Part 2: Indexing and Evaluation

Abril Masgrau Turró, Jorge Martínez Rosa and Enric Riba Sáez
Information Retrieval & Web Analytics
University Pompeu Fabra

Indexing

0) Introduction

Now that we have preprocessed the text and we have the tweets structured, we create the inverted index, the search() function for the queries, and then the TF-IDF Ranking. With this, we obtain a ranking of documents sorted by relevance.

1) Inverted Index

For the inverted Index, we implement the create_index(). This function iterates through each tweet, and for each term, it stores its positions in the tweet. This results in a dictionary index that contains for each term, the tweets that have the term and its position. It also has a dictionary that, for each document_id, stores other information about the tweet.

2) Querying the Index

For this part, we have implemented the function search(). This function returns the tweets which have all the terms of the query. To evaluate the search engine, we have chosen these queries because they were popular topics during September 2022 on Twitter.

- United States
- Trump russia
- Bombs cities
- World War
- Nazi war

3) TF-IDF

For this part, we have implemented several functions in order to apply the TF-IDF algorithm and provide ranking based results. First of all, we have implemented “create_index_tfidf” that implements the inverted index and also computes tf, df and idf values. Then, we have implemented “rank_documents” that carries out the ranking of the results of a search, based on some given tf-idf weights and the inverted index. Finally, we have implemented “search_tf_idf” that returns the list of documents that contains any of the terms of the given query (i.e., it’s the union set of documents of the query terms).

The results of this can be seen in the notebook “Indexing_Part.ipynb”. In the last part, we display the ten most “predicted relevant” documents for each of the queries.

Evaluation

For this part, we have created two separate notebooks. One for the queries you already gave us, called "Evaluate_queries_statement" and the other one for our own queries, called "Evaluate_our_queries".

Statement queries:

"Evaluate_queries_statement.ipynb"

For the questions given in the statement, we have formulated the following queries:

- tank in Kharkiv
- Nord Stream pipeline
- annexation territories Russia

For those queries, we have computed the rank of the documents with the TF-IDF algorithm, and then merged the results with the "Evaluation_gt.csv" file, so we can do the evaluation. For each of the queries, we get a dataframe with 10 relevant documents and 30 that are not relevant (10 not relevant for the query and 20 relevant for the other ones).

Then, we have calculated some different evaluation techniques where you can select the query to apply to. In the following table for a value of K = 10, where the rows are the evaluation criteria and the columns are the queries, you can see the results obtained:

	Q1: 'tank in Kharkiv'	**Q2: 'Nord Stream pipeline'**	**Q3: 'annexation territories Russia'**
Precision@K	0.8	1	1
Recall@K	1	1	1
Average Precision@K	0.7532	1	1
F1-Score@K	0.8888	1	1
Mean Average Precision (MAP)	0.2511	0.3333	0.3333
Mean Reciprocal Rank (MRR)	0.3333	0.3333	0.3333
Normalized Discounted Cumulative	0.8553	1	1

Gain (NDCG)			

As we can see, almost all values are 1 for queries Q2 and Q3 and that means these cases are very concrete cases and the evaluation is done in a perfect way. Although Q1 doesn't have 1s, its values are very high as well.

Our queries:

"Evaluate_our_queries.ipynb"

For the queries selected by us, it's the same, but we had to decide which documents were relevant and which not ourselves so we have the groundtruth. To decide which ones were most relevant we searched between tweets and discussed whether they were relevant or not depending on the message. Trying to find tweets that were relevant and the algorithm didn't rank them and others that were not relevant but they were ranked high.

So we ended up having a dataset with 100 documents, 10 relevant and 10 not relevant, for each of the 5 queries. That means that for each of the queries, we get a dataframe with 10 relevant documents and 50 that are not relevant (10 not relevant for the query and 40 relevant for the other ones). Then we do the same procedure as before and below, you can observe the table for K = 10.

