DMT – Homework1

Vigèr Durand Azimedem Tsafack(1792126)
Malick Alexandre N. Sarr (1788832)

✓ **First Step**: Search-Engine Evaluation

1. Assessment of 3 search engines using: P@K, R-Precision, MRR and nDCG

(Python code: *HW_1_part_1_1.py*, custom module: *hwmodule.py*)

Statistics on the Dataset:

Dataset	Number of Documents	Number of Queries
GT	728	222
SE_1	1099	222
SE_2	1398	222
SE_3	1395	222

P@k table:

Search Engine	Mean(P@1)	Mean(P@3)	Mean(P@5)	Mean(P@10)
SE_1	0.031532	0.03003	0.027928	0.025676
SE_2	0.301802	0.295796	0.263063	0.185586
SE_3	0.238739	0.205706	0.186486	0.143243

R-Precision table:

Search Engine	Mean (R-P_Distr.)	Min (R-P_Distr.)	1°_quart. (R-P_Distr.)	Median (R-P_Distr.)	3°_quart. (R-P_Distr.)	MAX (R-P_Distr.)
SE_1	0.022563	0	0	0	0	0.666667
SE_2	0.254943	0	0	0.25	0.428571	1
SE_3	0.179328	0	0	0.142857	0.333333	1

MRR table:

Search Engine	MRR
SE_1	0.081414
SE_2	0.486303
SE_3	0.395118

nDCG table:

Search Engine	Mean (nDCG@1)	Mean (nDCG@3)	Mean (nDCG@5)	*Mean (nDCG@10)*
SE_1	0.025521	0.061579	0.085546	*0.126786*
SE_2	0.195554	0.526353	0.672991	*0.80411*
SE_3	0.158024	0.388966	0.487971	*0.613001*

2. Helping the "NoobDataScience" Company to choose between 3 different SE's

(Python code: *HW_1_part_1_2.py*, custom module: *hwmodule.py*)

The data we have here show that we are dealing with ranking algorithms. Each query retrieves 200 documents ranked in order of relevance. This simple analysis makes us preliminary understand that Precision, Recall and even their harmonic mean (F-measure) are not good metrics for picking the best search engine here.

However, since the "NoobDataScience" app will return to the user only four of the 200 query results, we will go forward assuming that the app picks the first four results returned by the ranking algorithm and show them to the user in a casual order.

The previous assumption changes everything. In fact, considering only the first query, the three search engines will return the following results:

```
In [38]: set(se1_part1_2[1][:4])
Out[38]: {13, 51, 184, 486}

In [39]: set(se2_part1_2[1][:4])
Out[39]: {13, 202, 486, 875}

In [40]: set(se3_part1_2[1][:4])
Out[40]: {13, 51, 184, 486}
```

In this particular case (case where the app returns four unranked results), we are dealing with a "normal" retrieval system without ranking. Thus, measures as P@K, MAP, MRR, are not useful since they take into account the rank.

The metric suiting the most our purpose here is the *mean(F-measure)*. We are not simply using *mean(P)* cause we want to take into account not only the precision but also the recall.

F-measure table:

Search Engine	Mean (F_measure)
SE_1	0.247949
SE_2	0.177724
SE_3	0.253298

This F-measure's table tell us that the "NoobDataScience" company should choose the SE 3 since it has the greatest F-measure.

✓ **Second Step**: Near-Duplicates-Detection

1. Finding all near-duplicate documents inside a dataset

(Python code: *HW_1_part_1_2.py***)**

In this part, we were tasked to find in an approximated way, all near-duplicate documents present in a document set made of .html file containing lyrics. We were supposed to use as a metric for document similarity the Jaccard similarity between their associated sets of shingle (which should be above .85). Attached is the output file containing the results.

The document set contained 87046 files. When we imported the files, we lost 1470 documents that were considered as error due mostly to incompatible characters. Ergo we were left with a data-set of 85576.

We then created shingles from the document and recorded a total 20000000 shingles for all the documents and 6962837 Unique Set of shingles with no duplicate shingle. Each shingle was hashed into a unique 32bit number using the crc32 from Binascii package.

We then went on generating the 300 hash functions for the sketching process using as upper bound on number of distinct terms the total number of unique shingles in our document (6962837). We generated the 300 function which we used to convert shingles into hash numbers by passing each shingle for each document through our hash files parameters using the formula $\boldsymbol{a * shingle + b \% p}$ and keeping the min value as shingle id. We then generated the input file from the hashed shingles to pass through the tools.

