DMT - Homework 1

Tran Luong Bang, Juan Mata Naranjo
Master in Data Science

April 14, 2021

1 Part 1.1

Before we deep dive into each of the individual datasets and their respective results we will start out by first presenting the number of documents we have, the number of queries applied over these documents and finally the number of queries for which we have ground truth information at our disposal. When computing the evaluation metrics we will assume that the queries for which we don't have ground truth don't exist (we remove them since we cannot compare them against anything).

	Cranfield Dataset	Time Dataset
Num Indexed Docs	1400	423
Num Queries	225	83
Num Queries in GT	110	80

Table 1: Overview Table

1.1 Cranfield Dataset:

The configurations we have tested are the following:

Conf. ID	**Text Analyzer**	**Scoring Functions**
1	RegexTokenizer() | LowercaseFilter() | StopFilter(stoplist = STOP_WORDS)	scoring.MultiWeighting(scoring.Frequency(), title=scoring.BM25F(), content=scoring.TF_IDF())
2	RegexTokenizer()| StopFilter(stoplist = STOP_WORDS)| LowercaseFilter() | StemFilter()	scoring.TF_IDF()
3	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.TF_IDF()
4	FancyAnalyzer()	scoring.BM25F(K1=2, B=.8)
5	SimpleAnalyzer()	scoring.TF_IDF()
6	StandardAnalyzer()	scoring.Frequency()
7	FancyAnalyzer()	scoring.MultiWeighting(scoring.Frequency(), title=scoring.TF_IDF(), content=scoring.BM25F())
8	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.BM25F()
9	RegexTokenizer()| StopFilter(stoplist = STOP_WORDS)| LowercaseFilter() | StemFilter()	scoring.MultiWeighting(scoring.Frequency(), title=scoring.TF_IDF(), content=scoring.BM25F(K1=2, B=.8))
10	SimpleAnalyzer()	scoring.BM25F(K1=1.2, B=.75)
11	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.TF_IDF()
12	FancyAnalyzer()	scoring.TF_IDF()

Table 2: Configuration Overview for Cranfield Dataset

For which the resulting MRR and R-Precision results are the following (the table presented below is already sorted from highest to lowest MRR values):

Conf. ID	MRR	Mean	Min	1st Quartile	Median	3rd Quartile	Max
8	0.458	0.237	0.000	0.000	0.186	0.389	1.000
9	0.453	0.230	0.000	0.000	0.167	0.379	1.000
10	0.417	0.193	0.000	0.000	0.143	0.282	1.000
4	0.415	0.216	0.000	0.000	0.186	0.333	1.000
7	0.412	0.212	0.000	0.000	0.170	0.333	1.000
2	0.368	0.165	0.000	0.000	0.106	0.250	1.000
3	0.368	0.165	0.000	0.000	0.106	0.250	1.000
11	0.368	0.165	0.000	0.000	0.106	0.250	1.000
1	0.323	0.147	0.000	0.000	0.025	0.250	1.000
12	0.311	0.137	0.000	0.000	0.000	0.222	1.000
6	0.248	0.104	0.000	0.000	0.000	0.167	0.833
5	0.126	0.056	0.000	0.000	0.000	0.000	0.667

Table 3: MRR and R-Precision Overview for Cranfield

Finally, the next figure shows the results for the NCDG@k and P@k metrics for $k \in [1, 3, 5, 10]$

