Information Retrieval

Assignment 2

M. Fahad Zafar L14-4048
Adil Iqbal L14-4221

Scoring Function Performance

1. TF
2. TF-IDF
3. BM25
4. JM Smoothing

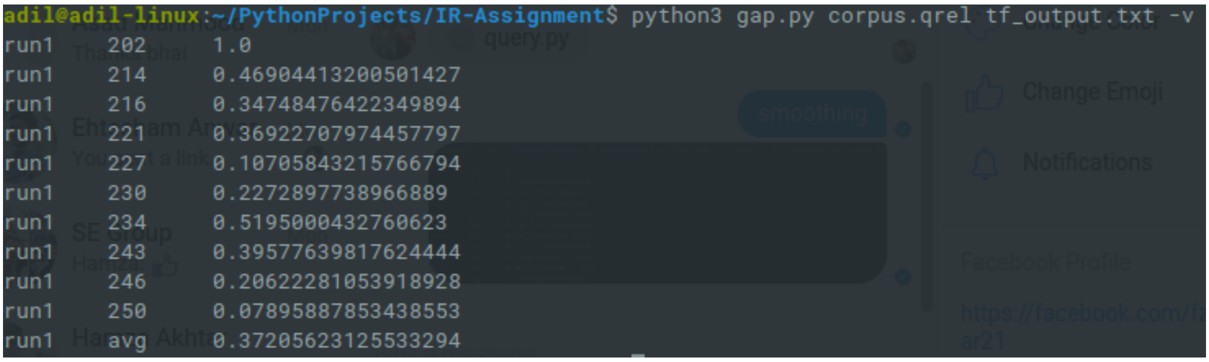
- **TF**

Total Time: 31 Seconds (Intel Core i3 on 1.80 GHz)

First, query vector is calculated using OkapiTF function which is used to calculate query normalization as well for each query. This is followed by calculation of vectors and normalization of each term of document inside the cosine similarity, which is then multiplied with the query vector to get the dot product and normalization of terms and query term to get the denominator in the cosine similarity. After these steps, we have document ranking for single query sentence against all the documents in the corpus.

The highest rank scores for each query statement are given in the table given below and the gap scores are given in the picture given below. Because of huge number of zeroes, gap average comes out to be near 0.

Gap Score:



```
adil@adil-linux:~/PythonProjects/IR-Assignment$ python3 gap.py corpus.qrel tf_output.txt -v
run1    202     1.0
run1    214     0.46904413200501427
run1    216     0.34748476422349894
run1    221     0.36922707974457797
run1    227     0.10705843215766794
run1    230     0.2272897738966889
run1    234     0.5195000432760623
run1    243     0.39577639817624444
run1    246     0.20622281053918928
run1    250     0.07895887853438553
run1    avg     0.37205623125533294
```

Results:

Query ID	**Top 3 Ranked Documents** **Document ID (Rank Score)**
202	1908 (0.62) 2555 (0.40) 233 (0.35)
214	1782 (0.32) 742 (0.30) 2856 (0.28)
216	3256 (0.57) 3161 (0.36) 2645 (0.34)
221	251 (0.37) 2315 (0.36) 72 (0.30)
227	3112 (0.24) 487 (0.24) 836 (0.24)
230	20 (0.38) 21 (0.38) 970 (0.28)
234	327 (0.44) 591 (0.34) 586 (0.33)
243	890 (0.30) 1419 (0.29) 1509 (0.24)
246	2197 (0.67) 3208 (0.35) 58 (0.29)
250	76 (0.20) 475 (0.20) 1292 (0.19)

- **TF-IDF**

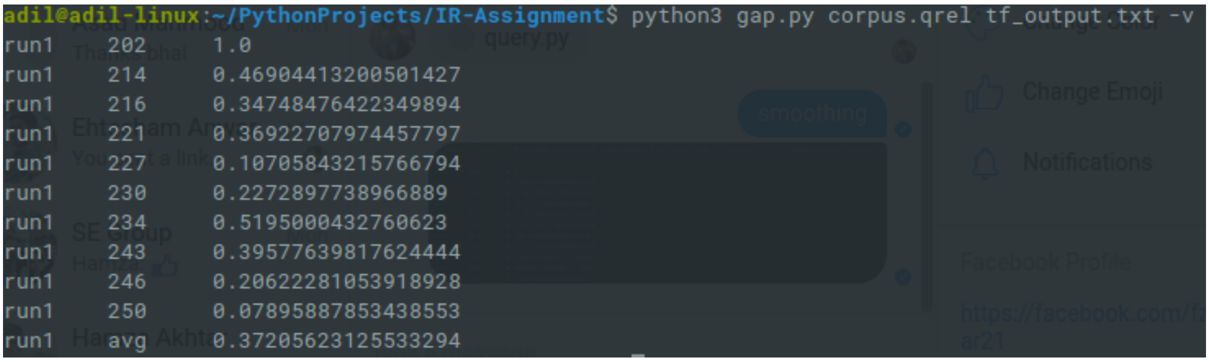
Total Time: 56 Minutes (Intel Core i3 on 1.80 GHz)

