Ingrid Sancho Aventin - 218033
Vladimir Trukhaev - 219328
Information Retrieval and Web Analytics

Universitat Pompeu Fabra Barcelona

Final Project Part 3: Ranking

Introduction

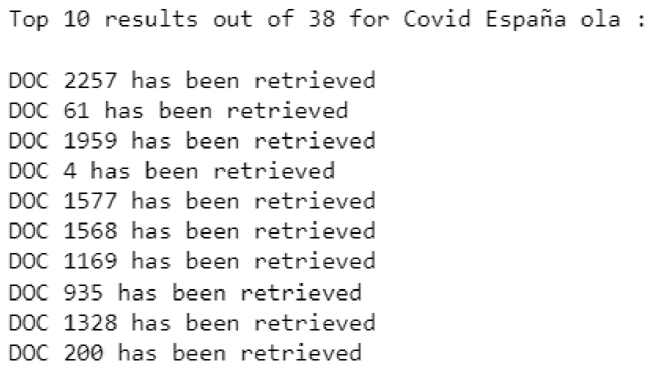
This third part of the final project consisted in Ranking our corpora from the World Health Organization Twitter account (*dataset_tweets_WHO.txt*). The goal of this part was to find all the tweets that contain all the words of our query and sort them by relevance with regard to such a query. At the end we want to get the top-20 documents related to the query. As a submission we have this PDF file (*IRWA-2021-final-project-U162442-U161809-part-3*) and the following GitHub repository where you can find the code we have used to accomplish all the tasks required for the submission: https://github.com/vladimirtrukhaev/IRWA_Project. In said GitHub repository you can find a branch called *"IRWA-2021-final-project-part-2"* (https://github.com/vladimirtrukhaev/IRWA_Project/tree/IRWA-2021-final-project-part-3) where the code for this part of the project, this PDF file as well as the provided text file are located.

1. Score

a) TF-IDF + cosine similarity

Like we explained in the previous part of this final project TF-IDF basically works by assigning each term in the document (in this case it is a list of Tweets) a weight based on its term frequency and the inverse document frequency. Meaning the higher the score, the more relevant a Tweet is to our search.

We can then obtain the cosine similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. The cosine similarity of two documents will range from 0 to 1, since the term frequencies (TF-IDF weights) cannot be negative. As well as the angle between two term frequency vectors cannot be greater than 90. In *Figure 1* we have the list of the retrieved documents for the query "Covid España ola" with such a rank.



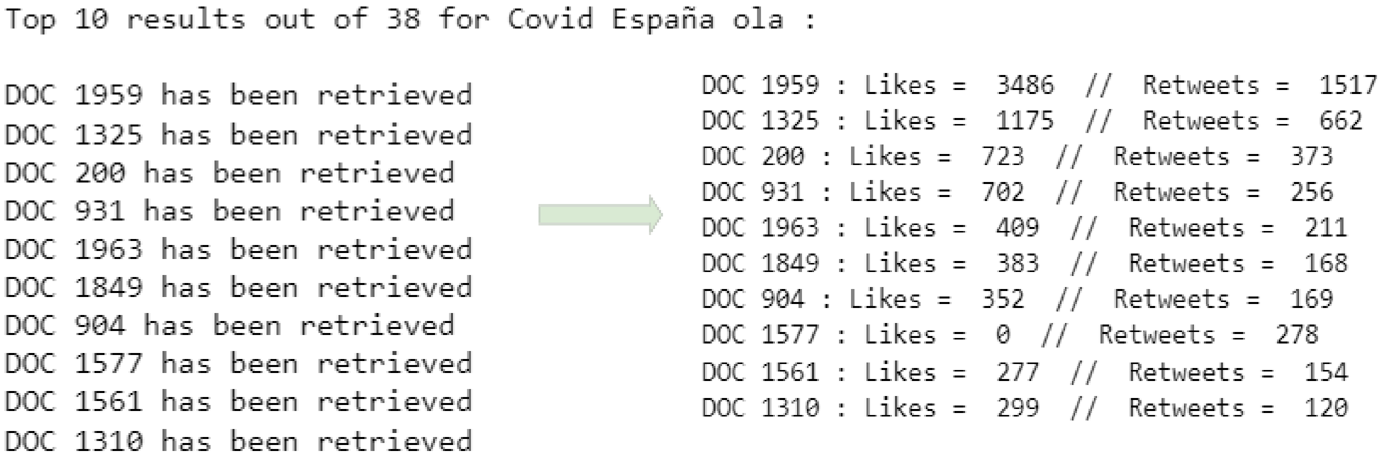
```
Top 10 results out of 38 for Covid España ola :

DOC 2257 has been retrieved
DOC 61 has been retrieved
DOC 1959 has been retrieved
DOC 4 has been retrieved
DOC 1577 has been retrieved
DOC 1568 has been retrieved
DOC 1169 has been retrieved
DOC 935 has been retrieved
DOC 1328 has been retrieved
DOC 200 has been retrieved
```

