DMT HW1: Search Engine and Near Duplicates Detections

Alessandro Quattrociochi - 1609286
Tansel Simsek - 1942297

Part 1.1

CRANFIELD DATASET

- There are 1400 number of indexed documents and 222 number of queries.
- There are 110 number of queries in the Ground-Truth.

SE configuration name	Analyzer	Scoring function
$SE1$	$SimpleAnalyzer()$	$scoring.TF\_IDF()$
$SE2$	$SimpleAnalyzer()$	$scoring.BM25F(B = 0.75, content\_B = 1.0, K1 = 1.5)$
$SE3$	$StandardAnalyzer()\|StopFilter()$	$scoring.TF\_IDF()$
$SE4$	$StandardAnalyzer()\|StopFilter()$	$scoring.BM25F(B = 0.75, content\_B = 1.0, K1 = 1.5)$
$SE5$	$StemmingAnalyzer()$	$scoring.TF\_IDF()$
$SE6$	$StemmingAnalyzer()$	$scoring.BM25F(B = 0.75, content\_B = 1.0, K1 = 1.5$
$SE7$	$StandardAnalyzer()$	$scoring.TF\_IDF()$
$SE8$	$FancyAnalyzer()$	$scoring.TF\_IDF()$
$SE9$	$FancyAnalyzer()$	$scoring.BM25F(B = 0.75, content\_B = 1.0, K1 = 1.5$
$SE10$	$FancyAnalyzer()$	$scoring.Frequency()$

Table 1. Tested Search Engine Configurations

SE configuration	MRR
$SE1$	0.16846277
$SE2$	0.49607558
$SE3$	0.3881298
$SE4$	0.50491926
$SE5$	0.4049334
$SE6$	0.49756892
$SE7$	0.3881298
$SE8$	0.3881298
$SE9$	0.50486719
$SE10$	0.31214

Table 2. MRR

- The set of all Top-5 search engine configurations according to the **Table 2** are *SE4, SE9, SE6, SE2* and *SE5*.

SE configuration	Mean	Min	$1st$ Quartile	Median	$3rd$ Quartile	Max
1	0.07399	0.0	0.0	0.0	0.1092	0.83333
2	0.24977	0.0	0.0	0.25	0.39375	1.0
3	0.17719	0.0	0.0	0.14286	0.28571	1.0
4	0.25858	0.0	0.0	0.25	0.4	1.0
5	0.17191	0.0	0.0	0.14286	0.28571	1.0
6	0.25969	0.0	0.0	0.25	0.44048	1.0
7	0.17719	0.0	0.0	0.14286	0.28571	1.0
8	0.17719	0.0	0.0	0.14286	0.28571	1.0
9	0.26017	0.0	0.0	0.25	0.4	1.0
10	0.12781	0.0	0.0	0.0	0.2	1.0

Table 3. R Precision Distribution

- In consequence of the **Figure 2**, SE6 has found as the best search engine among its other competitors. It is important to state that this decision has been made for the given k values. If the k restricted until 3, SE9 and SE4 have slightly better results than SE6. However, when k gets larger, on average SE6 performs continuous increase. As a result, if SE has only 10 documents of space for output, SE6 gives more relevant outcome than the others.

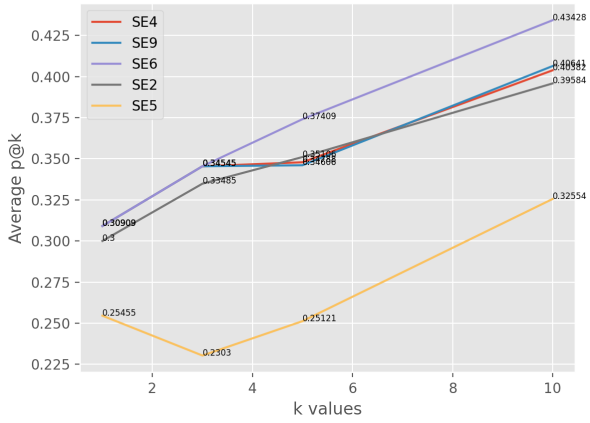

Figure 1. P@k plot with data from the Top-5 search engine configurations according to the **Table 2**.

