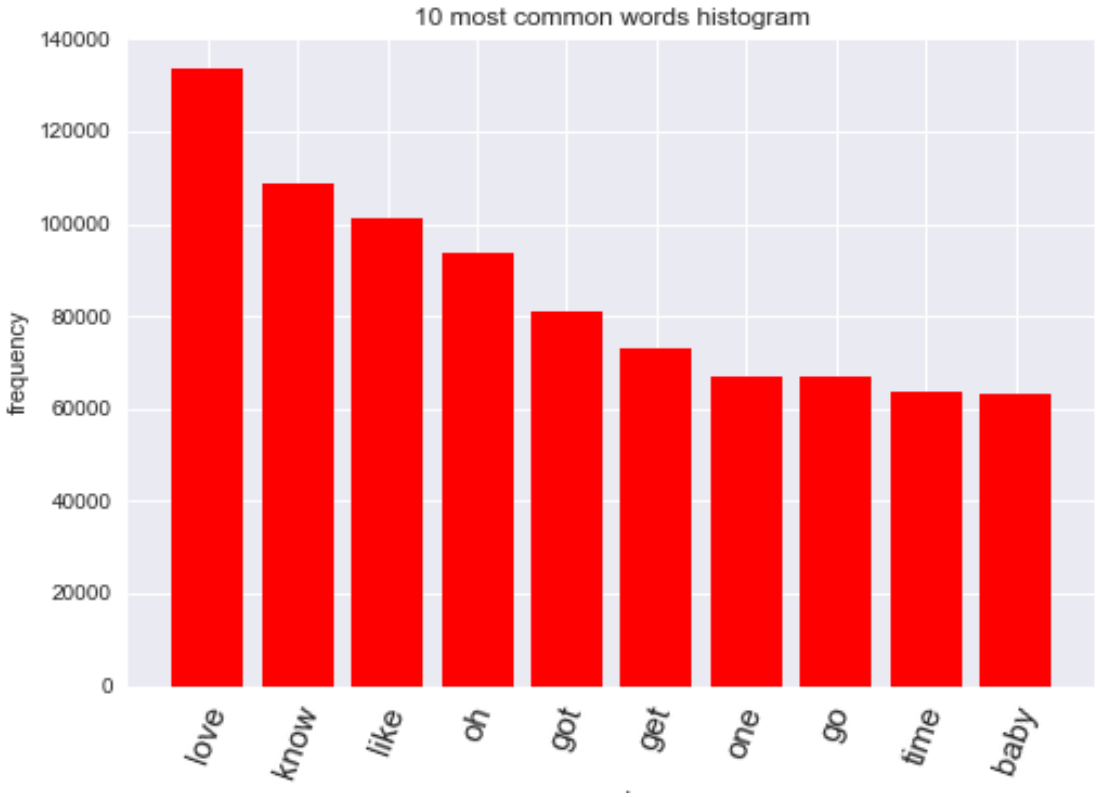
In according to the table above, the best module for the app seems to be the **Search engine 1**.

Ex 2.1

Let's plot some statistics.



We can see how the most of the songs are composed by less than 300 words. Only the 11% of the songs are composed by 500 words (more or less).



Here we computed a statistics about the most common words. 'Love' and 'like' has been resulted the most common as we expected.

Now, our goal is to find out the best **r** and **b** parameters for the **LSH**.

Given the following constraints:

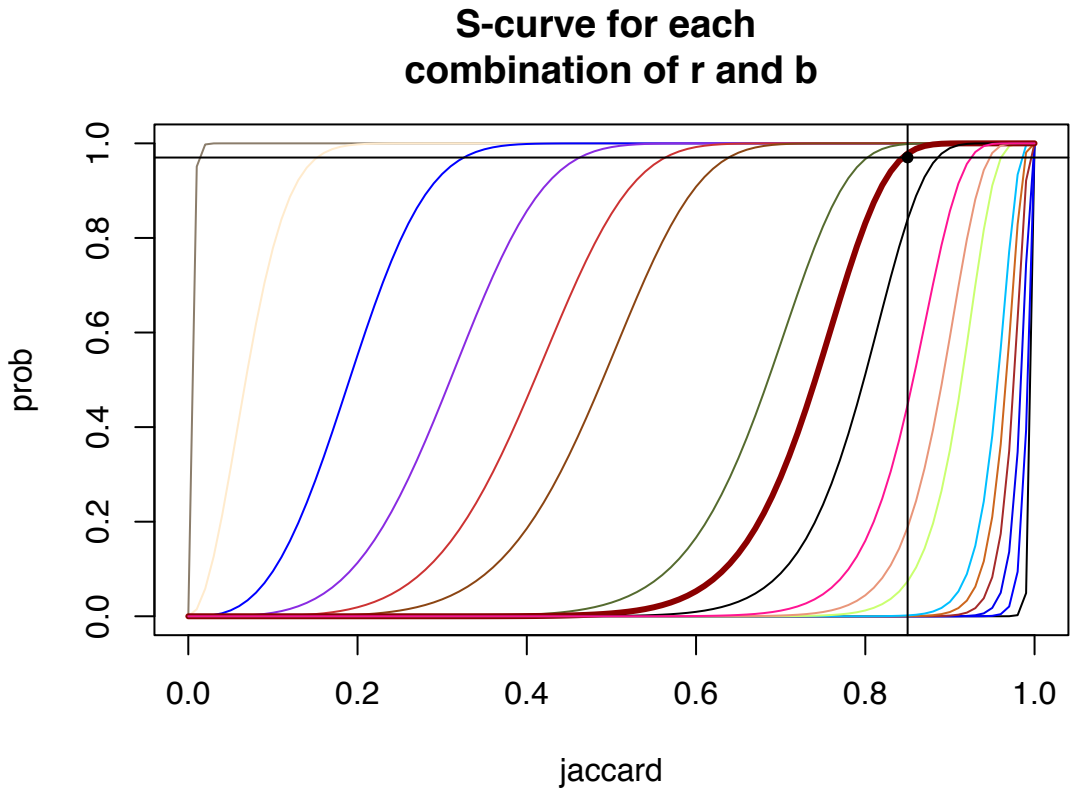
- $n = 300$
- $p \geq 0.97$
- $j \geq 0.85$