Similar to TF algorithm, query vectors and normalizations are calculated for each query stem with the modified formula. After this, vectors and normalizations are calculated for each term inside the documents which are multiplied with each query term vector to get the scoring.

The highest rank scores for each query statement are given in the table given below and the gap scores are given in the picture given below. Because of huge number of zeroes, gap average comes out to be near 0.

There weren't any noticeable changes in the TF-IDF than the TF algorithm. Instead, in our case, TF-IDF performed slower and produced very similar ranking as compared to TF algorithm. Thus, TF algorithm performed better in our case as it is efficient and gives out same results.

Gap Score:



```
adil@adil-linux:~/PythonProjects/IR-Assignment$ python3 gap.py corpus.qrel tf_output.txt -v
run1    202     1.0
run1    214     0.46904413200501427
run1    216     0.34748476422349894
run1    221     0.36922707974457797
run1    227     0.10705843215766794
run1    230     0.2272897738966889
run1    234     0.5195000432760623
run1    243     0.39577639817624444
run1    246     0.20622281053918928
run1    250     0.07895887853438553
run1    avg     0.37205623125533294
```

Results:

Query ID	**Top 3 Ranked Documents** **Document ID (Rank Score)**
202	1908 (0.76) 2555 (0.40) 233 (0.32)
214	656 (0.21) 742 (0.20) 2856 (0.18)

216	3012 (0.28) 3256 (0.26) 2925 (0.22)
221	251 (0.31) 2183 (0.22) 269 (0.22)
227	3112 (0.23) 695 (0.18) 487 (0.16)
230	352 (0.20) 2754 (0.14) 20 (0.13)
234	591 (0.31) 327 (0.29) 586 (0.25)
243	1419 (0.21) 1772 (0.20) 890 (0.19)
246	2197 (0.66) 3207 (0.20) 3208 (0.19)
250	475 (0.14) 707 (0.12) 113 (0.12)

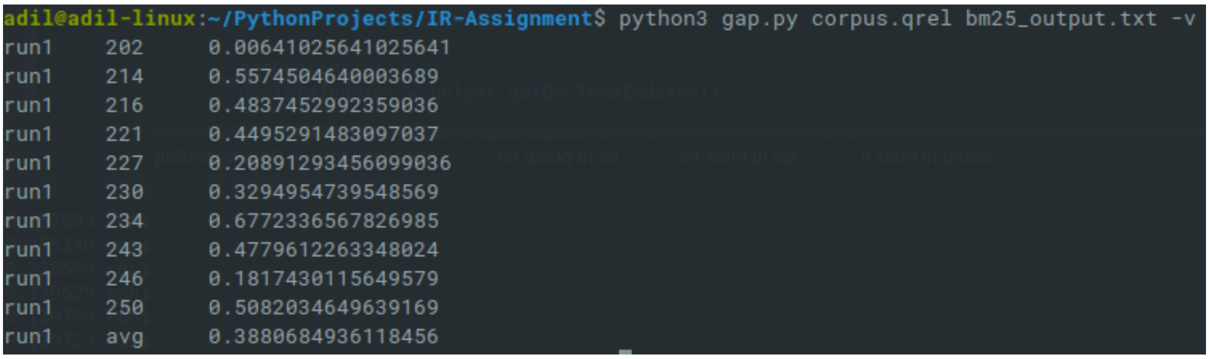
- **BM25**

Total Time: 11 Seconds (Intel Core i5 on 2.9 GHz)

For each query stem, all things required for BM25 Formula are calculated and on each on them, formula is applied.

It ranked documents pretty swiftly as compared to TF-IDF specially. But unlike TF and TF-IDF, there was a lot of difference in ranking results of bm25.

