Report Project part 2:

In the indexing part, we decided to implement both the inverted index and the TF-IDF algorithm. In order to do that, we have followed the guidelines from the labs, so in function create_index we create an inverted index for all the terms (using the terms obtained and filtered by the function implemented in the previous part of the project) and we also compute the tf, df and idf values for all the terms. Thus, with a single call of the function we obtain an inverted index and the tf-idf values for all the terms in all the tweets inside the set inserted as input. Therefore, we can use this function to obtain all this information for any subset of the tweets, which works to obtain the subsets for the evaluation of the different queries, but also to experiment with different subsets of the tweets.

Moreover, we have implemented a ranking algorithm based on these tf-idf values, which takes the outputs from the create_index function and the desired query as arguments, returning an ordered list of tweets according to the specified query, and also their scores. The original function, called search_tf_idf, returned only the ordered tweets, but since we need the scores for the evaluation part we modified it in order to obtain the scores as well. An important note here is that this algorithm immediately excludes all the words that do not contain the terms from the query, which means that the obtained tweets are a subset of the input set of tweets. Here, we could also experiment trying different queries for the different subsets to see what are the closest tweets to that query according to our chosen criteria.

In order to define the queries for the information needs described in the baseline cases, we eventually chose the following queries:

1. Query 1: Tank Kharkiv
2. Query 2: Nord Stream Pipeline
3. Query 3: Annexation territories

The selection of these queries was carefully chosen considering the popularity of the different terms related to the information needs. Multiple definitions of the different queries were tried in order to obtain the most optimal results. For instance, the third query was originally “Annexation territories Russia”, but the word “Russia” made the evaluation metrics perform slightly lower, mainly due to the lack of this word in some relevant tweets. We eventually decided to remove it since all the tweets are extracted from the Ukranian-Russia War discussion, so the term russia was implicit, and this last version did indeed perform better.

To evaluate the tf-idf ranking method, we proposed the following five queries:

1. Query 1: Russian military intervention
2. Query 2: Impact of sanctions on Russia
3. Query 3: Russian propaganda in the conflict
4. Query 4: International response to Russia-Ukraine war
5. Query 5: Humanitarian crisis

The definition of these queries followed the popularity of their terms, since we chose the most popular terms that were more suitable to define different information needs.

For such queries, we have developed a subset of tweet ids in order to evaluate the ranking method.This subset contains 10 relevant and 10 nonrelevant tweets for each of the 5 queries, constituting a total of 100 tweets. The aforementioned subset can be found in “evaluation_custom_queries.csv”. This file contains the selected subsets following the same format as the provided file with the baseline queries. In order to select these subsets, we carefully considered which of the tweets were relevant for each of the queries and which were not.

The results we have obtained for each query are summarized in the following tables:

Baseline Queries (from the 3 information needs)

Query	Precision@K	Recall@K	Average Precision@K	F1@K	NDCG
Q2	1.0	1.0	1.0	1.0	1.0
Q3	1.0	1.0	1.0	1.0	1.0
Q1	0.8	0.8	0.76	0.8	0.8572

With a Mean Average Precision in the three queries of 0.9192592592592592 and a Mean Reciprocal Rank of 1. Thus, we can see how our algorithms perform quite optimally for the baseline queries, since the obtained results are satisfactory. Query 1 could slightly be improved, maybe with a light query reformulation.