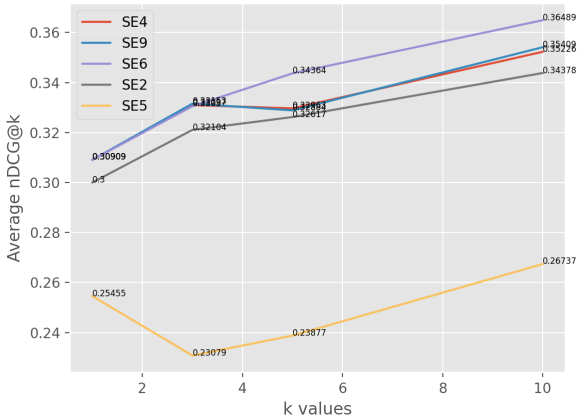

Figure 2. nDCG@k plot with data from the Top-5 search engine configurations according to the **Table 2**.

TIME DATASET

- There are 423 number of indexed documents and 83 number of queries.
- There are 80 number of queries in the Ground-Truth.
- The same configurations are tested for both datasets. Thus, **Table 1** can be taken to see Search Engine Configurations for this dataset.
- **Table 4** is created to show the MRR values for all tested search engine configurations.

SE configuration	MRR
$SE1$	0.24276197
$SE2$	0.65358297
$SE3$	0.53644418
$SE4$	0.65510556
$SE5$	0.50287687
$SE6$	0.70387899
$SE7$	0.53644418
$SE8$	0.53644418
$SE9$	0.65835231
$SE10$	0.45762004

Table 4. MRR

- The set of all Top-5 search engine configurations according to the **Table 2** are *SE6, SE9, SE4, SE2* and *SE8*.

SE configuration	Mean	Min	1*st* Quartile	Median	3*rd* Quartile	Max
1	0.08368	0.0	0.0	0.0	0.125	1.0
2	0.53312	0.0	0.2	0.5	0.88889	1.0
3	0.28497	0.0	0.0	0.17143	0.50962	1.0
4	0.52316	0.0	0.18571	0.5	0.88889	1.0
5	0.27099	0.0	0.0	0.2	0.5	1.0
6	0.54252	0.0	0.2	0.51667	0.88889	1.0
7	0.28497	0.0	0.0	0.17143	0.50962	1.0
8	0.28497	0.0	0.0	0.17143	0.50962	1.0
9	0.52941	0.0	0.2	0.5	0.88889	1.0
10	0.23556	0.0	0.0	0.0	0.5	1.0

Table 5. R Precision Distribution

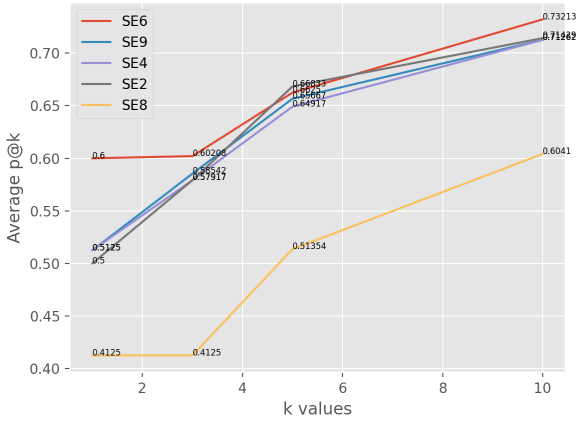

Figure 3. P@k plot with data from the Top-5 search engine configurations according to the **Table 4**.

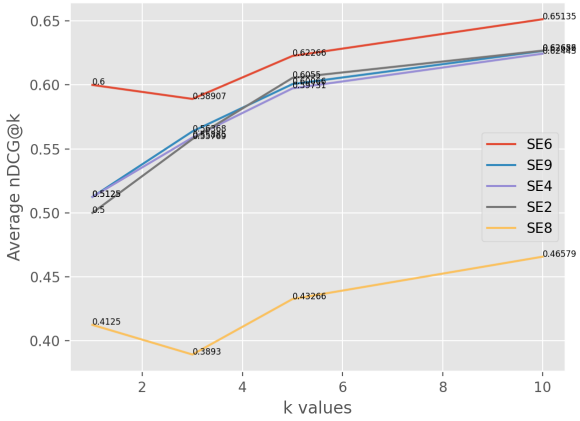

Figure 4. nDCG@k plot with data from the Top-5 search engine configurations according to the **Table 4**.