Gap Score:



```
adil@adil-linux:~/PythonProjects/IR-Assignment$ python3 gap.py corpus.qrel bm25_output.txt -v
run1    202     0.00641025641025641
run1    214     0.5574504640003689
run1    216     0.4837452992359036
run1    221     0.4495291483097037
run1    227     0.20891293456099036
run1    230     0.3294954739548569
run1    234     0.6772336567826985
run1    243     0.4779612263348024
run1    246     0.1817430115649579
run1    250     0.5082034649639169
run1    avg     0.3880684936118456
```

Results:

Query ID	Top 3 Ranked Documents Document ID (Rank Score)
202	233 (7.46) 1273 (7.45) 3346 (7.45)
214	2513 (6.51) 128 (6.51) 68 (6.51)
216	3156 (5.83) 1791 (5.82) 1042 (5.82)
221	3009 (6.07) 1750 (6.06) 3004 (6.05)
227	1364 (3.68) 831 (3.67) 1225 (3.67)
230	2727 (4.94) 2211 (4.90) 3019 (4.90)
234	1788 (7.06) 3457 (7.06) 861 (7.06)

243	3038 (3.81) 533 (3.80) 1116 (3.80)
246	1173 (7.50) 1822 (7.50) 1206 (7.50)
250	1891 (5.04) 1890 (5.04) 1893 (5.04)

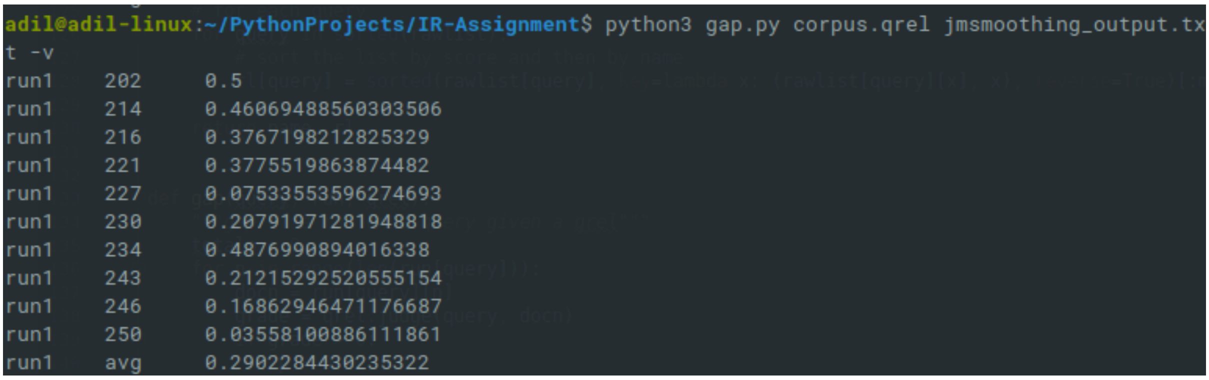
- ## JM Smoothing

Total Time: 8 Seconds (Intel Core i5 on 2.9 GHz)

For each query stem, all things required for JM Smoothing Formula are calculated and on each on them, formula is applied.

It ranked documents fastest of all previous. But the ranking scores for each document was very low in JM Smoothing. On Average, the score was under 0.1. Some of its queries had almost similar ranked documents like TF. But the overall time of JM Smoothing was way better than the rest.

Gap Score:



```
adil@adil-linux:~/PythonProjects/IR-Assignment$ python3 gap.py corpus.qrel jmsmoothing_output.tx
t -v
run1    202     0.5
run1    214     0.46069488560303506
run1    216     0.3767198212825329
run1    221     0.3775519863874482
run1    227     0.07533553596274693
run1    230     0.20791971281948818
run1    234     0.4876990894016338
run1    243     0.21215292520555154
run1    246     0.16862946471176687
run1    250     0.03558100886111861
run1    avg     0.2902284430235322
```

Results:

Query ID	**Top 3 Ranked Documents** **Document ID (Rank Score)**

202	233 (0.09) 1908 (0.09) 231 (0.062)
214	50 (0.07) 51 (0.07) 134 (0.06)
216	3256 (0.07) 1377 (0.06) 3219 (0.05)
221	251 (0.05) 170 (0.04) 2315 (0.04)
227	2769 (0.08) 564 (0.07) 144 (0.06)
230	345 (0.07) 20 (0.07) 21 (0.07)
234	327 (0.11) 2450 (0.09) 1170 (0.08)
243	498 (0.08) 1128 (0.08) 494 (0.08)
246	2197 (0.12) 1519 (0.12) 957 (0.07)
250	227 (0.07) 1168 (0.07) 2459 (0.07)

- **Hard Query**

For last query (250), results were different for each algorithm. That is why, this was harder than the rest.

- **Best Algorithm**

BM25 has more GAP score than any other algorithm.