and knowing that:

$$p = 1 - (1 - j^r)^b$$

we created a file in which are allocated all the possible combination of r and b. For each of them we computed the formula above, generating a value for p.

For each combination we plotted the so called **S-Curve**, as you can see in the figure below. The best combination, given the constraints above stated, turned out to be $\mathbf{r = 12}$ and $\mathbf{b = 25}$ (the best S-curve represented is highlighted in the plot below).



After sketching each set of shingles in a **Min-Hashing** sketch of 300 length, we applied the Near Duplicates tools given to compute the number of **Candidates** and **Near Duplicates**.

The results obtained are as follows:

- *Candidates* $\rightarrow$ 6761
- *Near duplicate* $\rightarrow$ 3016
- *Number of False Positive* $\rightarrow$ 3745

Ex 2.2 (Part-A)

In this part of the HW we focused on the problem of '*Set size estimation problem*' and '*Union size estimation problem*'.

Starting from the theory:

Suppose we have documents composed by shingles (k=3) as we did in the previous exercise we assign a unique id for each shingles and thanks to the tools given we computed Min-Hashing and LSH:

$$doc_1 = [(all, the, way), (the, way, love), (way, love, you)] \Rightarrow doc_1 = [0, 1, 2]$$

$$doc_2 = [(i, love, my), (love, my, cat), (my, cat, and)] \Rightarrow doc_2 = [3, 4, 5]$$

So in this case the universe set is:

$$U = [0, 1, 2, 3, 4, 5]$$

But how does this Min-Hashing work?

Using Min-Hashing we do a permutation of the universe set thanks to the following formula:

$$h(x) = ((a \cdot x + b)\%p)\%n$$

Suppose we do a first permutation with certain values assigned to a and b:

$$U = [1, 5, 3, 4, 2]$$

These are, after doing n permutation of the universe with different a and b parameters, the Min-Hashing of the doc_1 and doc_2 and this is our starting point in this exercise:

$$\begin{cases} MinHash(doc_1) = [0, 1, 3, 0, 1] \\ MinHash(doc_2) = [1, 2, 0, 1, 0] \end{cases}$$

It means that the first element of the permuted universe is inside doc_1 and we take the position of that value in the permuted universe (i.e zero)... and so on for the next permutation.

By this assumption we can say that:

given a multiple permutation of the universe

$$U = [1, 5, 3, 4, 2]...[3, 2, 4, 1, 5]$$

The Min-hashing computed for the universe set will be obviously composed always by zeros:

$$MinHash(U) = [0, 0, 0, 0, 0]$$

Knowing that:

$$J(Doc_k, Universe_{set}) = \frac{|Doc_k \cap Universe_{set}|}{|Doc_k \cup Universe_{set}|} = \frac{|Doc_k|}{|Universe_{set}|} \Rightarrow |Doc_k| = J(Doc_k, Universe_{set}) \cdot |Universe_{set}|$$

We know apriori that the size of $Universe_{set}$ is equal to 1.123.581.321.

So our aim is to calculate $|Doc_k|$, but first we need to know the jaccard. How can we compute $J(Doc_k, Universe_{set})$?

We simply take each Min-Hash document from the input file, comparing each element of them with each element of the Min-Hash of the universe set, that as far as we said before, is always composed by zeros, counting the elements that are equals and divided them by the cardinality of Min-Hashing universe:

$$MinHash(doc_1) = [0, 1, 3, 0, 1]$$

$$MinHash(U) = [0, 0, 0, 0, 0]$$

In this representation above we have that $J(Doc_k, Universe_{set}) = \frac{\text{n°of equal elements}}{\text{len(MinHash(U))}} = \frac{2}{5}$

In a straightforward way, for the numerator, we can simply count the number of zeros.

So the set size of our $|doc_1|$ will be $\rightarrow J(Doc_k, Universe_{set}) \cdot |Universe_{set}|$

Ex 2.2 (Part-B)

For what concern the **Union size estimation problem**, we approach it in this way:

based on the input given by the set of the sets ids, we zipped the claimed min-hashing docs taking the minimum. For example if the sets_of_sets_id is {0,1}:

$$MinHash(doc_0) = [0, 1, 3, 0, 1]$$

$$MinHash(doc_1) = [2, 3, 0, 1, 0]$$

We zipped the two min hash docs taking the minimum of each tuple:

$$[(0, 2), (1, 3), (3, 0), (0, 1), (1, 0)] \Rightarrow [0, 1, 3, 0, 0]$$

At this point we are able to compute the same point mentioned in PART-A

- $J(Doc_k, Universe_{set})$
- $|doc_1|$