Information Retrieval Report:

Name: Ilay cohen, ID:206515744, Email: Ilayc11@gmail.com

Name: Victor Gavrilenko, ID: 209406255, Email: vicgav97@gmail.com

GitHub: GitHub Repo

Descriptions of Experiments with Evaluations, Visualizations, and Statistics:

First Experiment:

Our initial experiment involved leveraging components from prior assignments to build inverted indexes for both the articles body and title. We utilized a Tokenizer object to preprocess text data, which included filtering out words shorter than two characters or longer than 22 characters, removing $stopwords$ sourced from the $`nltk.corpus`$ library, and stemming tokens.

Following this, we introduced the $CosineSim$ class to compute the cosine similarity between a query and an article body at runtime. To support this functionality, we expanded our indexing system to include additional structures such as L2Norm and DL for the body. These structures allowed us to efficiently calculate document norms and lengths necessary for similarity computations.

In the subsequent phase, we developed the $PageRankSim$ object to evaluate query similarity against article titles. By incorporating normalized PageRank values into our scoring mechanism, we aimed to provide more comprehensive and relevant search results.

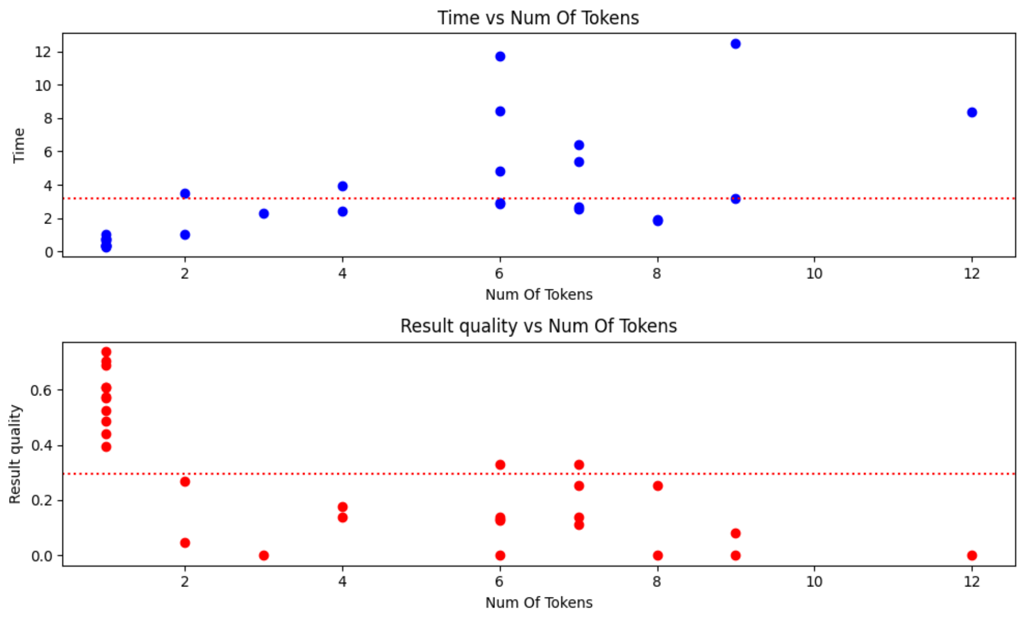
To generate the final ranking, we assigned weights of 0.7 to titles and 0.3 to bodies within our merge function. This function combined the top-scoring results from both $CosineSim$ and $PageRankSim$, delivering the best 30 matches as tuples containing document IDs and titles. Furthermore, to optimize runtime performance, we generated an additional index to quickly retrieve article titles by their corresponding document IDs.

Evaluation:

The way we evaluated our first experiment is by using the method provided to us in $'run\_frontend\_in\_colab\_.ipynb'$ (we evaluated all experiments on the GCP). We tried several values of k for precision_at_k and $f1\_at\_k$ and obtained relatively poor results. For $precision\_at\_k$, we obtained values ranging from 0.2 to 0.4, and for f1_at_k, we obtained values ranging from 0.05 to 0.1. Additionally, we used the $'results\_quality'$ and $recall\_at\_k$ methods to evaluate our experiment and also obtained pretty bad results (which can be seen in the graph below).

Visualization & Statistics:

Next, we will present a few visualizations and statistics from our first experiment on the training set provided to us. The graphs are based on "$result\_quality$" method given to us.



$< Time\ Mean$: 3.1653 $seconds$, $\quad Quality\ Mean$: 0.2952 $>$

Second Experiments:

In our second experiment, we aimed to address issues encountered with our indexes. Firstly, we observed that our $Tokenizer$ failed to identify certain meaningful words like "ai" and "3d" due to our word length filtering. To fix this, we included a list of two-character meaningful words.

Additionally, we noticed suboptimal performance quality-wise with the $CosineSim$ and $PageRankSim$ objects. To enhance it, we implemented the $BM25$ class. We created two objects—one for the title and one for the body—and utilized their corresponding indexes, along with page rank and page views dictionaries.

Experimenting with the hyperparameters, we found optimal values for $k1 = 2.2$ and $b = 0.85$ for the title $BM25$ object, and $k1 = 1.75$ and $b = 0.65$ for the body object.