	Q1: 'United States'	**Q2: 'Trump russia'**	**Q3: 'Bomb cities'**	**Q4: 'World War'**	**Q5: 'Nazi War'**
Precision@K	0.8	0.2	0.7	0.7	0.6
Recall@K	1	1	1	1	1
Average Precision@K	0.7578	0.0786	0.6115	0.5658	0.3755
F1-Score@K	0.8888	0.3333	0.8223	0.8235	0.7490
Mean Average Precision (MAP)	0.1516	0.0157	0.1222	0.1132	0.0751
Mean Reciprocal Rank (MRR)	0.2	0.5	0.2	0.2	0.2
Normalized Discounted Cumulative Gain (NDCG)	0.8572	0.1	0.7601	0.7467	0.5993

In this case, we have chosen five different topics related to the Ukraine-Russian war. Q2 is quite different from the others because words Trump and Russia are very common words on their own in the dataset and our query search couldn't detect some tweets that were relevant in our rank document algorithm.

At K equals 10, the **Precision@K** algorithm is not 1 because we have some False Positives (we decided that some documents were not relevant, they were not related with the query topic) in label equals 1.

Recall@K is always 1 because K is 10 and at least the first 10 rows have punctuation different from 0 and we have 10 labels equals 1.

Average Precision@K is a metric to evaluate each position K where a relevant item is found and then take the average of these Precision values. If the average value is high then systems ranking relevant is more accurate. In our query test, Q2 and Q5 were less accurate because topics' popularity made it more ambiguous to be precise.

F1-score@K evaluates the accuracy of the model taking into account the values of precision and recall. As in general both values are high, F1-score is high as well (this is not the case for Q2).

Mean Average Precision (MAP) basically computes the mean of the average precision for all the queries. As most of the queries have low scores, these values are tiny (close to zero).

Mean Reciprocal Rank (MRR) is the mean of the multiplicative inverse of the rank of the first huge score for each query. We can see that for case Q2 it's ½ = 0.5 and for the rest it's ⅕ = 0.2. This means that for most of the cases, the first huge score is the first entry.

Normalized Discounted Cumulative Gain (NDCG) is in most of the cases a huge value because we are comparing the ranking with an ideal one with several levels of relevance (as before, Q2 is an exception).

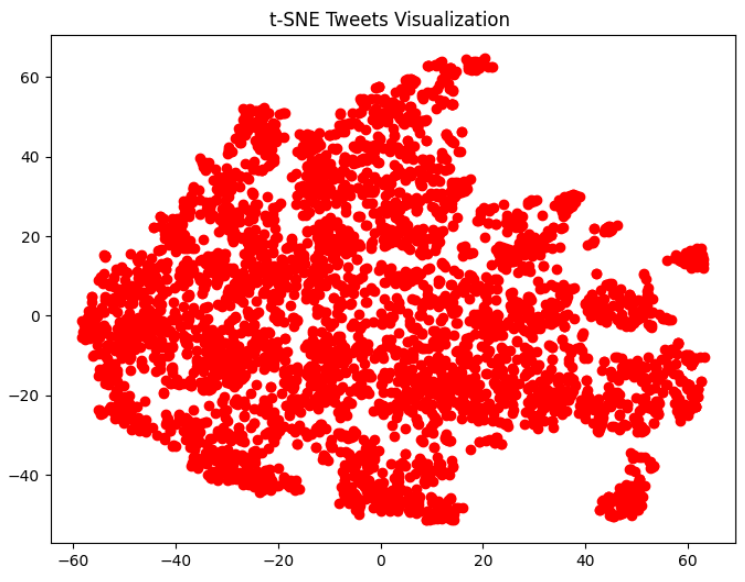

Figure 1: Scatter plot t-SNE Tweets Visualization

The t-SNE visualization of tweets helps us explore the structure and relationships within our tweet data by projecting the high-dimensional data into a 2D space while preserving similarity relationships as much as possible. As we can see in the graph, most of the points are close because the topic is unique (Ukraine-Russian War) and most of the tweets are comments about war. The subgroups that are far away from the center it's because they use non related words about war.

GitHub Repository: JorgeMRPOO/IRWA-2023-part-1 (github.com)