# Data Mining Technology for Business and Society
# Homework 1

Pavlo Vasylenko (1880258), Eleftheria Tetoula Tsonga (1896559)

April 2021

## Part 1

### Part 1.1

For this part of the homework we have two collections of documents which we had to index and then build a search engine. Using different configurations of search engines we can see which ones perform better than others. We will also analyse the performance by applying different evaluation metrics to our various search engine configurations.

To begin, we used the Cranfield and Time datasets that consist of 1400 and 423 documents accordingly. With them, we have a set of queries for each documents, namely 222 queries for the Cranfield dataset and 83 for the Time dataset. However, what we need to note is that the ground truths that we were given, were not for every query. To be more specific we have the ground truths of 110 queries for the Cranfield dataset (instead of the 222 for the whole query set) and the ground truths of 80 queries for the Time dataset (instead of the 83 for the whole query set).

After getting acquantained with our data , we moved on building the search engine configurations. In the following table we can see the different configurations :

| Search Engine Configuration | Text Analyser | Scoring Function |
|---|---|---|
| SE_1 | Simple Analyser | TF-IDF |
| SE_2 | Simple Analyser | BM25F |
| SE_3 | Standard Analyser | TF-IDF |
| SE_4 | Standard Analyser | BM25F |
| SE_5 | Stemming Analyser | TF-IDF |
| SE_6 | Stemming Analyser | BM25F |
| SE_7 | Stemming Analyser | Frequency |
| SE_8 | Fancy Analyser | TF-IDF |
| SE_9 | Fancy Analyser | BM25F |

We, then evaluated the above configurations with respect to the *MRR* evaluation metric. This metric, in particular, is very useful if we want to assess the quality of a search engine, in which we are interested in a one-result query kind of return. This means that we are interested in seeing any of the relevant documents as soon as possible in the ranking. Calculating this metric for our search engine configurations we ended up with the results seen in the Table 1

| | MRR Scores |
|---|---|
| SE_4 | 0.518066 |
| SE_9 | 0.518054 |
| SE_6 | 0.507879 |
| SE_2 | 0.475891 |
| SE_5 | 0.404933 |
| SE_3 | 0.388130 |
| SE_8 | 0.388130 |
| SE_7 | 0.315803 |
| SE_1 | 0.167498 |

Table 1: Cranfield dataset

| | MRR Scores |
|---|---|
| SE_6 | 0.696300 |
| SE_4 | 0.679269 |
| SE_9 | 0.678393 |
| SE_2 | 0.667941 |
| SE_3 | 0.535944 |
| SE_8 | 0.535944 |
| SE_5 | 0.502219 |
| SE_7 | 0.429372 |
| SE_1 | 0.233927 |

Table 2: Time dataset

| | Mean | | min | | $1^{st}$quartile | | Median | | $3^{rd}$quartile | | max | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cranfield | Time | Cranfield | Time | Cranfield | Time | Cranfield | Time | Cranfield | Time | Cranfield | Time |
| SE_1 | 0.073987 | 0.083677 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.109195 | 0.125000 | 0.833333 | 1.0 |
| SE_2 | 0.245674 | 0.542006 | 0.0 | 0.0 | 0.0 | 0.321429 | 0.250000 | 0.500000 | 0.428571 | 0.888889 | 0.666667 | 1.0 |
| SE_3 | 0.177194 | 0.284970 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.142857 | 0.171429 | 0.285714 | 0.509615 | 1.000000 | 1.0 |
| SE_4 | 0.258218 | 0.547587 | 0.0 | 0.0 | 0.0 | 0.321429 | 0.250000 | 0.500000 | 0.428571 | 0.888889 | 1.000000 | 1.0 |
| SE_5 | 0.171906 | 0.270991 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.142857 | 0.200000 | 0.285714 | 0.500000 | 1.000000 | 1.0 |
| SE_6 | 0.265465 | 0.527260 | 0.0 | 0.0 | 0.0 | 0.191667 | 0.250000 | 0.500000 | 0.428571 | 0.888889 | 1.000000 | 1.0 |
| SE_7 | 0.129988 | 0.215085 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.200000 | 0.448718 | 1.000000 | 1.0 |
| SE_8 | 0.177194 | 0.284970 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.142857 | 0.171429 | 0.285714 | 0.509615 | 1.000000 | 1.0 |
| SE_9 | 0.258218 | 0.548679 | 0.0 | 0.0 | 0.0 | 0.321429 | 0.250000 | 0.500000 | 0.428571 | 0.888889 | 1.000000 | 1.0 |

