Estellers, Oriol - 242142
Fuentes, Raimon - 242176
Ribas, Pol - 241620
Github link and TAG: IRWA-2023-part-2

Part 2: Indexing and Evaluation

This Part 2 project is based on the implementation of Part 1, since we will work with the pre-processed tweets. Therefore, we will use the code that we had already done, which refers to both the pre-processing and the exploratory analysis. Nevertheless, there is one aspect that in the previous part worked perfectly, but for this project it will no longer be correct, and it is the way in which we store the pre-processed tweets. In Part 1, we stored the pre-processed tweets in a new column of the dataframe in string format, but for this project we have changed this and now the pre-processed tweets are stored in the column as one array where each position refers to one word of the pre-processed tweet. Moreover, we now also remove the @ and # of the mentions and hashtags, so these words are also taken into consideration not only when creating the index, but also when querying it.

Now that we have everything settled up, we will begin by the first section, which is the implementation of the **Indexing**. To do so, we will implement a function so that each of the words in the dataset has an associated list with all the documents that contain this word. For this, we have adapted the function *create_index(lines)* from the second lab session, as we no longer have a 'title' and 'body_text', we now just have a kind of 'body_text' (the column 'full_text' of the dataframe), but the core of the function is the same. When creating the index, we will use the tweet id as the unique key for each element of the index, and the elements will be the processed tweets.

Once each word is mapped to the documents, the next step is to implement a function that receives queries as inputs and returns the tweets related to the query. Hence, we have defined the *search(query, index)* function, which given a query (a string), searches in the index the tweets that coincide with the query for, at least, one term. Therefore, if for example inputting a sentence of 3 words, this function will return all tweets containing at least one of these 3 words.

Until this point we are able to search among all tweets given a query, so the next step is to sort the obtained results by relevance or following some criteria, so that's why we will use the TF-IDF weighting scheme to obtain a list of ordered results. Firstly, we reused the function *create_index_tfidf(lines, num_documents)*, where lines are all tweets id and full_text columns, and num_documents is the number of tweets we have. What this function does is assign a weight to each term in a document based on its term frequency (TF) and inverse document frequency (IDF). The higher the score, the more important the term is. Afterwards, we create the *rank_documents(...)* function, which will allow us to perform the ranking of the results of a search based on the TF-IDF weights. Finally, to search in this new ranked-index method, we have implemented the *search_tf_idf(query, index)* function as well, where the objective is the same one as before (given a query, return the tweets related to it) but this time using the TF-IDF method in ranking. The tweets returned after using this searching method should be more accurate than before, since it should output the tweets that are more related to the query.

In order to evaluate our new searching engine, we will use the following queries:

- Drone attacks to Ukraine
- Crimea bridge explosion
- Ukraine joining NATO
- Pentagon provides 18 HIMARS
- Mykolaiv explosion bus station

These queries that we propose were not randomly chosen. In fact, we have looked up key events that happened in the Ukraine vs. Russia war and that were not very generic (ex: Russia attacks Ukraine). Therefore, these events are important enough to appear in some of the tweets of the dataset, but not so generic so that the majority of the tweets are related to them.

Related to the next section, the relevant tweets for each of these queries can be found at the attached .csv file. The irrelevant documents for each query are randomly chosen through a script in the notebook. We have chosen these relevant documents by looking at the first top-10 relevant documents that the search engine of the non-ranked index returns when inputting the queries, queries that are not processed. We have decided to do it this way because we ensure that we will have different documents when querying the ranked index. If we did not do it this way, when evaluating the search engine with the different techniques, we would mostly obtain ones, as the search engine would be returning the same documents as before. In other words, the documents that we would have chosen as ground truths would be exactly the same as the documents that the search engine returns when querying it, because both queries would be exactly the same.

The second part of the project is based on **Evaluating** the searching engine that we have implemented. This evaluation will be done in two parts: first, we will use the 3 information needs and the ground truth files for each query that were provided to us and, in the second part, we will judge ourselves how good the algorithm has performed with the 5 queries suggested above by selecting which are the relevant documents for each of our five queries. The evaluation techniques that we will follow are based on the following metrics:

→ *Precision at k*: As we saw in the labs, the precision at k is the fraction of the retrieved documents which is relevant to the searched query. To implement this method of evaluation we have adapted the function that we created in lab 2 to our project. First of all we sort the predicted scores, then we take the predicted relevant documents that are also in the ground truth list, and then we divide how many of them we have by k.