Finally, we passed the input file through the Java tools provided using a **band value** $\boldsymbol{b = 23}$ and the **number of rows** $\boldsymbol{r = 13}$ (the index n of our tools was 299, and $\boldsymbol{n = b * r}$. Also, 23 and 13 are the only number when multiplied give 299).

Mathematically, that means that there is $\boldsymbol{0.85^{13} = 0.12(12\%)}$ chance that our candidate pair are similar in a band $\boldsymbol{b}$. Ergo we have a probability $\boldsymbol{p = 1 - 0.12^{23} = 0.053(53\%)}$ chance that the ***"85% similar"*** column pairs are false negative. Meaning that in our result set $\boldsymbol{94.5\%}$ of pairs are truly similar documents.

The number of Near duplicate we found is $\boldsymbol{1541}$ having a Jaccard similarity above $\boldsymbol{0.85}$ after *LSH*. **The number of Near Duplicate Candidate we found is** $\boldsymbol{1644}$. **The number of false positive identified is** $\boldsymbol{103}$. Which correspond to our mathematical expectation of around $\boldsymbol{5 - 6\%}$ of false-positives.

2. "Set-Size-Estimation problem" and "Unions-Size-Estimation problem"

a. "Set-Size-Estimation problem" (Python code: HW_1_part_2_2_a.py, custom module: hwmodule.py)

We assume here that the sketching lists have been generated using the ***k-min*** approach (Cohen 1994).

- In this case, a sketch ***S(A)*** is a sequence of exactly ***k*** entries, $a_1, \dots a_k$.
- Let $r_1, r_2, \dots, r_k$ be normalized ranks r_i: $U \rightarrow \left\{\frac{1}{n}, \frac{2}{n}, \frac{3}{n}, \dots, 1\right\}$.

The *k-min* approach provides a set of operations among which the $SIZE(S(A))$, which estimates the number of distinct elements of A. The analysis performed in section 6 of the paper *[Estimating the Size of the Transitive Closure in Linear Time BY Edith Cohen] establishes that the estimator:*

$$\hat{S} = \frac{k}{\sum_{a_i \in S(A)} r(a_i)} - 1$$

Estimates $|A|$ with a good level of confidence. This is the estimator that we are going to implement here to estimate the original set sizes given the min-hashing sketches and the universe size.

A *piece of output file:*

Min_Hash_Sketch_INTEGER_Id	ESTIMATED_ORIGINAL_SET_SIZE
0	576195548.2
1	542792907.7
2	570345847.2
3	511649052.3
4	572671416.4
5	550775156.4
6	584589656.1

b. ***"Union-Size-Estimation problem"*** ***(Python code:*** *HW_1_part_2_2_b.py****, custom module:*** *hwmodule.py****)***

The *k-min* approach also provides a simple and very useful operation called $UNION$, we can use it to obtain the sketch of the union of two sets using the two sketches of those sets. The $UNION$ operation is defined as follows:

$$UNION\big(S(A), S(B)\big) = \{c_1, c_2, \dots, c_k\}, such\ that\ c_i = argmin\{r(a_i), r(b_i)\}.$$

We are going to use this operation to generate the sketches of the unions and call back the estimator we used before to compute an estimate for the union sizes.

The *output file:*

Union_Set_id	set_of_sets_ids	ESTIMATED_UNION_SIZE
0	{0, 1}	853785197.3
1	{0, 10}	966937452.5
2	{1, 11, 21}	852489620.4
3	{2, 12, 22, 32}	890318003
4	{3, 13, 23, 33, 43}	883318647.6
5	{4, 14, 24, 34, 44}	930116986.6
6	{12, 22}	539664418.3
7	{23, 33}	883318647.6
8	{4, 34, 44}	930116986.6
9	{4, 44}	636950861.2

REFERENCES:

[1]: *Estimating the Size of the Transitive Closure in Linear Time BY Edith Cohen*

[2]: MinHash Sketches: A Brief Survey BY Edith Cohen (June 2016)

[3]: P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. J. Comput. System Sci., 31:182–209, 1985.

[4]: Probabilistic Counting, Andrea Marino, University of Pisa (March 2015)