Table 3: R - Precision distribution table

for the Cranfield dataset and Table 2 for the Time dataset. From these we can choose the **top 5** for each dataset. That is {SE_4, SE_9, SE_6, SE_2, SE_5} for the Cranfield dataset, while {SE_6, SE_4, SE_9, SE_2, SE_3} for the Time dataset. From the computations it is clear that the Search Engine Configurations 2,4,6 and 9 do well for both the datasets. To ensure this we evaluated the search engine configurations, yet again, with the *R - Precision* metric, this time. However, as opposed to the *MRR* which is a comprehensive evaluation metric, the *R - Precision* is computed for each query. Thus, we have demonstrated, in this table the descriptive statistics of the distribution of the *R - Precision* scores evaluated in the query set. What is worth pointing out is that there is an evident difference in the performance of the search engines regarding the dataset. We can see that SE_1 and SE_2 for Cranfield dataset do not even reach the maximum value of 1, meaning we don't get a perfect result for our query, ever. However the biggest difference can be noted when looking at the $3^{rd}$ quartile column, where it is easy to see that the distribution of *R - Precision* with respect to the Time dataset is skewed towards the maximum value 1, while we cannot say the same for the Cranfield dataset.

As an additional test for the performance of our top 5 selected search engine configurations regarding the *MRR*, we will examine their behaviour for two other metrics, namely ***P@k*** and ***nDCG@k***. In Figure 1 and Figure 2, we can see the perfrormance of our top 5 search engine configurations for the ***P@k*** metric, while varying the k. In Figure 3 and Figure 4, we can see the

perfrormance of our top 5 search engine configurations for the *nDCG@k* metric, while varying the k.

By looking at the plots representing the performance of the search engines with the *nDCG* metric we can confidently say that for the Time dataset, the configurations SE_6 and SE_4 (that are actually overlapping in the plot, perform the best in comparison to the other and their performance keep getting better as we increase the positions we take into consideration(k). As for the Cranfield dataset, we can see that SE_4, SE_9 (that are, again, overlapped in the plot) as well as the SE_6 gives us the highest *nDCG* score. An interesting thing to point out, however, is that while SE_4 and SE_9 perform well in the beginning, with a growing k, their performance is deteriorating. That's the opposite of what's happening to the SE_6, which doesn't score high for one-result queries, but with a larger retrieved document window it outperforms SE_4 and SE_9. As a final remark, we need again to point out that the performance of the almost all the search engines we have created perform far better in the Time dataset than the Cranfield dataset, as seen also by the figures.

## Part 1.2

In this part of the homework we are asked to perform quantitative analysis to assess the quality of different Search-Engines for a specific application and select the best one. It is important to note that the application provides in output only four results for each query. In addition to that, the results are shown in random order, so we cannot consider any metric which take into account position (rank) of a result. That means that we are dealing with unranked case described in 8.3 of [1]. The two basic metrics which are used in such occasion are *Precision* and *Recall* [1]. However, after checking our ground truth (GT) data, it is observed that the relevant documents for some queries (their ground truth) are less than the retrieved ones (our 4 randomly shown results). By using *Precision*, the result we would have gotten would be the relevant results divided by 4.