We also adjusted weightings for question and non-question queries, settling on 0.425 for title and 0.575 for body weights for questions, and 0.725 for title and 0.275 for body weights for non-question queries.

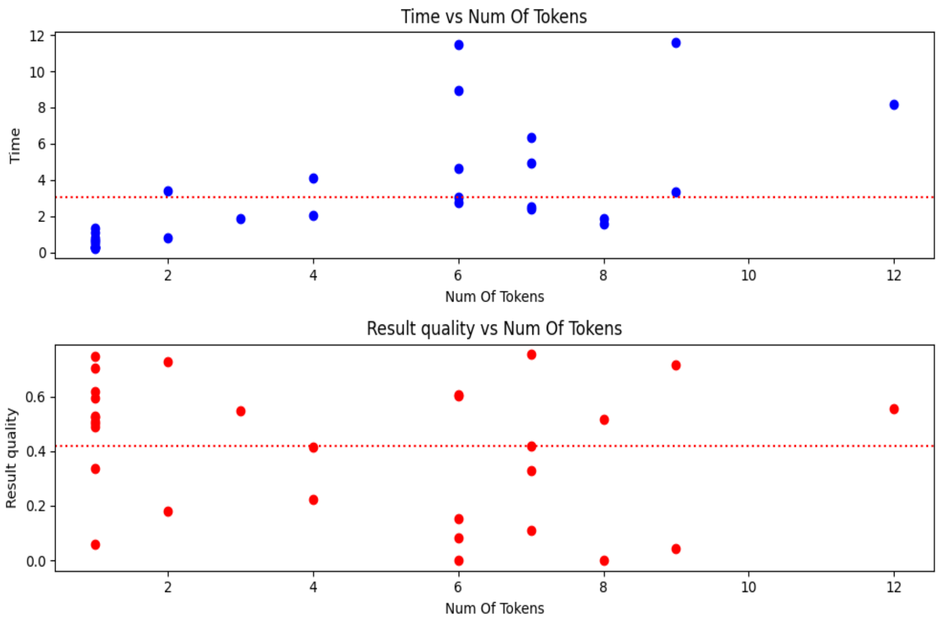
After retrieving the top N documents with scores from both the title and body, we merged them and combined their $BM25$ scores with the page rank and page views scores to enhance performance and prioritize documents with higher page rank and page views values. Normalizing these values, we utilized $\log_e()$ for page rank and $\log_6()$ for page views. Finally, we presented the best 30 documents for a given query.

Evaluation:

In the second experiment, we used the same evaluation techniques as with the first one. For example, with $precision\_at\_k$ with $k = 5$, we obtained an average score of 0.673, and for $f1\_at\_k$ with $k = 30$, we obtained an average score of 0.3123, which are much better than the results of the first experiment. Moreover, we evaluated our search engine with the $'results\_quality'$ method, which can be seen in the graph below.

Visualization & Statistics:

Next, we will present a few visualizations and statistics from our second experiment on the training set provided to us. The graphs based on "$result\_quality$" method.



$<Time\ Mean: 2.9140\ seconds, Quality\ Mean: 0.4205>$

Qualitative Evaluation of the Top 10 Results:

For this evaluation, we utilized the $'precision\_at\_k'$ method with $k = 10$, as the task required assessing the top 10 results. One of the queries that achieved a perfect score of **1.0** was *'What is the meaning of the term "Habeas Corpus"?'* Conversely, the query *'Who is considered the "Father of the United States"?' yielded* the lowest score of **0.1**. The high score for the first query can be attributed to its specificity, containing terms like 'Habeas' and 'Corpus,' which likely corresponded to informative documents in the corpus. Additionally, the resulting high page rank and page views of retrieved documents may have influenced their prioritization. Conversely, the second query, with tokens 'considered,' 'Father,' 'United,' and 'States,' may not have provided sufficient information for relevant document retrieval. Treating 'United States' as separate tokens during index creation could have contributed to mismatches at runtime. Furthermore, examining retrieved documents revealed direct matches like 'John Dickinson' indicating potential limitations in query relevance. To address this, employing deep learning models such as $Doc2Vec$ or $Word2Vec$ for semantic content comparison could enhance result robustness and relevance.

We attempted to execute $'gsutil\ du\ -ch\ gs://inverted\_indexes\_bucket'$ command to list all index files as desired. However, it resulted in an excessive number of files. Consequently, we will list the indexes we generated in a table and provide a link to our bucket. (link to the bucket)

Name	Size
body_dl_.pkl	44.54 MiB
body_index_.pkl	4.32 MiB
body_index_final.pkl	11.7 MiB
body_stem_index.pkl	15.96 MiB
title_dictionary.pkl	168.88 MiB
title_dl_.pkl	42.34 MiB
title_index_.pkl	468.78 KiB
title_index_final.pkl	823.6 KiB
title_stem_index.pkl	873.29 KiB
doc_l2_norm.pkl	84.73 MiB
pageRank.pkl	32.33 MiB
pageviews-202108-user.pkl	73.5 MiB
TOTAL: 12 objects, (~480 MiB)	