Part 2: Indexing and Evaluation

0. Data preparation

Before starting this part of the project, it is important to note that we have incorporated all the work completed in the previous part into a new notebook. and created a new dataframe performing a series of operations with *data_df* and *tweet_document_ids_map* to merge them based on a common column 'tweet_id'.

1. Indexing

In this next section, we'll explore the construction of inverted indexes using the pre-processed data. Inverted indexes are a fundamental component of information retrieval systems, and they play a crucial role in organizing and facilitating efficient search operations. By creating these indexes from our pre-processed data, we are laying the foundation for powerful and precise information retrieval, search engines, and data analysis. To achieve this, we developed a function named *create_index()* specifically designed for the task. This function takes a dataframe containing tweets as input, processes each tweet to extract terms and their respective positions, and subsequently constructs an inverted index. This index associates each term with the documents in which it occurs, as well as the positions within those documents. Additionally, the function maintains a mapping of document IDs to their corresponding URLs. The resulting inverted index serves as a valuable resource for text-based search and retrieval tasks, significantly enhancing the efficiency and effectiveness of these operations.

Then, we built a search engine called *search()* which will return the list of tweets in documents for each query term, then take the union of them. To assess the performance of our search engine, we merged the keywords derived from the word cloud analysis. These keywords provided valuable insights into the most prominent and relevant terms within our dataset. With this information at hand, we created the following five queries:

- Query 1: "Russian intervention in Ukraine timeline"
- Query 2: "NATO involvement in Ukraine"
- Query 3: "Dnipro region war news"
- Query 4: "Zelenski leadership during Ukraine war"
- Query 5:" Stop Putin's aggression in Ukraine"

Finally, we ranked the results obtained of a search based on the tf-idf weights for the previous queries. You can see the most relevant documents of these queries in the notebook.

2. Evaluation

2.1) Evaluation components

2.1.1) Using a subset of the dataset

We began our evaluation by initially working with a subset of the dataset. While experimenting with different query types to retrieve the necessary information, we found that the most effective queries were:

Information need	Respective Query	Results
What is the discussion regarding a tank in Kharkiv?	"Kharkiv tank"	Truly relevant: 3. Not relevant: 0
What discussions are there about the Nord Stream pipeline	"Nord Stream pipeline"	Truly relevant: 6. Not relevant: 0
What is being said about the annexation of territories by Russia?	"Russia Annexation territories"	Truly relevant: 2. Not relevant: 0

For each query, we work with a total of 20 documents in this approach. This comprises 10 documents that are deemed relevant to the query, 10 documents that are labeled as not relevant to the query under consideration, and among these 40, 20 documents are relevant to the remaining queries.

2.1.2) Being the experts judges

In this section, we take a closer look at the results generated by our search engine and conduct a thorough manual evaluation to determine the relevance of each document retrieved. To manually assess the relevance of documents (that we obtained with the tf-idf evaluation), we employed the following criteria for each query:

- Documents considered relevant were assigned a score of (1).
- Documents deemed not relevant received a score of (0).

These two evaluation components collectively aim to assess the performance of the system or methodology under scrutiny. The baseline offers an initial benchmark, while expert judgment provides a subjective assessment, helping us categorize documents with respect to meeting or not meeting the information needs. This binary approach simplifies the evaluation process, making it easier to quantify and compare the effectiveness of the system or approach being evaluated.

2.2) Prior evaluation components

In the evaluation of our algorithm, we employ several evaluation techniques to assess the performance of each query.

