Data Mining Technology for Business and Society

Alice Schirinà, Maria Luisa Croci

10 aprile 2019

1 Search Engine Evaluation

The main goal of the first part is to provide a rank of our Search Engines using three main evaluation measures:

- Mean reciprocal rank (MRR)
- R-precision
- Normalized discounted cumulative gain (nDCG)

Starting from 1400 documents (in html format) we have to index their title and body into a schema in order to compute the Search Engines using as input 222 given queries.

We used the following 7 analyzers from the `analysis` module of `whoosh` package:

- KeywordAnalyzer(): Parses whitespace- or comma-separated tokens
- SimpleAnalyzer(): Composes a RegexTokenizer with a LowercaseFilter
- StandardAnalyzer(): Composes a RegexTokenizer with a LowercaseFilter and optional StopFilter
- StemmingAnalyzer(): Composes a RegexTokenizer with a lower case filter, an optional stop filter, and a stemming filter
- FancyAnalyzer(): Composes a RegexTokenizer with an IntraWordFilter, LowercaseFilter, and StopFilter
- NgramAnalyzer(): Composes an NgramTokenizer and a LowercaseFilter
- LanguageAnalyzer(): LanguageAnalyzer(): Configures a simple analyzer for the given language, with a LowercaseFilter, StopFilter, and StemFilter

composed with the `scoring algorithm classes`:

- BM25F()
- TF_IDF
- Frequency

thus the total amount of search engine configurations tested is 21, composed as all the combinations of the 7 analyzers and 3 scoring functions.

1.1 Evaluation Metrics

In order to compute the measures we compare our results with the Ground Truth.

MRR The following table shows the MRR for each configuration:

Configuration	MRR
SE_KeywordAnalyzer_Frequency	0.05
SE_KeywordAnalyzer_TF_IDF	0.15
SE_KeywordAnalyzer_BM25F	0.46
SE_SimpleAnalyzer_Frequency	0.05
SE_SimpleAnalyzer_TF_IDF	0.17
SE_SimpleAnalyzer_BM25F	0.47
SE_StandardAnalyzer_Frequency	0.31
SE_StandardAnalyzer_TF_IDF	0.38
SE_StandardAnalyzer_BM25F	0.49
SE_StemmingAnalyzer_Frequency	0.31
SE_StemmingAnalyzer_TF_IDF	0.39
SE_StemmingAnalyzer_BM25F	0.51
SE_FancyAnalyzer_Frequency	0.31
SE_FancyAnalyzer_TF_IDF	0.38
SE_FancyAnalyzer_BM25F	0.49
SE_NgramAnalyzer_Frequency	0.26
SE_NgramAnalyzer_TF_IDF	0.33
SE_NgramAnalyzer_BM25F	0.42
SE_LanguageAnalyzer_Frequency	0.33
SE_LanguageAnalyzer_TF_IDF	0.4
SE_LanguageAnalyzer_BM25F	0.52

Looking at the table above it can be observed that the acceptable configuration, the ones with $MRR \geq 0.32$ are 13 on 21: (`SE_KeywordAnalyzer_BM25F`, `SE_SimpleAnalyzer_BM25F`, `SE_StandardAnalyzer_TF_IDF`, `SE_StandardAnalyzer_BM25F`, `SE_StemmingAnalyzer_TF_IDF`, `SE_StemmingAnalyzer_BM25F`, `SE_FancyAnalyzer_TF_IDF` `SE_FancyAnalyzer_BM25F`, `SE_NgramAnalyzer_Frequency`, `SE_NgramAnalyzer_TF_IDF`, `SE_NgramAnalyzer_BM25F`, `SE_LanguageAnalyzer_Frequency`, `SE_LanguageAnalyzer_TF_IDF`, `SE_LanguageAnalyzer_BM25F`)

R-precision Now for each acceptable configuration we have provided the following informations:

Configuration	Mean	Min	First	Median	Third	Max
SE_KeywordAnalyzer_BM25F	0.29	0.0	0.14	0.25	0.45	1.0
SE_SimpleAnalyzer_BM25F	0.3	0.0	0.17	0.29	0.44	1.0
SE_StandardAnalyzer_TF_IDF	0.25	0.0	0.11	0.25	0.33	1.0
SE_StandardAnalyzer_BM25F	0.3	0.0	0.17	0.29	0.5	1.0
SE_StemmingAnalyzer_TF_IDF	0.23	0.0	0.0	0.2	0.33	1.0
SE_StemmingAnalyzer_BM25F	0.31	0.0	0.17	0.28	0.5	1.0
SE_FancyAnalyzer_TF_IDF	0.25	0.0	0.1	0.24	0.33	1.0
SE_FancyAnalyzer_BM25F	0.3	0.0	0.17	0.29	0.5	1.0
SE_NgramAnalyzer_TF_IDF	0.22	0.0	0.0	0.2	0.33	1.0
SE_NgramAnalyzer_BM25F	0.27	0.0	0.14	0.25	0.36	1.0
SE_LanguageAnalyzer_Frequency	0.21	0.0	0.0	0.17	0.33	1.0
SE_LanguageAnalyzer_TF_IDF	0.24	0.0	0.07	0.2	0.33	1.0
SE_LanguageAnalyzer_BM25F	0.33	0.0	0.2	0.29	0.5	1.0

nDCG@k Plot The following curves represents for each acceptable configuration the nDCG average over all the queries when k move from 1 up to 10. As we expected as k increase the average increase too.

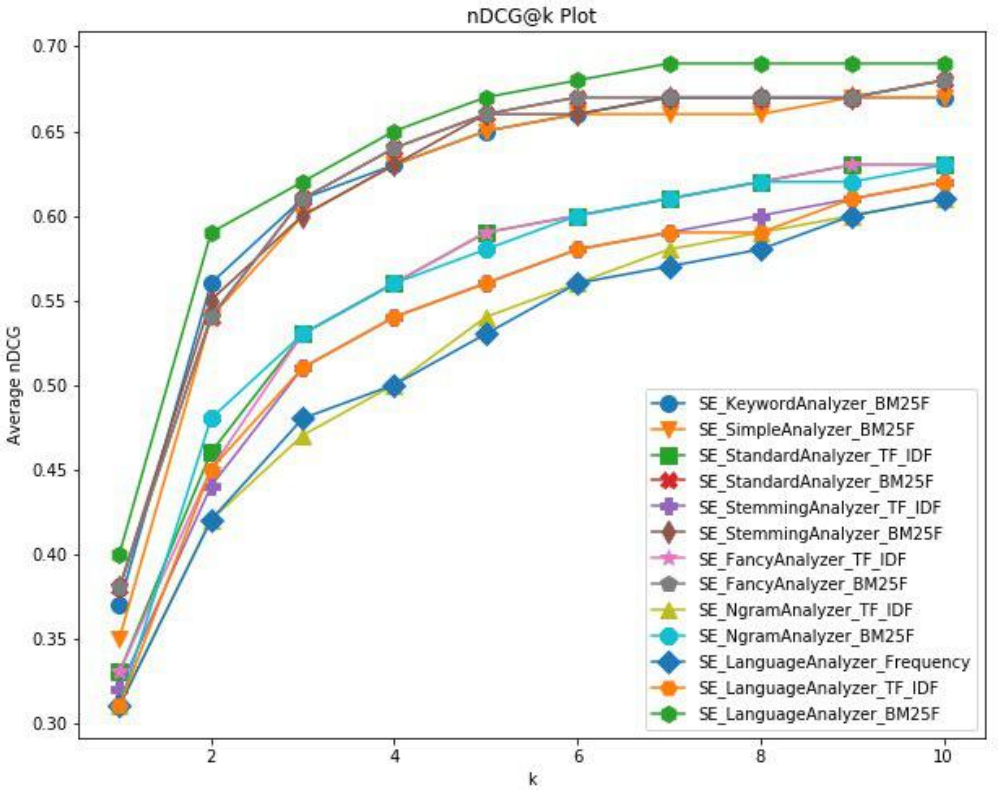

Figura 1: nDCG@k plot

2 Near Duplicates Detection

The main goal for the second part is to find out the couples of *near_ duplicates* documents from a dataset of around 261K songs. In order to do that we looked only at the columns "id" and "lyrics". At first we removed punctuation and upper case and then we produced a subsequences of shingles, each of size 3. At the end for each shingle we have assigned a unique id and we have replaced the shingles' id for each song.

