Universitat Pompeu Fabra Barcelona

Pau Folch, 242211
Biel Alcaraz, 240100
Max Frigols, 241156

IRWA PROJECT REPORT
PART 2

1. Indexing

From part 1 we get the merged dataframe, which has as columns the tweed id, the processed text (just done in part 1), when it was created, hashtags, retweet count and the document id. Rows represent tweets with all the mentioned information.

Once we know this, we start converting our dataframe into a string with the following format: **tweet_id | document_id | Processed_Text**. Once we have this structure, we can implement it to the following functions for performing different aspects.

1.1. Inverted Index

After this first step, we have defined the function **create_index**, which receives as input the mentioned string, and returns two variables. The first one is called **index**, and it basically contains each term in the string, it tells us in which documents it appears and the exact position in it (if the term appears more than once in the same document, it tells us all of the positions in it). Then, the second variable **title_index** returns the tweet id corresponding to its doc_id.

Then, we have defined the **search** function, which receives as argument a query (here we will introduce the terms we want to search information about), and returns us all the documents where our queried terms appear.

Now we have already created our search engine, and we have proved it with 3 different **queries**. An insight is, although it seems to be quite obvious, that queries like “putin” or “countries against russia” appear in a lot more documents than others like “biden” or “australia”. As already said, it is evident that queries directly related to the war (like “putin” or countries against russia” will appear in more documents than others.

1.2. TF-IDF Ranking Method

Once we have created our search engine, we can go further and take a look at the documents our search engine returns. The TF-IDF method is useful when we want to evaluate the importance of a term in a document or in a collection of documents. For the application of this method, we have defined the **created_index_tf_idf** function, which does the same as the previous one, but now it also computed per each term:

- tf: normalized term frequency in each document it appears

- df: number of documents it appears
- idf: its inverse document frequency

These extra computations will be used for our query documents ranking.
The next step is to define the **search_tf_idf** function as before, but now this function will call another one (**rank_documents**). This last function receives the query, the documents, the index, title_index, and the recently computed values. With all this information, rank_documents performs the ranking of the most relevant documents depending on the terms in the query.

This new search now is based on the tf-idf weights, so, it is quite logical to think that the documents from a query like “putin” will be different from the ones searched in the previous section.

2. Evaluation

For the evaluation section, we have two main evaluation sections. To begin with, we will use three information needs and the ground truth files for each query. In this case, our queries, developed and ranked with score in the previous TF-IDF section, will be **tanks in kharkiv**, **nord stream pipeline** and **annexation of territories by russia**. After setting those three queries, we have developed the ground truth files as commented in the report, that is, the elements regarding each query, relevant and not relevant, as well as the not relevant elements of the other two queries. Afterwards, a merging process between the results obtained from the TF-IDF section and the ground truth files is done and so, the evaluation process with different algorithms is done.

2.1. Queries evaluation

In this exercise, we employed the TF-IDF (Term Frequency-Inverse Document Frequency) ranking mechanism to retrieve relevant documents for the different queries. This report presents a number of documents related to each query.

Query 1 = Information need 1: What is the discussion regarding a tank in Kharkiv?

Number of documents related with query 1 = 404

Query 2 = Information need 2: What discussions are there about the Nord Stream pipeline?

Number of documents related with query 2 = 46

Pau Folch, 242211
Biel Alcaraz, 240100
Max Frigols, 241156

Query 3 = Information need 3: What is being said about the annexation of territories by Russia?

Number of documents related with query 3 = 1437

2.2. Metrics for ranking

In the fields of information retrieval and machine learning, evaluating the performance of algorithms is crucial to understanding their effectiveness. Various metrics are employed to assess different aspects of a system's performance. In this report, we will use seven important metrics: Precision, Recall, Average Precision, F1 Score, MAP@10, Reciprocal Rank (RR), and Normalized Discounted Cumulative Gain (NDCG).

We will assess the seven crucial metrics for the three distinct queries provided, and here are the outcomes of our evaluation.