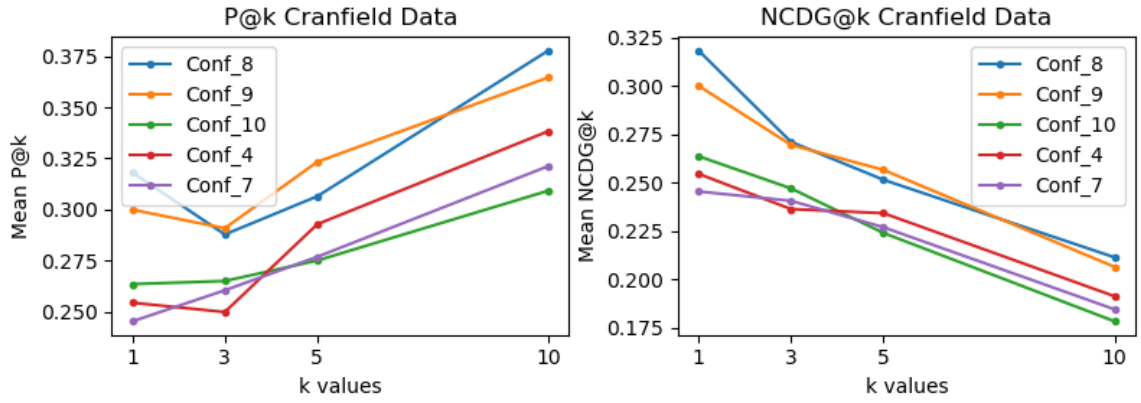


Figure 1: P@k and NCDG@k Plot Results for Cranfield Dataset

If we were to choose the best Search Engine solely based on the NCDG@k curve we would choose the configuration for which this ratio is highest when k is maximal. Therefore, based on the previous plot, the champion configuration would be **Conf 8** (Stemming Analyzer & BM25F).

1.2 Time Dataset:

We will follow the same structure and order to present the Time Dataset results:

Conf. ID	Text Analyzer	Scoring Functions
1	RegexTokenizer() | LowercaseFilter() | StopFilter(stoplist = STOP_WORDS)	scoring.TF_IDF()
2	RegexTokenizer()| StopFilter(stoplist = STOP_WORDS) | LowercaseFilter() | StemFilter()	scoring.BM25F(K1=1.2, B=.75)
3	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.TF_IDF()
4	FancyAnalyzer()	scoring.BM25F(K1=2, B=.8)
5	SimpleAnalyzer()	scoring.TF_IDF()
6	StandardAnalyzer()	scoring.Frequency()
7	FancyAnalyzer()	scoring.BM25F()
8	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.BM25F()
9	RegexTokenizer()| StopFilter(stoplist = STOP_WORDS) | LowercaseFilter() | StemFilter()	scoring.BM25F(K1=2, B=.8)
10	SimpleAnalyzer()	scoring.BM25F(K1=1.2, B=.75)
11	StemmingAnalyzer(stoplist=STOP_WORDS)	scoring.BM25F(K1=2, B=.8)
12	FancyAnalyzer()	scoring.TF_IDF()

Table 4: Configuration Overview for Time Dataset

Conf. ID	MRR	Mean	Min	1st Quartile	Median	3rd Quartile	Max
11	0.725	0.555	0.000	0.264	0.539	0.906	1.000
8	0.717	0.562	0.000	0.200	0.564	1.000	1.000
2	0.708	0.557	0.000	0.200	0.564	1.000	1.000
9	0.703	0.550	0.000	0.200	0.539	0.906	1.000
4	0.689	0.539	0.000	0.200	0.500	0.889	1.000
7	0.678	0.549	0.000	0.321	0.500	0.889	1.000
10	0.669	0.542	0.000	0.321	0.500	0.889	1.000
1	0.590	0.351	0.000	0.000	0.376	0.600	1.000
3	0.566	0.309	0.000	0.000	0.333	0.500	1.000
12	0.536	0.285	0.000	0.000	0.171	0.510	1.000
6	0.457	0.238	0.000	0.000	0.063	0.500	1.000
5	0.240	0.084	0.000	0.000	0.000	0.125	1.000

