Homework 1

Data Mining Technology for Business and Society

Claudio Battiloro, Lorenzo Giusti

9 April 2019

Part 1 : assignment

In this part of the homework, we had to index a collection of documents and improved the search-engine performance by changing its configurations using the provided set of queries and the associated Ground-Truth. For this part of the homework we had to use the Whoosh API.

Part 1 : Solution report

The indexed corpus of documents is made by 1400 documents.

The performed queries are 222.

We used 24 different configuration of the Search Engine. This could be possible by tuning 3 hyperparameters of the SE:

- The documents index field to search on: “Title”, “Content” or “Title&Content”.
- The Analyzer of the index: “Stemming” or “Standard”.
- The scoring function: “PL2”, “Frequency”, “TF_IDF” or “BM25F”.

We run this configuration and computed $MRR(Q)$ for each of them. Remember that:

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

Where Q is corpus of queries.

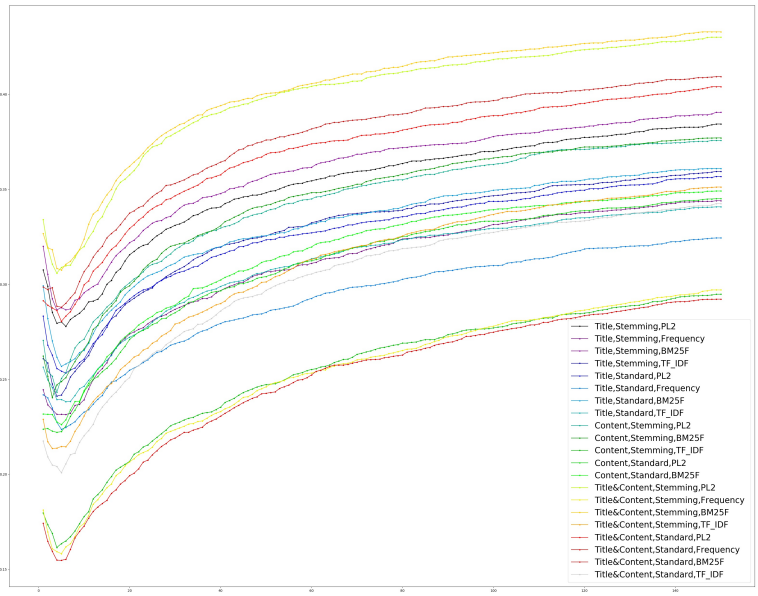
The obtained results are:

Configuration	MRR
Title,Stemming,Frequency	0.4133250
Title,Stemming,BM25F	0.4903998
Title,Stemming,TF_IDF	0.4272384
Title,Standard,PL2	0.4541420
Title,Standard,Frequency	0.3961957
Title,Standard,BM25F	0.4627735
Title,Standard,TF_IDF	0.4161223
Content,Stemming,PL2	0.4359004
Content,Stemming,Frequency	0.2733495
Content,Stemming,BM25F	0.4350341
Content,Stemming,TF_IDF	0.3286768
Content,Standard,PL2	0.3995825
Content,Standard,Frequency	0.2633283
Content,Standard,BM25F	0.4132132
Content,Standard,TF_IDF	0.3068916
Title&Content,Stemming,PL2	0.5089573
Title&Content,Stemming,Frequency	0.3229328
Title&Content,Stemming,BM25F	0.5122717
Title&Content,Stemming,TF_IDF	0.3959903
Title&Content,Standard,PL2	0.4678939
Title&Content,Standard,Frequency	0.3215617
Title&Content,Standard,BM25F	0.4931491
Title&Content,Standard,TF_IDF	0.3861550

We're interested in the configurations that have $MRR(Q) \geq .32$ For each of these and for each query we computed $R - precision(q)$ and summarized the resulting distribution:

Configuration	MRR	R.Precision.Mean	R.Precision.Min	R.Precision.first.Quartile	R.Precision.Median	R.Precision.third.quartile	R.Precision.Max
Title,Stemming,PL2	0.4822354	0.2399626	0	0	0.2500000	0.4000000	1
Title,Stemming,Frequency	0.4133250	0.2112877	0	0	0.1428571	0.3333333	1
Title,Stemming,BM25F	0.4903998	0.2450037	0	0	0.2500000	0.4000000	1
Title,Stemming,TF_IDF	0.4272384	0.2254537	0	0	0.1952381	0.3909091	1
Title,Standard,PL2	0.4541420	0.2109296	0	0	0.2000000	0.3333333	1
Title,Standard,Frequency	0.3961957	0.2033331	0	0	0.1428571	0.3333333	1
Title,Standard,BM25F	0.4627735	0.2122936	0	0	0.2000000	0.3511905	1
Title,Standard,TF_IDF	0.4161223	0.2155331	0	0	0.1666667	0.4000000	1
Content,Stemming,PL2	0.4359004	0.2247137	0	0	0.1742424	0.3715278	1
Content,Stemming,BM25F	0.4350341	0.2266747	0	0	0.1818182	0.3620130	1
Content,Stemming,TF_IDF	0.3286768	0.1343773	0	0	0.0000000	0.2430556	1
Content,Standard,PL2	0.3995825	0.2006326	0	0	0.1666667	0.3333333	1
Content,Standard,BM25F	0.4132132	0.2081182	0	0	0.1666667	0.3333333	1
Title&Content,Stemming,PL2	0.5089573	0.2680925	0	0	0.2500000	0.4285714	1
Title&Content,Stemming,Frequency	0.3229328	0.1288525	0	0	0.0000000	0.2166667	1
Title&Content,Stemming,BM25F	0.5122717	0.2713564	0	0	0.2500000	0.4285714	1
Title&Content,Stemming,TF_IDF	0.3959903	0.1751909	0	0	0.1428571	0.2857143	1
Title&Content,Standard,PL2	0.4678939	0.2527772	0	0	0.2500000	0.4000000	1
Title&Content,Standard,Frequency	0.3215617	0.1328734	0	0	0.0000000	0.2451923	1
Title&Content,Standard,BM25F	0.4931491	0.2531025	0	0	0.2500000	0.4000000	1
Title&Content,Standard,TF_IDF	0.3861550	0.1776843	0	0	0.1428571	0.2964286	1

In the end, we computed and plotted, for each acceptable configuration, the average $nDCG@k$ as a function of k in the interval $\{1, .., 150\}$:



Part 2: Assignment

We had to find, in an approximated way, all near-duplicate documents inside the 261K_lyrics_from_MetroLyrics dataset. The dataset contains data on 261K songs. Two songs are considered near-duplicates if, and only if, the jaccard similarity between their associated sets of shingles computed only on their lyrics is ≥ 0.88. To complete this part of the homework, we had to use the Near_Duplicates_Detection_Tool.

Part 2: Solution report

Q1: What values for 'r,' 'b' and 'n' did you choose?

A1: For the values (r, b, n) we've chosen the values $(25, 85, 2125)$. This values come from a numerical solution of the optimization problem:

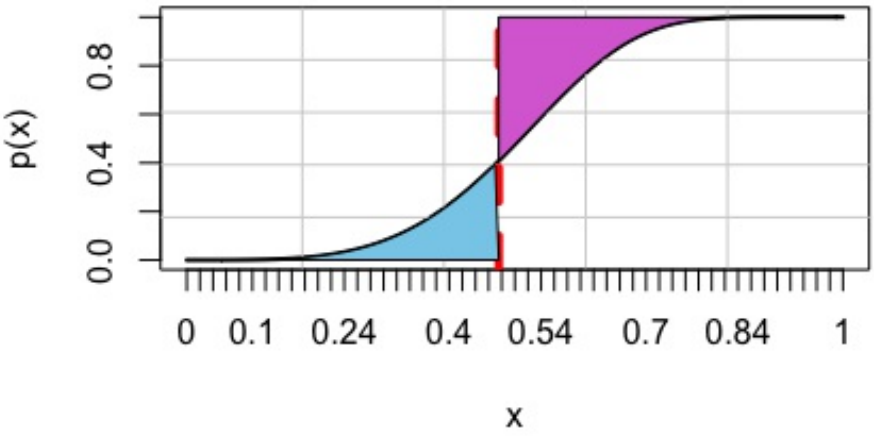
$$maximize \;\; f(r,b) = 1 - (1 - j^r)^b$$
$$s.t. \;\; r, b \in \mathbb{N}$$

With $j = .88$. Of course this is a non convex function so we cannot be sure to have found the optimal solution. But let's start like this. This choice will reach, obviously, good results in terms of FPs and FNs but with a pretty computational effort. The value for *n* comes out immediately from the identity $n = r \cdot b$

Q2: The probability to have False-Negatives for the following Jaccard values: 0.88, 0.9, 0.95 and 1.

Q3: The probability to have False-Positives for the following Jaccard values: 0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55 and 0.5.

A{2,3}: Given the following well known s-shaped figure:



As usual in this kind of framework,the area of the azure region equals the probability of get a false positive out of the LSH mehod and the area of violet region equals the probability of a false negative result out of the LSH method. So, we need to compute the integral of this function in order to get those probabilities. The way to compute the integral for this function is as follow:

$$\int 1 - (1 - j^r)^b \, dj \; = \; \int dj - \int \sum_{k=0}^{b} (-1)^k \binom{b}{k} j^{r \cdot k} \, dj \; = \; j - \sum_{k=0}^{b} (-1)^k \binom{b}{k} \int j^{r \cdot k} \, dj$$

Therefore, the primitive of this integral looks like:

$$F(j) \; = \; j - \sum_{k=0}^{b} (-1)^k \binom{b}{k} \frac{1}{rk+1} \, j^{rk+1}$$

Now, in order to compute the probability of getting a false positive we have to calculate the following integral:

$$\int_0^t 1 - (1 - j^r)^b \, dj \; = \; F(t) - F(0)$$

Let FP be the event that the LSH method produces exactly one false positive candidate, $\mathbb{P}(FP) = F(t)$ since $F(0) = 0$

So for $J = \{0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5\}$ we get the following probabilities of a false positive result $\{3.46 \cdot 10^{-2}, 9.13 \cdot 10^{-3}, 1.81 \cdot 10^{-3}, 3.06 \cdot 10^{-4}, 4.46 \cdot 10^{-5}, 5.58 \cdot 10^{-6}, 5.80 \cdot 10^{-7}, 4.87 \cdot 10^{-8}\}$

In order to compute the probability of getting a false negative, we have to compute the integral of the function from the point t to 1 and then reduce this amount from the area of the rectangle which has been created in the right side of the figure. The integral of the function from point t to 1 is computed as follows:

$$\int_t^1 1 - (1 - j^r)^b \, dj \; = \; F(1) - F(t)$$

And now we have to subtract this result from the area of the rectagle sided in the right part of the graph $Area = 1 \times (1 - t)$

Let FN be the event that the LSH method produces exactly one false negative candidate, $\mathbb{P}(FN) = 1 - t - F(1) + F(t)$

So for $J = \{0.88, 0.9, 0.95, 1\}$ we get the following probabilities of a false negative result $\{2.26 \cdot 10^{-04}, 8.53 \cdot 10^{-06}, 1.14 \cdot 10^{-15}, 0\}$.

Q4: How did you handle the presence of False-Positives in the set of candidate pairs to be Near-Duplicates?

A4: We found two ways to compute and handle FPs.