→ *Recall at k*: The recall is the fraction of the relevant documents which has been retrieved. To implement the recall at k, we have done the same as precision at k, but dividing the number of predicted documents that were in the ground truth list by the total number of relevant tweets.

→ *Average precision at k:* In this evaluation method we take into account the order of the most and least relevant tweets of the relevant documents.
To implement this evaluation method we have first calculated the number of relevant documents that there are, and then sorted our prediction. Then take the documents that we predicted to be relevant in the ground truth list, and we iterate through them to apply the formula.

→ *F1 at k*: The F1 at k method combines the recall and precision into a single score. To implement this evaluation method we have simply applied the formula given that we already had the precision and recall functions implemented. The F1 at k is calculated in the following way: $2 \cdot (\text{Precision} \cdot \text{Recall}) / (\text{Precision} + \text{Recall})$

→ *Mean average precision*: For this part we have done the mean between the average precision of the queries that we are evaluating.

→ *Mean reciprocal rank*: In this part we have also done the mean of the reciprocal rank between the queries that we are evaluating. The reciprocal rank method consists of evaluating the position of the first correct answer: the reciprocal rank value for any query is 1 divided by the position of the first correct answer, as long as the position isn't greater than k. If it is, the value will be 0.
The mean reciprocal rank is the average of the reciprocal rank values for all queries.

→ *Normalized Discounted Cumulative Gain (NDCG)*: For this evaluation metric, we first need to introduce Discounted Cumulative Gain (DCG), which is based on the notion of Cumulative Gain (CG): it is the sum of the relevance values of all results in the search result list. From this point, DCG not only takes into consideration the relevance value, but also the documents' position. One way to do so is by rewarding those relevant results that appear higher in the result set or, what is the same, discounting results that appear lower. To this end, in DCG relevance values are reduced logarithmically proportionally to the position of the document.
NDCG is the normalization of the DCG scores when analyzing different queries, as DCG values can significantly differ between queries. This is done by sorting all relevant documents in the corpus by their relative relevance, producing the maximum possible DCG through position n (also called ideal DCG - IDCG) through that position (usually the ground truth).

For the evaluation of the information needs provided to us, we will proceed as follows: we were given for each of the 3 queries 10 documents that were relevant to the query (1) and 10 that were not relevant to the query (0). Hence, the dataset contains 60 rows, 20 for each query, and 3 columns: one for the document id, one for the query id (Q1/Q2/Q3) and the last one for the label (1/0). What we have initially done is modify this dataset by adding 6 columns: predicted scores (3 cols) and labels (3 cols) for each query, so every document will now have a predicted score and a label for each of the three queries. We fill in the dataset with the information before analyzing the search engine with the metrics defined above. Another important point to note is that, again, the queries that we use are written in natural language (i.e. with stop words and not in the word root form), but before searching them are processed, as this way it will be easier to match the indexes in the search engine.

Given our five queries and the evaluation techniques, the results obtained are the following:

→ ***Query 1:*** *Drone attacks to Ukraine*

```
Precision for Q1: 0.8
Recall for Q1: 1.0
F1 at k for Q1: 0.888888888888889
Average precision for Q1: 0.7434920634920634
Reciprocal rank for Q1: 1.0
Normalized Discounted Cumulative Gain for Q1: 0.8522
```

We can observe that the precision in our prediction is pretty high. The recall is 1, since we expect 10 documents as output and that is what we get in our prediction.
We can see that our F1 value is higher than the precision since it combines it with the recall. The average precision is not very high, even though it is not bad either. Afterwards, we can say that the reciprocal rank in this query is very high since it is one, which means that we get the first relevant document very fast. Finally we can see that the NDCG is 0.85

→ ***Query 2:*** *Crimea bridge explosion*

```
Precision for Q2: 1.0
Recall for Q2: 1.0
F1 at k for Q2: 1.0
Average precision for Q2: 1.0
Reciprocal rank for Q2: 1.0
Normalized Discounted Cumulative Gain for Q2: 1.0
```

Our precision is very high. The recall is 1, which is the most common value that we are getting since we are expecting 10 predictions. Our F1 value is 1 since the recall and precision are also 1. The average precision is very high, which means that we have predicted the order correctly. Moreover, we can see that the reciprocal rank and NDCG are also 1.

