Guillem Gauchia Torres - u186410 - 240215
Àlex Herrero Díaz - u186402 - 240799
Adrià Julià Parada - u188319 - 242195
Date: 28/10/2023

Part 2: Indexing and Evaluation

This project has been carried out by implementing in a notebook the necessary functions and code to process a database of tweets about the current war between Ukraine and Russia. And for the second part it is intended to index the data to create a system of querying and evaluate the results.

IRWA_2023_u186402_u186410_u188319_part_2.ipynb is the Notebook used to carry out the objectives of the practice and to obtain results on the analysis of the queries and evaluations obtained from the results. The code files are located at the following Github Repository:
https://github.com/AlexHerreroDiaz/IRWA-2023/tree/main/IRWA-2023-part-2

Indexing

1. Build inverted index:

After having pre-processed the data, you can then create the inverted index.
Documents information: Since we are dealing with conjunctive queries (AND), each of the returned documents should contain all the words in the query.

To build the inverted index, we use some of the functions that we create in the part-1 to preprocess the documents (***split_hashtag_words***, ***build_terms***, ***get_data_ids***) that we already explained, and then we create the inverted index function, ***create_index_tfidf***, that also compute the tf (normalized term frequency for each term in each document), df (number of documents each term appear in) and idf (inverse document frequency of each term).

```
fatherland: ['doc_572']
firmli: ['doc_575']
mp: ['doc_576', 'doc_723', 'doc_2274', 'doc_3550']
strip: ['doc_576', 'doc_2539', 'doc_2778', 'doc_3346']
unsc: ['doc_576', 'doc_1333', 'doc_2113']
coordin: ['doc_577', 'doc_1390', 'doc_2275', 'doc_2906', 'doc_3210', 'doc_3679']
zaporozhzhia: ['doc_578']
signific: ['doc_579', 'doc_1359', 'doc_2020', 'doc_2407', 'doc_2467', 'doc_3332']
point: ['doc_579', 'doc_1006', 'doc_1007', 'doc_1239', 'doc_1276', 'doc_1365', 'doc_1632', 'doc_1923', 'doc_2004', 'doc_2329', '
seven: ['doc_579', 'doc_1481', 'doc_1690']
deoccupi: ['doc_581']
steal: ['doc_582', 'doc_681', 'doc_1884', 'doc_2182', 'doc_2805', 'doc_3744']
event: ['doc_585', 'doc_783', 'doc_1415', 'doc_1464', 'doc_1634', 'doc_2475', 'doc_3130', 'doc_3131', 'doc_3828', 'doc_3892', 'd
somnia: ['doc_586', 'doc_2474']
spotlight: ['doc_586', 'doc_2474']
cnni: ['doc_587', 'doc_707', 'doc_3142', 'doc_3175', 'doc_3176']
span: ['doc_587', 'doc_601', 'doc_661']
pure: ['doc_587', 'doc_601', 'doc_661']
mata: ['doc_587', 'doc_601', 'doc_661']
fail: ['doc_587', 'doc_601', 'doc_661', 'doc_1919', 'doc_2166', 'doc_2348', 'doc_2601', 'doc_2666', 'doc_2722', 'doc_2896', 'doc
nihtonobodi: ['doc_589', 'doc_608']
sens: ['doc_589', 'doc_2969', 'doc_3042', 'doc_3139', 'doc_3520', 'doc_3816']
guid: ['doc_589', 'doc_3268']
wolf: ['doc_589', 'doc_1684']
calcul: ['doc_590']
sketchi: ['doc_591']
knock: ['doc_592', 'doc_1475', 'doc_2112', 'doc_3636']
ol: ['doc_592', 'doc_1926']
hivka: ['doc_592']
finish: ['doc_593', 'doc_814', 'doc_2092', 'doc_2777', 'doc_3028']
exclud: ['doc_593', 'doc_1485', 'doc_3785']
throughout: ['doc_594', 'doc_3461']
incorpor: ['doc_594', 'doc_597', 'doc_671', 'doc_1617', 'doc_3076']
```