- Accordingly only the **Figure 4**, SE6 gives the best result than any configuration in Top-5 search engines. It can be seen from the graph that for the given k values, SE6 is well ahead from its competitors. Moreover, under investigation of curve shape shows that for the given k values SE6 performs neither increasing nor decreasing behaviour. It can be said that on average SE6 finds better match for k 1 than for k 3. For the following k values, curve shows an increase which can be stated as possibility of matching with relevant documents has higher change when k gets larger.

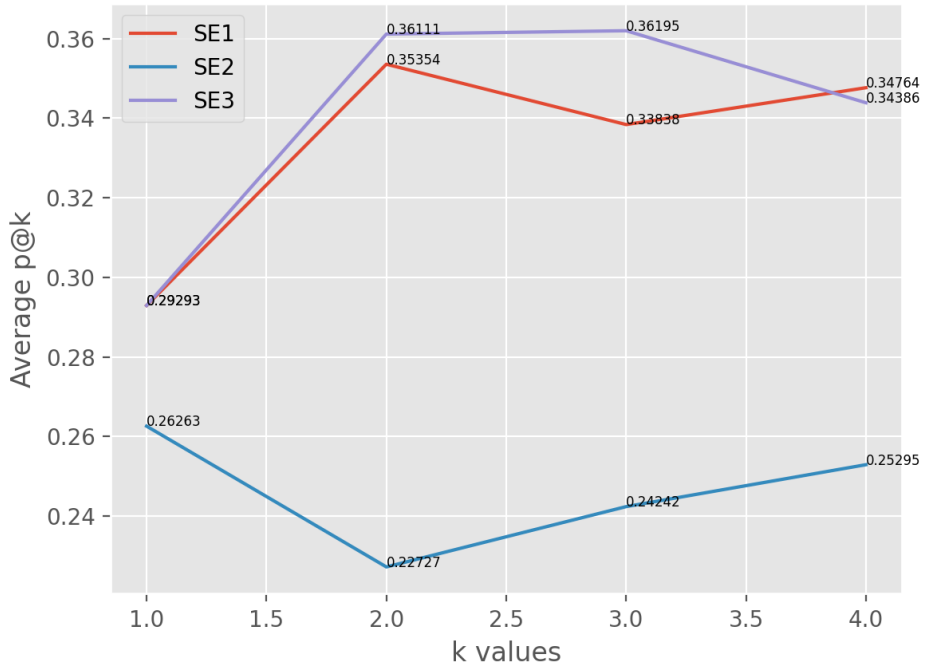

Figure 5. p@k plot for *SE1, SE2* and *SE3*.

First of all, it is more convenient to understand the setting in order to choose the most representative metric. There are 3 different search engines and one of them has to be selected to provide the following needs. The maximum number of result to be displayed is 4. More importantly, the results will be given in random order. Thus, k has to be considered and taken as 4 at maximum. Therefore, Mean Reciprocal Rank (MRR) metric is eliminated from the measurement since it is main concern is first relevant result. R precision is not taken into account because it examines based on the ground truth length. Normalised Discounted Cumulative Gain metric is not useful for the given setting because this metric cares the order. For instance, for the same query when k equals 4, SE_A has [1,0,1,1] result, while SE_B has [1,1,0,1] result. In our setting, they are equally fine because the result will be provided in random order. However, when we calculate nDCG@k = 4, SE_B will have higher value than SE_A, just because SE_B matched relevant result in lower number of position than SE_A. Hence, the selected measurement shouldn't care the order. In this way, Precision@k metric has been chosen because for the same example given for nDCG@k, both SE_A and SE_B will return the same result which is desired for this setting. The above curve is fitted, but it is important to understand that just examination for k equals 4 is sufficient. As comparison between 3 SE's, SE2 gives very unsatisfactory result among the others. SE1 and SE3 gives pretty close results. But, SE1 is decided as the best one because it has slightly better value than SE3.