Table 5: MRR and R-Precision Overview for Time

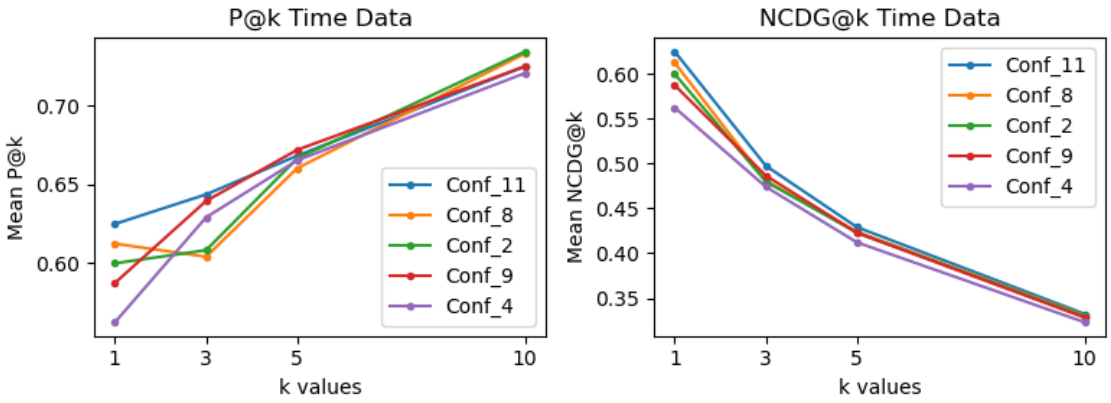


Figure 2: P@k and NCDG@k Plot Results for Time Dataset

2 Part 2.1

The first thing we have to do before running the LSH algorithm is to figure out the best r (number of rows in each band), b (number of bands) and n (number of hash functions used) parameters such that we can minimize the number of False Positives and False Negatives. In particular, the choice of parameters will mainly help us to minimize the quantity of False Negatives. Taking inot account the other constraints which are the following:

$$r \cdot b = n \tag{1}$$

$$0.97 > 1 - (1 - 0.95^r)^b \tag{2}$$

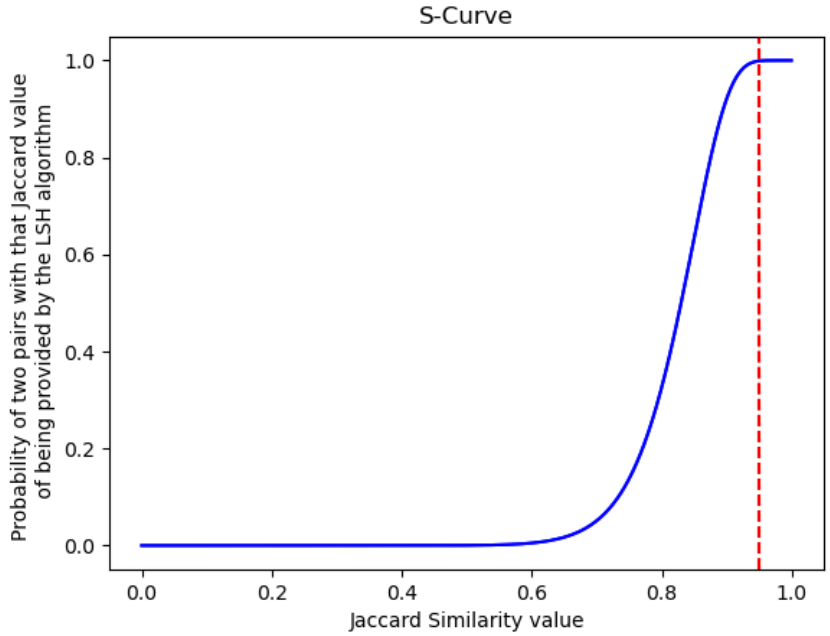
From the second constraint we can see that the following relation holds:

$$b > \frac{log(1 - 0.97)}{log(1 - 0.95^r)} \tag{3}$$

In order to minimze the False Negatives we will stress the 0.97 boud as much as possible, specifically up to 0.999. This will reduce the number of False Negatives, but will of course increase the number of False Positives. Since we will be able to remove some of the False Positives later on we will not worry too much for the moment. With this assumption, fixing a value of $r = 15$ we have decided to use the following parameters:

r	15
b	12
n	180

Table 6: Parameter Overview



After running the LSH algorithm with the previous set of parameters we got the following number of *potential* near duplicate matches: **34655**. However, as we already mentioned before, we can still reduce the number of False Positives a posteriori by computing the Jaccard Similarity over the set of pairs provided by the LSH tool, and therefore eliminating those pairs which we will not consider as near duplicates. For this purpose we have used the code named "False Positive Reduction" (in which we re-compute the real Jaccard similarity and not the approximated one), after which the number of *final* near duplicates was: **33868**. Of course, in this specific case we could reduce the number of False Negatives, making the LSH algorithm return more potential pairs (approx. 13%). This might be too many potential pairs in other applications where instead of 250K documents we have billion documents. In these cases the percentage of potential pairs in output must be much lower, and therefore we might have to deal with having more False Negatives.

The total time required to run the LSH tool with the parameters highlighted previously was of **7 minutes and 15 seconds** on our machine.