2. Propose test queries:
Define five queries that will be used to evaluate your search engine (e.g., "presidents visiting Kyiv", "countries supporting Ukraine")

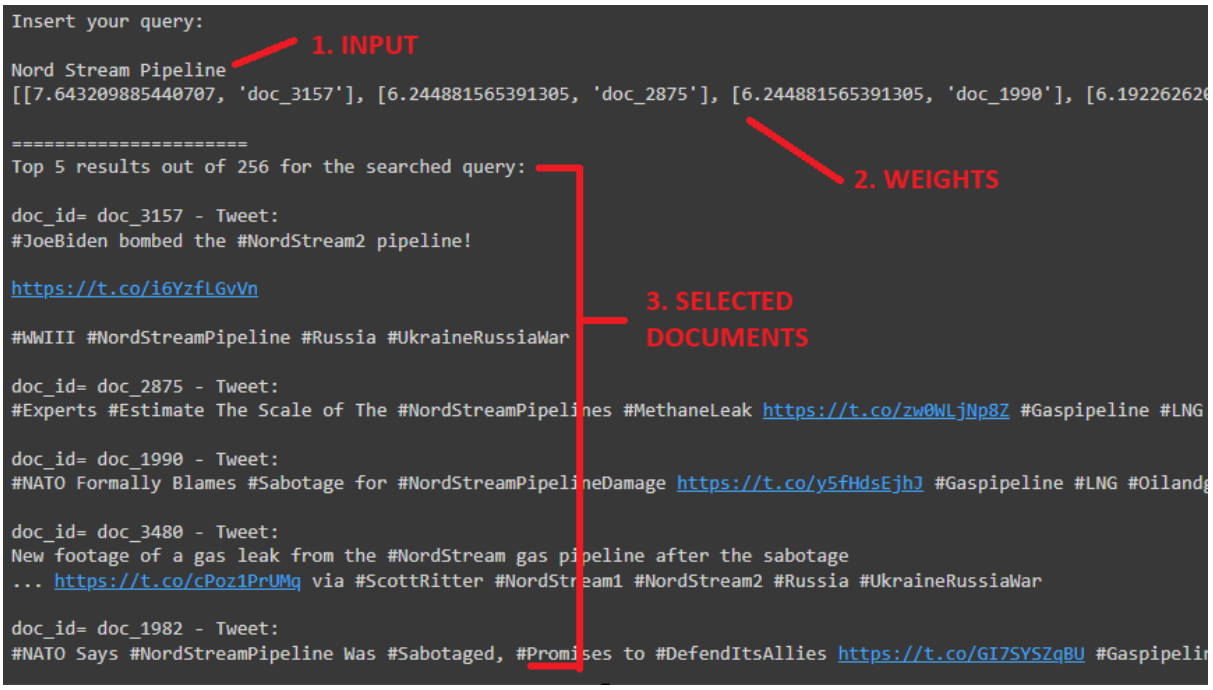
Once the inverted index has been created, we can now create a search system that will select the most relevant documents by means of a phrase according to what has been used as the search phrase.

This are 5 queries that we selected to be evaluated:

1. Presidents visiting Kyiv
2. Countries supporting Ukraine
3. Tanks in Kharkiv
4. Nord Stream Pipeline
5. Annexation of territories by Russia

3. Rank your results:
In order to rank our results, we will implement the TF-IDF algorithm and provide ranking based results in a function called ***search_tf_idf*** that calls a ranking function called ***rank_documents***.

In the Notebook provided we show the selected documents for each of the five topics, in the following image we show an example of how this information is displayed. First we show the list of selected documents and their relevance values, and followed by the Top-N selected documents with the best valuation (in this case N = 5).



From the results observed in the Notebook, we determined that the more frequent the terms used for the query input, the easier it is to find results that match the searched topic.