Custom Queries

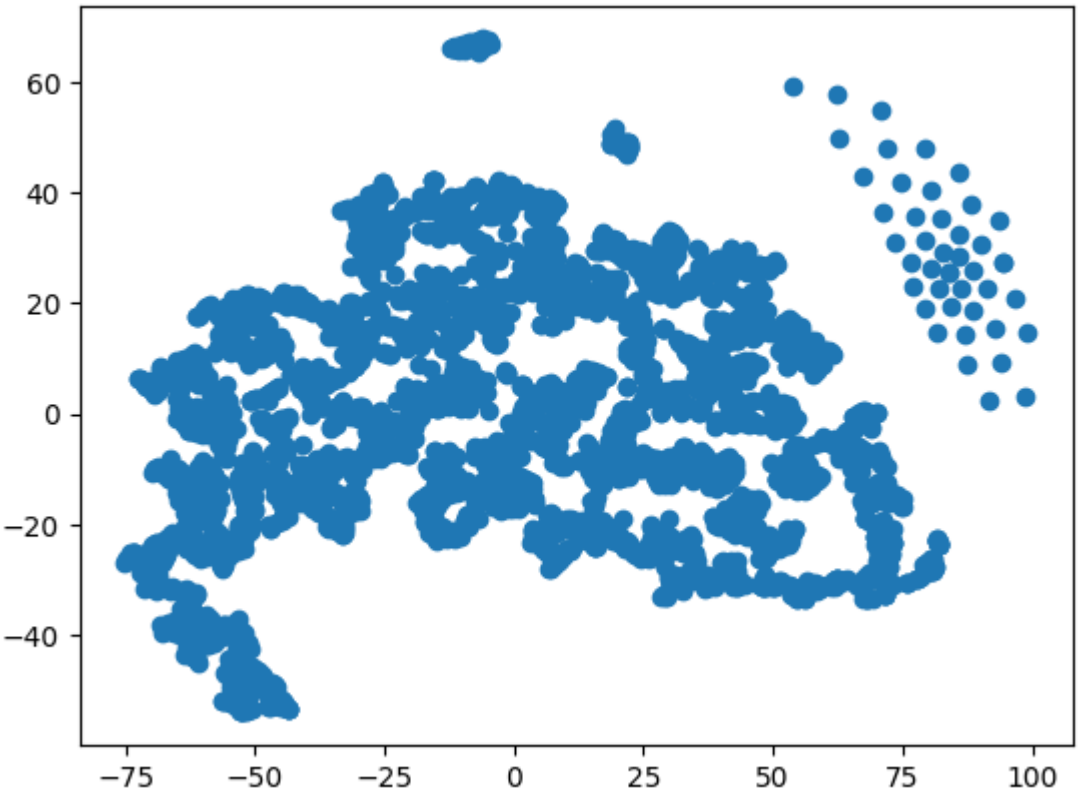
Query	Precision@K	Recall@K	Average Precision@K	F1@K	NDCG
Q1	0.4	0.4	0.3	0.4	0.54
Q2	0.9	0.9	0.9	0.9	0.93
Q3	1.0	1.0	1.0	1.0	1.0
Q4	0.7	0.7	0.47	0.7	0.68
Q5	0.9	1.0	1.0	0.95	1.0

With a Mean Average Precision of 0.6667892416225749 and a Mean Reciprocal Rank of 1. Query 1 was the one with the worst results of all the queries, making the mean values decrease. It captured relevant results but the order of comprehensiveness could significantly be improved. A further query reformulation or ground truth redefinition may improve these values. Query 2 and 5 were almost perfect, query 3 was perfect and query 4 was good but with some room for improvement yet. In conclusion, our algorithms performed satisfactorily well for all the queries except for the first one, which means that it returns appropriate results and we proceeded carefully at the meticulous process of selecting the subsets and their ground truth.

About the different evaluation techniques comparison, we notice how the recall and precision are exactly the same, and it is perfectly understandable, since for the evaluation we chose k = 10, and the ground truth was set to 10 tweets as well, both techniques returned the same values, since the divisors of their operations were the same. Also, we see how the average precision is exactly the same as the precision in the three best queries, but it is slightly inferior in the other. The average precision is similar to the precision, but taking into account the position in which the returned tweets are . Thus, if the average precision is equal to the precision in the best cases, it means that the algorithm did not only choose the relevant tweets well but also recognized their order of relevance well.

In the worst cases it makes sense that it is slightly inferior since with worse results it is most likely to order them differently. The f score also was quite high for the best 3 results and bad for the others, since it depends on the recall and precision. Finally the NDCG was pretty high for the best results and lower for the rest of queries, probably because they are more difficult queries.

Our vector representation (through T-SNE) was:



As we can see the plot shows multiple clusters, those clusters represent a group of tweets that have similar word2vec representation or in other words that share a similar content. We can clearly see a main cluster in the center that if we zoom in, we might divide it into several small clusters. At least we can split the bottom left cluster from the others. In the center above the main cluster we can see 2 smaller clusters. The clusters can be used to identify major themes or topics in the dataset.

Some regions in the plot are densely packed, meaning many tweets have very similar word2vec representations. In contrast, sparse regions indicate more variation in the content or context of tweets.

The isolated points on the right may indicate tweets that do not share any similarity between them. The overlaps between clusters could suggest that many tweets discuss multiple topics or that there's a gradient in the themes.

README

The main function of the code is "evaluation", this function is the one in charge of processing the queries, both the baseline queries provided and the custom queries made by us. This function essentially creates a small index with the corresponding subset of documents (tweets) that belongs to each query test and then it computes the prior evaluation metrics with functions defined for every metric.
To execute the evaluation of the baseline and custom queries it is only required to execute the “main()” function in “index.py” and it will print the metrics of evaluation for each query and then it will plot the 2D-scatter t-SNE of the word2vec representation of the tweets.

In evaluate_query.py, we have a functionality where the user can query our database of tweets and the system will write the ranked tweets in a txt file for the user to review. Essentially, it first creates the index for the entire tweets database, and has a loop where the user can provide queries.