Part 2.1

In this second part we will describe the Near Duplicates Detection pipeline used on the dataset "250K_lyrics_from_MetroLyrics.csv", analysing its operation and the tools used estimate the near duplicates songs present in the dataset.
The entire pipeline consists of four main blocks:

- **Shingling**
- **Min-Hashing**
- **LHS- Locality Sensitive Hashing**
- **Approximate Similarity Computation**

For the shingling part we implemented the code ourselves, while for the Min Hashing and LHS part we used the available tools, in particular the one which includes Min-Hashing function and LHS.

Shingling

Before proceeding with the creation of the shingles we removed the punctuation and applied a lower case filter to the entire dataset. After this preliminary step we created a function that, given the dataframe's columns for the name and lyrics of the songs as input variable, adds to it a new column containing an integer number that uniquely identifies a shingles. This step is fundamental and will be used in the same in the last part of the homework.In our case, the size of the number of shingles is set to $w = 3$.
The following example shows the final result after applying the algorithm we implemented.

$$\boxed{(ilovemydog) \rightarrow (i, love, my), (love, my, dog)}$$

MinHashing and LHS tools and Approximate Similarity

These three blocks represent the core of the entire pipeline, in fact are used to compute the Jaccard similarity between songs, considering title and lyrics and find a near duplicates. For this purpose, we used one tool of the four provided named *"NearDuplicatesDetectorMinWiseHashingWithLSH.class"*, in which the entire estimation process was employed. It is a java tool that we launched from the command shell, given some input parameters as the number of GB of memory allocated, the type of algorithm chosen (in our case *"lsh plus min hashing"*) the Jaccard threshold (0.95), the number of rows and bands (13,10) and the number of hash functions ($130 = 13 \cdot 10$) generated given by the product of $n \cdot b$.
The code used for running is as follows:

```
java -Xmx3G tools.NearDuplicatesDetector lsh_plus_min_hashing 0.95 13
   10 ./hash_functions/130.tsv./dataset/250K_LYRICS.tsv ./dataset/
   dataset_LSH__13_10_near_duplicates_output.tsv
```

Probability Measure and Parameter Selection

The choices of the number of rows, bands are made accordingly with the two initial constrains:

- Min-Hashing sketch with a length of at most $300 -> n = r \cdot p \leq 300$
- Probability of a near-duplicate candidate a pair ≥ 0.97 with Jaccard Threshold $= 0.95$

To meet these conditions we looked for a numerical optimization generating with *Numpy* package two vectors **r** and **b** of integers numbers in the range [0,300] and performed all the combinations whose product was less than 300. Finally, after calculating the probability according to the expression below, we selected only those pairs that guaranteed $p > 0.97$ with $J = 0.95$ fixed.
The model used to calculate probabilities is as follows:

$$\boxed{p = 1 - (1 - j^r)^b} \tag{1}$$

Several pairs (r,b) met that condition, but the pair with rows and bands equal to **(13,10)** was chosen. In the next section we will justify why this particular pair was chosen in accordance with the probability plot.