→ ***Query 3:*** *Ukraine joining NATO*

```
Precision for Q3: 0.9
Recall for Q3: 1.0
F1 at k for Q3: 0.9473684210526316
Average precision for Q3: 0.9
Reciprocal rank for Q3: 1.0
Normalized Discounted Cumulative Gain for Q3: 0.9364
```

For this query the precision is 0.9, which means that we have correctly predicted 9 out of the 10 relevant documents. The recall is also 1, as in the cases before. The F1 score is very high since the recall and precision are also high. The average precision indicates that we have predicted the order of the relevant documents almost perfectly, since it is 0.9. Finally the Reciprocal rank and NDCG are extremely high.

→ ***Query 4:*** *Pentagon provides 18 HIMARS*

```
Precision for Q4: 1.0
Recall for Q4: 1.0
F1 at k for Q4: 1.0
Average precision for Q4: 1.0
Reciprocal rank for Q4: 1.0
Normalized Discounted Cumulative Gain for Q4: 1.0
```

We can see that the values for the precision recall and F1 are 1, this also causes that the average precision is 1 and the reciprocal rank and NDCG are 1 as well. This may happen because the query is very specific, and there may not be many other tweets apart from the relevant ones that have the words of the query.

→ ***Query 5:*** *Mykolaiv explosion bus station*

```
Precision for Q5: 0.9
Recall for Q5: 1.0
F1 at k for Q5: 0.9473684210526316
Average precision for Q5: 0.8354365079365079
Reciprocal rank for Q5: 1.0
Normalized Discounted Cumulative Gain for Q5: 0.9149
```

For the fifth query we have very good results as well. If we look at the precision we can see that it is almost perfect. The recall is 1, just like in the other queries. The F1 score is very high as well since it combines both the recall and the precision. The average precision is pretty good, even though we have not completely predicted the

order correctly. Finally the reciprocal rank and the NDCG are pretty high, being 1 and 0.91 respectively.

→ **Mean results:**

```
The mean averagre precision is (0.8957857142857142, [0.7434920634920634, 1.0, 0.9, 1.0, 0.8354365079365079])
The mean reciprocal rank is (1.0, [1.0, 1.0, 1.0, 1.0, 1.0])
The mean normalized discounted cumulative gain is (0.9407, [0.8522, 1.0, 0.9364, 1.0, 0.9149])
```

The mean average precision is 0.89, which indicates that our predictions are very accurate, not only in terms of guessing the 10 most relevant documents, but also in guessing the order of relevancy.

In second place we can see that the reciprocal rank has been 1 in all of our queries, which indicates that in all of them the first tweet returned is relevant. For a search engine this is very important, as a significant amount of times users just look at the first entry returned.

Finally the mean of the normalized discounted cumulative gain is 0.94, which is a very high value. This tells us that, for every query, the NDCG value is very high, which leads us to believe that the prediction scores of the tweets returned are the highest possible ones.

As a sum up, we can see that our algorithm performs quite well, so most of the time the first documents that it returns are the most relevant ones.

For the vector representation, we have chosen the Word2Vec model to do a one vector representation of the tweets. First, we have converted all the words into a 2-dimensional vector, hence having x and y coordinates each of them. The main objective that we wanted to achieve is to visualize the tweets in 2d in order to observe the clusters formed. Therefore, we have decided to convert the tweets labeled as 1 for our 5 proposed queries. Since we already had the coordinates in 2-dimensions of each word, we have computed the coordinates of the tweets by doing the average of the coordinates of the words from each tweet. With this, we have successfully been able to represent the tweets in the x and y axis.

Even though we could represent the tweets in a 2d form, the plot is not done with TNSE because we did not find the way to implement it. As we can see from the plot, the relevant tweets from the same query do not seem to follow any kind of tendency, which makes us think that it is not completely correct, since we think that the tweets from the same queries should be grouped together in a kind of cluster.