Evaluation

1. Information needs

The three main queries according to the three questions suggested to answer that we treated were the following:

- **Question/Query 1:** “Tanks in Kharkiv”
- **Question/Query 2:** “Nord Stream pipeline”
- **Question/Query 3:** “Annexation of territories by Russia”

2. Setting the ground truth

You will be the expert judges, so you will be setting the ground truth for each document and query in a binary way for the test queries that you defined in step 2 at the indexing stage.

For this part we have used the ***search_tf_idf*** function to determine the relevance weight of each document that is part of the ***evaluation_gt*** csv, to achieve our goal we have created 6 new columns, 3 to map the corresponding ground truth of each query and 3 others to assign the weights that represent how relevant each document is for each query and thus to evaluate how efficient the implemented search system is with respect to the ground truth.

The following image shows the format that follows the ***label_relevance*** data frame created to carry out the evaluation part:

	doc_id	is_relevant_Q1	is_relevant_Q2	is_relevant_Q3	predicted_relevance_Q1	predicted_relevance_Q2	predicted_relevance_Q3
0	doc_1452	0	1	1	0.0	4.945905	0.000018
1	doc_2908	0	1	1	0.0	5.113889	0.000019
2	doc_618	0	1	1	0.0	4.794443	0.000053
3	doc_489	0	1	1	0.0	2.988094	0.000017
4	doc_110	0	1	1	0.0	3.273101	0.000019
5	doc_3439	0	1	1	0.0	2.403367	0.000026
6	doc_3137	0	1	1	0.0	2.799881	0.000016
7	doc_3913	0	1	1	0.0	2.029107	0.000035
8	doc_2944	0	0	1	0.0	2.534592	0.000029
9	doc_2696	0	1	1	0.0	2.907431	0.000017

Something important to note is that for question 3 the values are proportionally lower, which indicates that the documents selected for the third search are not as relevant as they are in the other queries. And in the case of Q1 we have several null values that have been assigned with 0.0, this is due to the fact that of the selected documents these do not have any type of relation with the query in question.

3. Evaluation techniques

For the prior evaluation components you must evaluate your algorithm by using different evaluation techniques:

To analyze and evaluate how efficient our search system is, we carry out the following metrics that allow us to monitor performance according to the results:

3.1. Precision@K (P@K)

==> Precision@5: 0.8 for Question 1

Check on the dataset sorted by score:

	doc_id	is_relevant_Q1	is_relevant_Q2	is_relevant_Q3	predicted_relevance_Q1	predicted_relevance_Q2	predicted_relevance_Q3
31	doc_2234	1	0	0	4.684973	0.0	0.000026
32	doc_2656	1	0	0	3.783179	0.0	0.000021
37	doc_2701	1	0	0	3.348105	0.0	0.000025
40	doc_1336	0	0	0	2.854752	0.0	0.000031
34	doc_592	1	0	0	2.156594	0.0	0.000048

==> Precision@5: 1.0 for Question 2

Check on the dataset sorted by score:

	doc_id	is_relevant_Q1	is_relevant_Q2	is_relevant_Q3	predicted_relevance_Q1	predicted_relevance_Q2	predicted_relevance_Q3
1	doc_2908	0	1	1	0.0	5.113889	0.000019
0	doc_1452	0	1	1	0.0	4.945905	0.000018
2	doc_618	0	1	1	0.0	4.794443	0.000053
4	doc_110	0	1	1	0.0	3.273101	0.000019
20	doc_2677	0	1	0	0.0	3.029321	0.000035