Metrics	Query 1	Query 2	Query 3
Precision	0.7	1.0	0.7
Recall	1.0	1.0	1.0
Average Precision	0.794	1.0	0.619
F1 Score	0.125	0.18181	0.125
Map 10	0.199	0.333	0.297
RR	1.0	1.0	0.5
NDCG	0.7606	1.0	0.6118

Metrics	Query 1	Query 2	Query 3	Query 4	Query 5
Precision	1.0	0.7	0.3	0.9	0.1
Recall	1.0	1.0	1.0	1.0	1.0
Average Precision	1.0	0.37	0.9	0.86	0.1
F1 Score	1.0	0.7	0.3	0.9	0.1
Map 10	0.2	0.16	0.08	0.2	0.13
RR	1.0	0.25	0.33	1.0	1.0
NDCG	1.0	0.53	0.25	0.92	0.22

Pau Folch, 242211
Biel Alcaraz, 240100
Max Frigols, 241156

Now, we are asked, for our 5 queries, to comment on each of the evaluation techniques stating how they differ, and which information gives each of them.

Precision@K (P@K)

Precision at K evaluates the proportion of relevant items among the top K recommendations or retrievals. A higher P@K value indicates a higher accuracy of the system at retrieving relevant items within the top K positions. Our queries show a range of P@K values, indicating varying levels of precision. For instance, query 1 with a P@K of 1.0 demonstrates perfect precision, while query 5 with a P@K of 0.1 indicates poor precision within the top K items.

Recall@K (R@K)

Recall at K measures the proportion of relevant items retrieved out of the total number of relevant items. A higher R@K value signifies better coverage of relevant items. In our case, all queries have a R@K value of 1.0, which means that all relevant items were retrieved for each query within the top K items.

Average Precision@K (P@K)

This metric averages the precision values at the ranks where relevant items are found, up to position K. It provides insight into precision across different cutoffs within the top K positions. Our queries exhibit varying Average Precision@K values, reflecting different levels of precision over the ranks of relevant items. Query 1 with a value of 1.0 demonstrates perfect average precision, while query 5 with a value of 0.1 reflects poor average precision.

F1-Score@K

The F1-Score at K is the harmonic mean of precision and recall at position K, providing a balance between the two metrics. It is useful when we want a single measure that balances both precision and recall. The F1-Score values in our queries range from 1.0 (perfect balance) in query 1 to 0.1 (poor balance) in query 5.

Mean Average Precision (MAP)

MAP calculates the mean of the average precision values for a set of queries. It gives an overall indication of the system's precision across different levels of recall. The MAP values in our queries are relatively low, indicating varying levels of system precision across queries.

Pau Folch, 242211
Biel Alcaraz, 240100
Max Frigols, 241156

Mean Reciprocal Rank (MRR)

MRR evaluates the ranking quality of the system by calculating the reciprocal of the rank at which the first relevant item is found. Higher MRR values indicate better ranking quality. Our queries show a range of MRR values, with queries 1, 4, and 5 having perfect ranking (MRR of 1.0) and query 2 showing poor ranking (MRR of 0.25).

Normalized Discounted Cumulative Gain (NDCG)

NDCG measures the quality of the ranking by considering the position of relevant items. Since it is normalized, values range between 0 (worst) and 1 (best), and it is especially useful in scenarios where the position of items matters. The NDCG values in our queries exhibit varying ranking quality, with query 1 having perfect ranking (NDCG of 1.0) and query 5 having poor ranking (NDCG of 0.22).

2.3. Two dimensional scatter plot for tweets

In this last point, we are asked to generate a two dimensional scatter plot for all tweets. To do so, we will use the T-SNE (T-distributed Stochastic Neighbor Embedding) in order to map the high-dimensional data points, which are all the tweets within their terms, to a lower-dimensional space.

Here we use the variable text_processed, which consists of a vector, in which each position is each tweet, of vectors (each tweet has its terms as positions of the vector). Once we have this variable the next step is to initialize a Word2Vec model using text_processed. Then we extract the word vectors from the trained model and store them in variable ‘X’. Afterwards, a TSNE model needs to be initialized (using scikit-learn library) with the number of components equal to 2, because we want to plot a 2D scatter plot. The next step is to apply TSNE to the word vectors “X”. Finally it is just a matter of plotting the scatter plot using the Matplotlib library. The following image consists of the final plot we got after all mentioned steps.