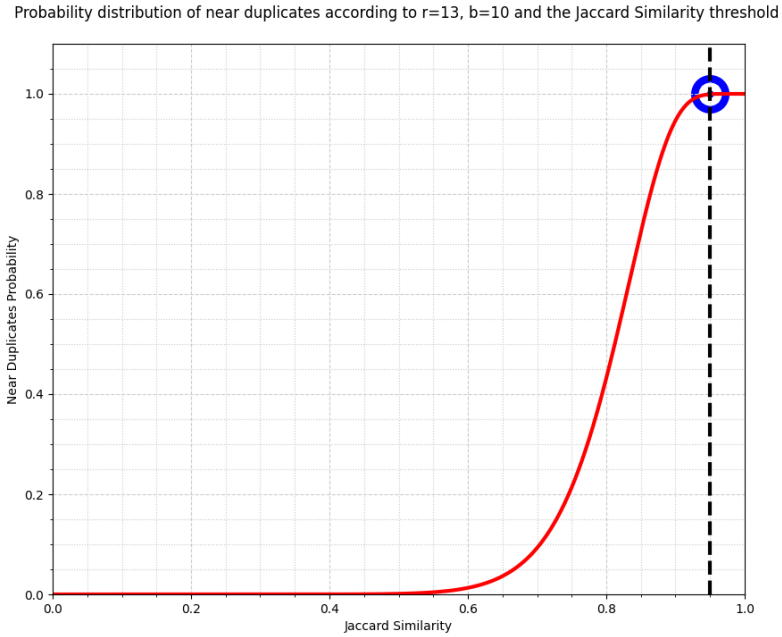

Figure 6. Probability of near duplicates songs given Jaccard threshold

False Negatives Reduction

A false negative is a set of documents with a real Jaccard similarity upper than the threshold but our pipeline improperly provide in output. The presence of false positives is due to the fact that the probability of finding near duplicates, when the curve crosses the Jaccard threshold, is not yet unitary. This generates in output some wrong sets. We compensated for the number of false negatives by choosing a probability value very close to 1 at the point where the intersection with the line given by the Jaccard similarity threshold occurs.

False Positive Reduction

A false positive in our case is a set of documents with a real Jaccard similarity lower than the threshold but our pipeline improperly provide it in output. The number of false positives can be reduced by increasing the selectivity of the S-curve and thus the number of rows. An ill-advised decrease in the selectivity of the curve, however, would increase the number of false negatives. Looking at the curve from a single point of view of the number of false positives, we can push the curve to the y-axis, on this parameter we can only act a posteriori.

Time Execution of Near-Duplicates-Detection Tool

The entire process performed by the tool took 2 minutes and 30 seconds, including the time needed to create the sketches.

Number of Near-Duplicates couples

The estimated number of near duplicates find in the dataset, in accordance with the parameters chosen, is 34266, about 13.71% of the number of songs in the entire dataset.

Part 2.2

Also in this part we identified the number of near duplicates but, this time, using a different approach. In fact, instead of using the java tools of the previous section that provided an estimate we attempted an exact resolution approach, although this road presents several problems.

Dataset Preprocessing and Shingling

We have analyzed the same dataset as before but this time taking only into account the songs name column and we performed a slightly different preprocessing replacing the character "-" with a blank space, removing the punctuation and applying a lower case filter. After that we ran the same algorithm to generate the shingles but this time having a lower number of words compared to the previous case (in which we also took into account the lyrics of the songs in addition to the title) we added the condition that if the number of words was less than 3, we created a single shingle with all the content of the title

Algorithm description

The algorithm works in the following way: for each song represented by its own set of shingles each represented by an integer from 0 to the number of shingles generated minus one, we iterated over the whole dataset calculating the Jaccard similarity for each pair and if this value was equal to 1, then it was considered as duplicate. For cases where the words contained in the title were equal to one and then a single shingle was generated we used the following method. Have Jaccard similarity equal to 1 means that the numerator and denominator are equal, so in that case the duplicates were found by returning all the titles represented by the same identifier of the shingles.
The Jaccard similarity definition used in this one is as shown below:

$$\boxed{Jaccard(A, B) = \frac{A \cap B}{A \cup B}} \tag{2}$$

False Negatives and False Positive Reduction

In this case, unlike the previous one, we are not estimating the approximate value of the Jaccard similarity but we are calculating it, for each of all possible pairs of song titles in the whole dataset, so we do not have false positives and false negatives.

Time Execution of Near-Duplicates-Detection Tool

Calculating the exact value of the similarity between the songs implies to iterate on all the n rows of the dataset and for each row of it to calculate J. This aspect quickly highlight its own limitations and how the computational aspect could not be neglected. So, considering the number of possible permutations that can taking two documents from n, we can exploit the definition of binomial coefficient and find the time complexity needed to solve the following algorithm.

$$\text{Time complexity} = \binom{N}{2} = \frac{1}{2}n(n-1) \sim \mathcal{O}(n^2) \tag{3}$$

As a result, the total execution time of the method took approximately 4 hours and 52 minutes.

Number of Near-Duplicates couples

It has been found that there are 443875 number of near duplicates couples.