==> Precision@5: 0.2 for Question 3

Check on the dataset sorted by score:

	doc_id	is_relevant_Q1	is_relevant_Q2	is_relevant_Q3	predicted_relevance_Q1	predicted_relevance_Q2	predicted_relevance_Q3
17	doc_3422	0	0	0	0.000000	0.000000	0.000059
41	doc_2295	0	0	0	1.700542	0.000000	0.000055
2	doc_618	0	1	1	0.000000	4.794443	0.000053
22	doc_3326	0	0	0	0.000000	0.000000	0.000053
38	doc_2956	1	0	0	0.703644	0.000000	0.000050

$$\text{Precision} = \frac{TP}{TP + FP}$$

To calculate the precision@k we have created a function ***precision_at_k***, which calculates the precision taking into account only the first k documents, these being the k documents with a larger predicted value of having relevance to the selected query. Therefore we assume that all these documents have been predicted to be relevant with that query, and therefore to calculate the precision we only have to add the number of documents that the ground truth of these documents among k.

Here we can see that in the first two queries, the precision is very high and that means that most of the potential documents that are expected to be relevant, they actually are. Eventhough, in the third query we don't get good results.

3.2. Recall@K (R@K)

```
==> Recall@5: 1.0 for Question 1
==> Recall@5: 1.0 for Question 2
==> Recall@5: 1.0 for Question 3
```

$$\text{Recall} = \frac{TP}{TP + FN}$$

To compute the recall@k we have created a function called ***recall_at_k***, that computes the recall taking into account only the first k documents with the k highest prediction values of having relevance on the selected query.
As we can see in the results, the recall is 1 for all questions, this is because the first k documents are considered relevant because they have the highest predicted relevance, and therefore there are no fn cases.

3.3. Average Precision@K (P@K)

```
==> Average Precision@10: 0.716547619047619 for Question 1
==> Average Precision@10: 0.89 for Question 2
==> Average Precision@10: 0.025 for Question 3
```

$$AP@K = \frac{1}{GTP} \sum_{k}^{n} P@K \times rel@K$$

To compute the average_precision@k we have created a function ***average_precision_at_k***, which calculates the average precision on k documents.
As we can see in the results both question 1 and question 2 have a quite high average precision unlike question 3.

3.4. F1-Score@K

```
==> F1-Score@5: 0.888888888888889 for Question 1
==> F1-Score@5: 1.0 for Question 2
==> F1-Score@5: 0.33333333333333337 for Question 3
```

$$F_1\text{−score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

To compute the f1_score@k we have created a function called ***f1_score_at_k*** that computes the F1 Score taking into account only the first k documents with the k highest prediction values of having relevance on the selected query. This specific statistic calculates the harmonic mean of the precision and recall scores.
As we can see in the results both queries 1 and 2 have a high F1 Score. In the other way, query 3 is just a 0.33 F1 Score indicating that it is hard to give really relevant documents.

3.5. Mean Average Precision (MAP)

```
==> Mean Average Precision@5: 0.6694204291172254 for All 3 Questions
```

$$mAP = \frac{1}{N}\sum_{i=1}^{n} AP_i$$

To compute mean_average_precision@k, we have created a function called ***map_at_k*** that computes the mean of all the queries average precision.

As we can see in the result, the map is slightly higher than sixty percent for the three questions.

3.6. Mean Reciprocal Rank (MRR)

```
==> Mean Reciprocal Rank (MRR) for k = [3, 5, 10]: {3: 0.6667, 5: 0.75, 10: 0.75} for All 3 Questions
```

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}$$

To compute mean_reciprocal_rank@k, we have created a function called ***rr_at_k*** to compute the reciprocal rank at the first k documents with the k highest prediction values of having relevance on all the queries together. Once we have computed this, we did the mean reciprocal rank for k = 3,5,10.