Questions

First and fifth question:
We plotted the S-curve in function of the similarity for some couples of r and b. In particular, we looked at the trend of the curve for the couple (5, 10), (5, 20), (10, 20), (15, 30) and we observe (Figure 2) that the probability of having False-negatives is lower if we choose as optimal values $r = 10$ and $b = 20$ (and so $n = r * b = 200$).

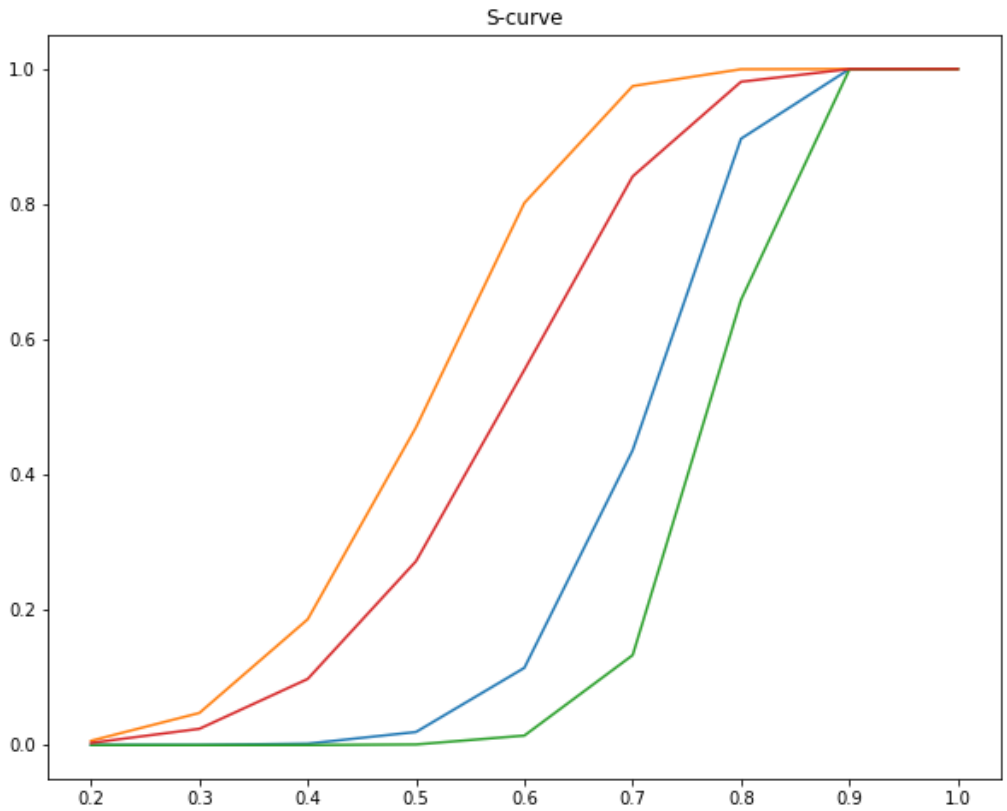

Figura 2: S-curve plot

Second and third question:
As requested we estimated the values of False-negatives for the following values of the Jaccard similarity [0.88, 0.9, 0.95, 1] and obtained [1.5e-03, 1.9e-04, 1.2e-08, 0].

Then we computed the values of False-positives for [0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55,0.5] and obtained [0.98, 0.89, 0.68, 0.44, 0.24, 0.11, 0.05, 0.02].

Fourth question:
Then we moved on to the detection of the *near_ duplicates*: in order to do this we first used the python script `hash_functions_creator` to generate our 200 hash functions and then the `Near_Duplicates_Detection_Tool`. By the latter we obtained the candidate pairs of near duplicates, so we decided to compute the Jaccard similarity between these couple and to divide those for which the similarity is actually bigger (or equal) than 0.88 from the others. Finally we dropped those for which the Jaccard similarity was less than 0.88 and created a new .tsv file called "near_dup.tsv" which contains the actual near duplicates we detected.

Sixth question:
The execution time is 3.3 minutes for the sketches creation and about 5 for the whole python script.

Seventh question:
Finally we obtained 41824 near_duplicates couples stored in the file "near_dup.tsv" together with their Jaccard similarities.