Although it might sound like a good metric, in our case we could never achieve a perfect score ( $= 1$), if the relevant documents of the query are $\leq 4$. An example of why *Precision* might not be a good metric in our situation is the case where our query has 2 relevant documents (its ground truth) and both of them appear in our final 4 results. This should mean that our search engine is working well, since it shows all the relevant documents. When using the *Precision*, however, our results would be $2/4 = 0.5$. Therefore, instead of *Precision* we can use ***P@k*** where $k = 4$. With this kind of tool, in the previous example, we would get the maximum value of 1, since the relevant documents that are retrieved are divided with the $min(k, |GT(q)|)$.

It is important to point out that ***P@4***, when the relevant documents of a query are $\leq 4$, gives us recall. So since we have ***P@4*** and recall (in one) for the

queries with $|GT(q)| \leq 4$, we also consider computing the recall for the cases where the queries have $|GT(q)| > 4$. After computing this two metrics, for the two cases we can combine them into computing the F1 score for each Search Engine. In Table 3, we can see the values of ***P@4***, recall and the F1 score for each candidate Search Engine. It is evident that the F1 for the first search engine has the largest value, making it the best choice according to our analysis.

| SE | P@4 | Recall | F1 |
|---|---|---|---|
| 1 | 0.347643 | 0.267531 | **0.302370** |
| 2 | 0.252946 | 0.193593 | 0.219325 |
| 3 | 0.343855 | 0.266398 | 0.300211 |

Table 4: P@4, Recall and F1 measure for given search engines

# Part 2

## Part 2.1

In this part of the homework we need to find, in an approximated way, all near-duplicate documents inside a dataset. That was done by shingling the documents, sketching then in a Min-Hashing sketch with a length of at most 300 and finally parsing them through our LSH tool.

For the Shigling, we had to represent the songs with singhles of maximum length 3. This was done by, firstly, removing punctuation and converting everything in lower-case via *whoosh.analysis.SimpleAnalyzer*. After that, we created the shingles for each document while building the universal set of shingles. After building the universal set of shingles, we associated an integer, starting from 0, for each shingle as an identifier and we replaced the set of singles of each document with their corresponding set of identifiers.

Then we used the tools provided to us to build the Min-Hashing sketch but also to perform LSH. For these last steps we had to take into consideration that the number of hashing functions should be at most 300 and that The probability to have as a near-duplicate candidate a pair of documents with Jaccard=0.95 must be ¿ 0.97. Keeping that in mind, we had to come up with a way to choose the best number of rows and the number of bands.

As a first step in our analysis, we need to choose the number of rows and the number of bands for our Min-Hashing sketch. For this task we need to take into consideration that our sketch need to be of length up to 300. So we focus on, first, choosing the parameter b (the number of bands). In order to calculate the minimum $b$ for which the probability to have as a near-duplicate candidate a pair of documents with Jaccard=0.95 greater than 0.97, we first solved the equation $1 - (1 - 0.95^{\frac{300}{b}})^b - 0.97 = 0$ via *scipy.optimize.root_scalar* which

gave us solution of 11.5094 Figure 5. By going right (positive sign of function) from the root, we could find the closest integer that divides 300. Thus, 12 is our minimum $b$ that we can choose. Then our choice for $r$ is obvious since $r \times b = 300$.

The next step to ensure that our tool works well is to find the False Negatives created by the approximation. We know that the probability of getting False Negatives is $(1 - j^r)^b$. Therefore, to reduce this probability, we can increase the value of $b$. In our work, we set $b$ to be 30 which gives us a probability of $1.27 * 10^{-12}$. That means that approximately 1 pair out of $10^{12}$ 95%-similar pairs is wrongly excluded from the bucket. It might be too extreme to consider such small probability since we have an overall number of pairs $250000^2 \approx 6 * 10^{10}$, however we do not know in advance how many similar pairs are there, so this makes the occurrences of false negatives almost impossible even if there are all similar pairs. However, it increases the numbers of false positives.

Following the reduction of False Negatives, we need to take into consideration also the reduction of the False Positives. The good news is that after performing LSH, the tool re-use the Min-Hashing sketches to calculate the approximate Jaccard similarities, so after this computation the set of suggested near-duplicates become smaller. $[Lab\_22 : 54]$ After using the NearDuplicate tool, we found that there were 35040 candidate pairs. Finally we calculated real Jaccard similarity for each candidate pair, and we ended up with 34992 pairs that satisfied $j \geq 0.95$. It can be noted that the time of NearDuplicate tool took 47.6 sec (Apple Mac Book Pro with M1 chip).