Fig. 1. Top 10 results using the TF-IDF Cosine Similarity.

b) “Our-Score” + cosine similarity

We based our approach on the popularity of the tweet. In other words, we took the likes and retweet values of each doc and computed a normalized factor for each of the tweets terms. We stored his score for each term in a dictionary of dictionaries called “tweet_score”. Then, as asked, we computed the cosine similarity after computing the query and doc Vectors, using the previous factor. By taking this approach we will only focus on the repercussion of the tweet containing a given query. The major change is that it depends on the users interactions and not on the df, tf or idf factors. This implementation takes considerably less time to run, the index is created in about 0.5 secondsHere we have the list of the retrieved documents for query “Covid España ola” with our rank:



```
Top 10 results out of 38 for Covid España ola :

DOC 1959 has been retrieved
DOC 1325 has been retrieved
DOC 200 has been retrieved
DOC 931 has been retrieved
DOC 1963 has been retrieved
DOC 1849 has been retrieved
DOC 904 has been retrieved
DOC 1577 has been retrieved
DOC 1561 has been retrieved
DOC 1310 has been retrieved
```

```
DOC 1959 : Likes =  3486  //  Retweets =  1517
DOC 1325 : Likes =  1175  //  Retweets =  662
DOC 200 : Likes =  723  //  Retweets =  373
DOC 931 : Likes =  702  //  Retweets =  256
DOC 1963 : Likes =  409  //  Retweets =  211
DOC 1849 : Likes =  383  //  Retweets =  168
DOC 904 : Likes =  352  //  Retweets =  169
DOC 1577 : Likes =  0  //  Retweets =  278
DOC 1561 : Likes =  277  //  Retweets =  154
DOC 1310 : Likes =  299  //  Retweets =  120
```

Fig. 2 and 3. Top 10 results using Our Score with Cosine Similarity and the amount of likes and retweets per Tweet.

We consider this approach interesting since it gives full credit to the interactions of other users. We rely on the fact that a tweet is relevant if others found it relevant and we also assume that users “like” or “retweet” with a critical sense and it is not random. It would be interesting to combine both approaches, the tf-idf and the “popularity” score, in order to have a more balanced and accurate answer.

2. Tweet2Vec + cosine similarity for returning top 20 tweets

For this part we had to do some research regarding the implementation of a word2vec model and we had to carefully think how we are going to handle this exercise. We firstly implemented the word2vec environment with the word2vec.model and our dictionary of tokens (my_dict) as an input. We ended up with a model for which each of the words had a 100-dimension vector.

Then we had to compute the tweet vector representation: for this, we implemented a function called “doc_2_vec(my_dict)”. This function iterates through my_dict and, for each document (tweet), takes all the vector representation of the terms and averages them. Then it stores such a vector as a value in a dictionary (tweet2vec) whose keys are the tweets’ ids. At the end we have a dictionary with a unique vector representation (also of 100 dimensions) for each tweet.

We also implemented here a similar function called “query_2_vec(query)” that does a similar task, but just for a given query: it takes all the vectors of all the words in the query and averages them in order to obtain a unique representation of our input query.

Having implemented the “to vec” functions we needed to create a function that would compute the similarity and return the 20-top documents: we called this function “top_20(query)”. What this function does is take the given query and process it (by calling build_terms) and then it represents the query as a vector (by using query_2_vec). After this, we compute the similarity by calling a previously implemented function (it calculates similarity of non normalized vectors) and we end up with a list of (similarity, doc) for all tweets. Then we sort it in descending order so that the most similar tweets come first. Lastly, for the top-20 ones, we print the information as we were asked.

3. Better representation?

Word2Vec creates a distributed semantic representation of words. Using two different ways to deal with training: the Continuous Bag of Words and The Skip Gram Model. Where the first one includes foreseeing the context words utilizing a middle word, whereas the other one includes anticipating the word utilizing said context words.

This idea can then be applied to both sentences and documents making the other two models (Sentence2Vec and Doc2Vec) possible. Meaning that in Sentence2Vec and Doc2Vec instead of creating the aforementioned semantic representation of words you do it instead of sentences and documents respectively.

Having said that, Word2Vec is commonly used when we want to calculate word similarities, Sentence2Vec when we are trying to capture the similarity between sentences and finally Doc2Vec for documents.

In this case we think that Sentence2Vec would be a better representation to deal with this type of problem. We think so because, even though the Word2Vec model takes into consideration the context, for the purpose of tweets it might not achieve our goal and might return not so relevant documents (we deduct this from our results in exercise 2, since some of the top-20 tweets did not seem relevant to our queries). Also, as mentioned earlier, Sentence2Vec creates a distributed semantic representation of sentences instead of words. Meaning that it will take the context of sentences as a whole thus improving our ranking function.