Case 1:	**Query Q1: Kharkiv tank**	**Query Q2: Nord Stream pipeline**	**Query Q3: Russia Annexation territories**
Precision@K (P@K)	0.8	0.5	0.4
Recall@K (R@K)	1.0	1.0	1.0
Average Precision@K (P@K)	0.5142	0.3767	0.2
F1-Score@K	0.8889	0.6667	0.5714
Mean Average Precision (MAP)	0.3636	0.3636	0.3636
Mean Reciprocal Rank (MRR)	0.3333	1.0	0.5
Normalized Discounted Cumulative Gain (NDCG)	0.641	0.5861	0.3824

Case 2:	**Query Q1: Russian intervention Ukraine**	**Query Q2: NATO Ukraine**	**Query Q3: Dnipro war**	**Query Q4: Zelenski leadership Ukraine war**	**Query Q5: Stop Putin's aggression Ukraine**
Precision@K (P@K)	0.7	0.7	1.0	0.9	0,5
Recall@K (R@K)	1.0	1.0	1.0	0.0	0,0
Average Precision@K (P@K)	0.7024	0.6043	1.0	1.0	0,7417
F1-Score@K	0.8235	0.8235	1.0	0.0	0,0
Mean Average Precision (MAP)	0.8097	0.8097	0.8097	0.8097	0.8826
Mean Reciprocal Rank (MRR)	0.5	0.5	1.0	1.0	1.0
Normalized Discounted Cumulative Gain (NDCG)	0.7957	0.7493	1.0	1.0	0.8826

When comparing Case 1 and Case 2 based on various query metrics, Case 2 consistently demonstrates superior performance. It achieves higher precision values, suggesting a more accurate retrieval of relevant documents. Both cases excel in recall in the majority of the

queries, retrieving all relevant documents within the top K results. Case 2 also excels in average precision, indicating that relevant documents are ranked more prominently in its results. F1-scores in Case 2 are generally higher, reflecting a balanced trade-off between precision and recall. Despite sharing the same mean average precision, Case 2 outperforms Case 1 in mean reciprocal rank and normalized discounted cumulative gain (NDCG) for the common queries. This demonstrates that Case 2 offers more effective ranking strategies, making it the preferred choice for better retrieval performance.

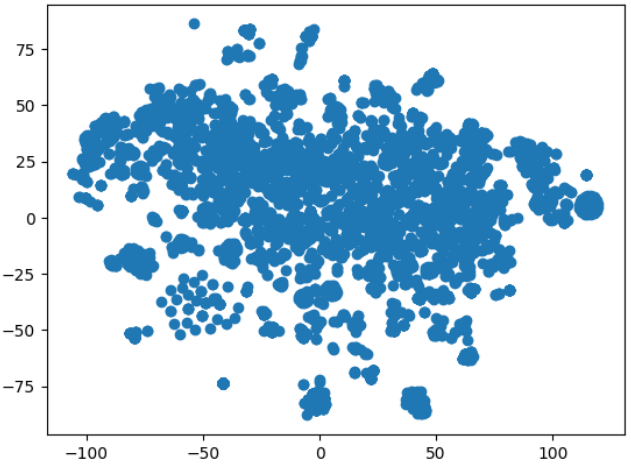
2.3) Two-dimensional scatter plot through the T-SNE

In this upcoming section, we did a one-vector representation of our dataset, followed by the visualization of tweets in a two-dimensional scatter plot using the T-SNE (t-distributed stochastic neighbor embedding) algorithm. This approach allows us to simplify the tweet data for analysis and gain a clear visual understanding of the relationships between tweets.

By generating these one-vector representations, we streamline the complexity of individual tweets, facilitating easier analysis. The T-SNE algorithm further aids our analysis by providing a two-dimensional representation that helps reveal patterns and relationships in the data. This section will explore the technical aspects of these processes, offering valuable insights into our dataset and its underlying structure.

The following distribution along the y-axis from -75 to 75 and the x-axis from -100 to 100 provides an overview of where data points are located in the two-dimensional space.

The dense cluster of points around *y=[-25, 50]* and *x=[-100, 100]* suggests that there is a group of tweets or data points that share similarities in their word embeddings within this range.



Outside of the primary concentration area, there are still points scattered throughout the plot, albeit less densely. These outliers might represent tweets or data points that are distinct or unrelated to the main cluster. Exploring some of these outliers may reveal interesting or unique content.

Furthermore, the data points seem to be symmetrically distributed around the origin (0,0), with a roughly circular pattern. This suggests that the word embeddings have a balanced distribution and are not skewed in a particular direction.