## Part 2.2

In this part of the homework we need to do a similar task as in Part 2.1. However, now we are asked to create a set of shingles that are not of the documents themselves but of their title. In addition to that, we also had to take into consideration that the Jaccard similarity for this task was 1.

Having the Jaccard similarity equal to 1 means that we are searching for an exact match of the set of singles. That, also, means that our relation is is transitive so if $a = b$ and $b = c$ then $a = c$.

Taking that into account we can simple apply hashing to the sets of singles and put the same-hashing elements into one bucket. By using $frozenset$, Python makes our sets hashable by default so we can use them as keys in dictionary of lists (here we used $defaultdict$). In such way, we can just add a new element into the list, which is located in the dictionary with key,the specific set of shingles. After doing that for each song, we ended up having a dictionary of lists which is, then, traversed. Then, for each list in the dictionary that has size greater than 1, we compute pairs by applying $itertools.combination(list, 2)$.

In this method there is no False Positives and False Negatives since we do

not perform any approximation.

Time with shingles: 23.7 sec. Time algorithm only: 11.37 (Apple Mac Book Pro with M1 chip)

Number of pairs found: 435514

# Figures

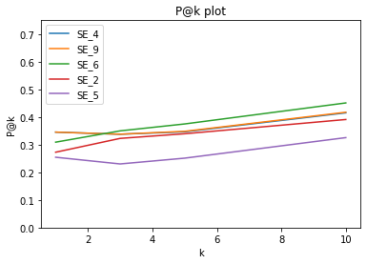


Figure 1: Precision at k for the Cranfield dataset

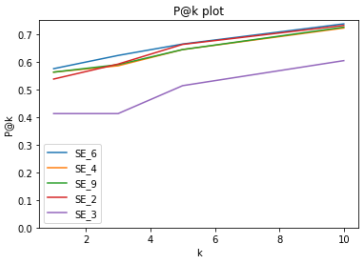


Figure 2: Precision at k for the Time dataset

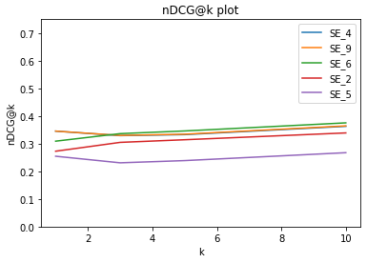


Figure 3: nDCG at k for the Cranfield dataset

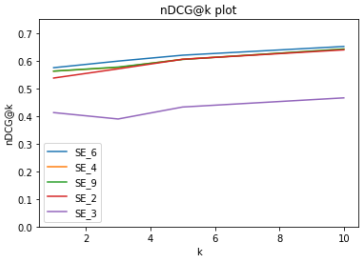


Figure 4: nDCG at k for the Time dataset

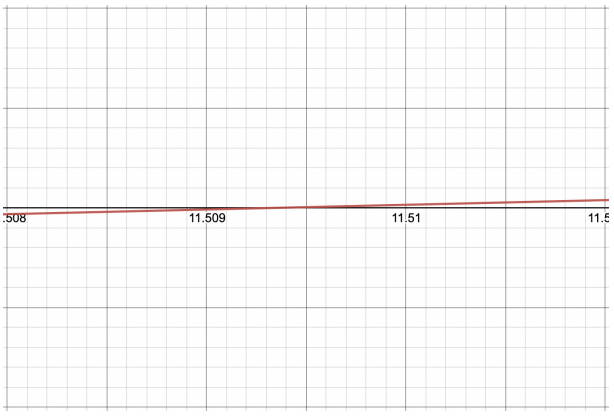


Figure 5: Root of $1-(1-0.95^{\frac{300}{b}})^b-0.97=0$

# References

[1] IIR book