- The first is the naive one. Just take all the near-duplicates candidates and compute the exact Jaccard similiraty for all of them. We found 42 false positives fixed the treshold to $.88$. Anyway the computational time using this solution is pretty huge becouse the candidate pairs are 42226.

- The second way is a more powerful statistical approach. We can use the properties of the estimator $\hat{J}$ we get from the LSH provided tool. From theory, we know that the obtained estimates are the min-hash collisions relative frequencies, that is fundamentally a $plug-in$ estimator for the probability of interest $\mathbb{P}(m(A) = m(B)) = Jaccard(A, B)$ where $m()$ is the min-hash function and A, B are two lyrics (we assume the LSH pipeline is known).

This estimator, with a deeper obvious look, is nothing more than an average and this is very good becouse this class of estimators is a well known **asymptotic normal and unbiased** class (due to CLT). This means that:

$$\hat{J} \approx N(J, se^2_{\hat{J}})$$

So we can compute **asymptotic normal confidence intervals** for it and perform a Wald test with this hypothesis:

$$H_0 : J \geq 0.88 \quad and \quad H_1 : J < 0.88$$

From theory we know that a 95% CI (so the true value is trapped in it 95 times out of 100) is:

$$\hat{J} \pm 2 * se_{\hat{J}}$$

If, for an estimate $\hat{J}(A, B)$ values smaller than 0.88 are in the CI means that we can reject the null hypothesis.

We don't have the real standard error of the estimator becouse it depends by the true Jaccard value but, due to the huge amount of available data ($\underline{\hat{J}}_n$ is our sample, so the Jaccard estimates) we can get a good approximation by easily compute:

$$\hat{se}_{\hat{J}} = sd(\underline{\hat{J}}_n)$$

Our idea is to compute the true Jaccard **only** for the pairs whoose CIs contain the treshold, so for the pairs whoose true Jaccard might be under that treshold with 95% confidence.

With this criteria, we reduced the pairs to check from 42296 to about 5000!

And all the previous FPs found with the naive approach are in this subset too. This is a very nice result and allowed us to drastically scale down the dimensions. Of course, we obtained a perfect matching but it can happen that the used probabilistic framework misses some pairs (it dependes,of course, by the level of the computed CIs).

Q5: How did you reduce the probability to have False-Negatives?

A5: The only way to reduce the FNs probability is better tune r, b. We can improve the initial optimization approach by plotting the curves of equation $1 - (1 - x^r)^b$ with (r,b) as parameters and picking the values that takes the false negative probability close to zero and false positive probability as low as possible. Once we got the values for *r* and *b*. To do this we attached a simple **R** program.

Q6: The Execution-Time of the Near-Duplicates-Detection tool.

A6: We've run the Near-Duplicates-Detection tool twice, and the execution time is more ore less the following in both cases:

```
real    26m15.286s
user    26m14.664s
```

Q7: The number of Near-Duplicates couples you found.

A7: On both the executions, as said, we've found 42296 duplicates:

```
[MacBook-Air-di-Lorenzo:Desktop ince$ cat test_uno.tsv | wc -l
   42296
[MacBook-Air-di-Lorenzo:Desktop ince$ cat test_due.tsv | wc -l
   42296
MacBook-Air-di-Lorenzo:Desktop ince$
```

Attached Files

- ***part_1_sol.py*** : the implemented sofware to achieve part 1 assignment.
- ***shingler.py*** : the implemented software to achieve the shingling process.
- ***estimate_r_b.R*** : a tiny software that allows to see graphically how the sigmoid is reparametrized for varying **r** and **b**, higligthing the false positive e false negatives areas. Moreover, we used it to numerically validate the obtained FPs and FNs probabilities.
- ***Near_duplicates_found.tsv*** : the set of found near duplicates.
- ***fps_handler.py*** : a tiny software that detect the FPs (using both previous explained ways).
- ***report.html*** : if something is shown wrong in the pdf version.