3.7. Normalized Discounted Cumulative Gain (NDCG)

```
==> NDCG@5: 0.8539 for Question 1
==> NDCG@5: 1.0 for Question 2
==> NDCG@5: 0.1461 for Question 3
```

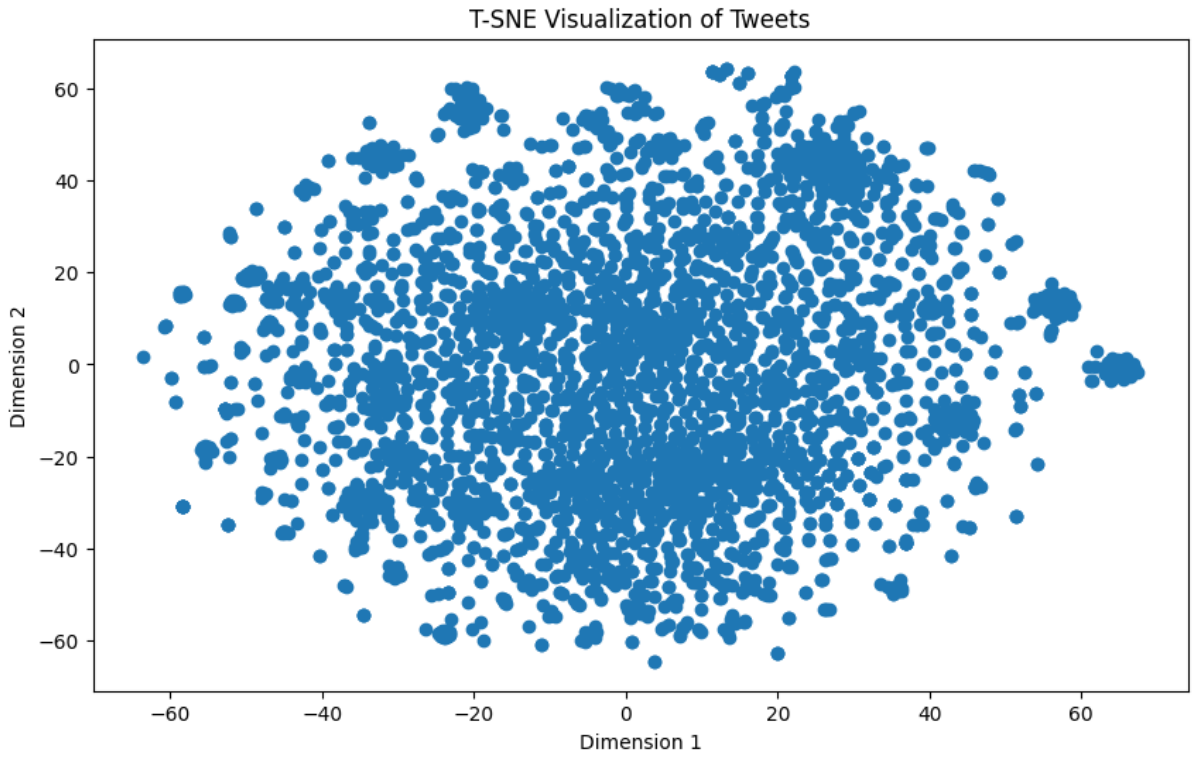
$$\mathrm{nDCG}_k = \frac{\mathrm{DCG}_k}{\mathrm{IDCG}_k}$$

$$\mathrm{DCG}_k = \sum_{i=1}^{k}\frac{\mathrm{rel}_i}{\log_2(i+1)} \qquad \mathrm{IDCG}_k = \sum_{i=1}^{k}\frac{\mathrm{rel}_i}{\log_2(i+1)}$$

To compute the normalized_discounted_cumulative_gain@k, we have created a function called ***ndcg_at_k*** that computes this statistic using another function called ***dcg_at_k***. This normalized discounted cumulative gain is computed at the first k documents with the k highest prediction values of having relevance on each of the three queries.

The normalized discounted cumulative gain at k=5 gave us that the queries 1 and 2 have quite bigger values than the query 3.

3.8. T-SNE Scatter Plot



As we can see from the T-SNE plot, all the points are very close together, we can deduce that it is because all the documents are related to each other, as they are all tweets about the Russia-Ukraine war. Also something interesting to note at the screenshot is that there are several clusters of data points representing the several topics that are discussed from the entire set of tweets and that represents how similar are the tweets near groups of tweets around them.