Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

REPORT PART 3

Ranking

Github LINK and TAG: IRWA-2024-part-3

This second part of the project is based on the implementation of the first and second part, since we again use the processed tweets and the created tokens and, also, we use part of the process to compute the tf-idf for a given query and retrieve its ranked documents. Therefore, we will again use the code that is in the **Text Processing** and **Inverted Index** sections. After the feedback we received from the second part of the project, we have improved the function that computes the tf-idf because we didn't take into account the cosine similarity. So, we created a new function that does it. Also, we have used the dictionary created in this section *clean_tweets* to be able to work with the tokens in a simpler and easier way, the inverted index created and the `search` function to retrieve the common documents. Then, we proceeded to create the 3 different ways of ranking asked and, then, returned a top-20 list of documents for each of the 5 queries, using word2vec + cosine similarity.

The first part of this project was to provide three different ways of ranking, explain them and compare.

TF-IDF + cosine similarity

In this part we have defined the function `def tfidf_cosine(query, inverted_index, total_tweets, clean_tweets)` which is the one where we implemented the improvements needed after the feedback. This function computes the **TF-IDF+cosine similarity** based on the function created in the second part of the project but including the cosine similarity scores to be able to rank the documents more accurately as we are now computing the similarity between the document and query vectors.

First, it computes the tf-idf for the query terms, that is, for each term in the query we compute its TF (which will be 1 in most cases as each word will appear 1 time in the query) and its IDF. Having this, we can add t

he tf-idf score of the query term to the query vector. Then, we compute the document vector with the tf-idf score.

When we have both vectors, we can proceed to compute the cosine similarity. For each document, we compute the dot product between the query vector and the document vector.

Finally, we sort the documents and we can observe that for a given query, for example, 'indian government' we get the following results:

Note that the output of the code is the top 20 tweets, but, for simplicity in this report we include the top 10.

Top 1 tweet: tweet_id = doc_17769 - score = 1.9905568825
Content: Shame on Indian government
#FarmersProtest https://t.co/oW61P13nXH

Top 2 tweet: tweet_id = doc_11484 - score = 1.9905568825
Content: Shame on Indian government #FarmersProtest https://t.co/khyXi7pNAy

Top 3 tweet: tweet_id = doc_29545 - score = 1.9905568825
Content: #FarmersProtest What a shame on Indian government. https://t.co/WgtT04W6ur

Top 4 tweet: tweet_id = doc_38240 - score = 1.9905568825
Content: Shame on Indian Government 😔
#FarmersProtest https://t.co/XuLV671Gof

Top 5 tweet: tweet_id = doc_31188 - score = 1.9905568825
Content: The Indian government is facist #FarmersProtest

Top 6 tweet: tweet_id = doc_34728 - score = 1.592445506
Content: So disgusted by Indian government now !
#FarmersProtest #FreeDishaRavi

Top 7 tweet: tweet_id = doc_31710 - score = 1.592445506
Content: #IamAgainstModiGovt
#farmersprotest
Shame on indian government https://t.co/RMwUNfYMV1

Top 8 tweet: tweet_id = doc_42096 - score = 1.592445506
Content: I am appalled by the ignorance of the Indian government. #FarmersProtest

Top 9 tweet: tweet_id = doc_14237 - score = 1.592445506
Content: Shame on Indian government #FarmersProtest #DPstopIntimidatingFarmers
https://t.co/9OftsBizhY

Top 10 tweet: tweet_id = doc_18946 - score = 1.592445506
Content: Indian government has to do better! Seriously. #FarmersProtest

Our score + cosine similarity

Now, in this part, we have to create our own score for the classification of the documents. Our main idea is to use some properties of the tweets that we are not taking into account in any of the other tasks. After observing the information given in each tweet, our conclusion was to use information like the number of likes, the number of retweets, the number of followers and if the user is verified or not.

- Number of likes and retweets: Among the relevant documents, the tweets with more likes and retweets should be considered more popular and, therefore, more influential. Therefore, we decided to add a score to those tweets having more likes and amount of

retweets. However, as we know how Twitter works, we think that it is more likely to give a like than a retweet, so we have given more importance to retweets as they are less likely than likes.

- Verification and number of followers: We wanted to take into account the condition of verification. Imagine that two tweets have the same number of likes and retweets and one of the users is verified and the other isn't. Then, we wanted to give more score to the user that is not verified as it is not as likely to get those numbers as the verified user. So, if the account is verified, a higher follower count slightly reduces the score, as the tweet might be popular because of the author's influence rather than content. If the account is not verified, the score increases slightly with a higher follower count, implying trustworthiness.

Our first step to compute this scores is the tf-idf as in the previous part, with document vectors and query vectors, Then, after computing the dot product we have retrieved the properties that we want to take into account and we have normalized the numerical values with a min max scaling because the different variables may have very different range of numbers. Then, we used the verification as an if statement as explained above and we have added the respective score as a proportion. The dot product accounts for 60% and the added score is 40%. Inside this last, we have the number of likes multiplied by 0.4, the number of retweets by 0.6 and the number of followers by 0.3. With this, we wanted to give the highest importance of this new score to the number of retweets.

With this score and the query 'indian government' we get the following results

Note that the output of the code is the top 20 tweets, but, for simplicity in this report we include the top 10.

Top 1 tweet: tweet_id = doc_11484 - score = 1.1957303021054737
Content: Shame on Indian government #FarmersProtest https://t.co/khyXi7pNAy

Top 2 tweet: tweet_id = doc_31188 - score = 1.1948877505597453
Content: The Indian government is facist #FarmersProtest

Top 3 tweet: tweet_id = doc_38240 - score = 1.1948839099054072
Content: Shame on Indian Government 😔
#FarmersProtest https://t.co/XuLV671Gof

Top 4 tweet: tweet_id = doc_17769 - score = 1.1948838188226956
Content: Shame on Indian government
#FarmersProtest https://t.co/oW61P13nXH

Top 5 tweet: tweet_id = doc_29545 - score = 1.194334296484971
Content: #FarmersProtest What a shame on Indian government. https://t.co/WgtT04W6ur

Top 6 tweet: tweet_id = doc_40124 - score = 0.9563138779654897
Content: Shame on Indian government #FarmersProtest #GoBackModi https://t.co/97vElTIDkY

Top 7 tweet: tweet_id = doc_14237 - score = 0.9562014646135094
Content: Shame on Indian government #FarmersProtest #DPstopIntimidatingFarmers https://t.co/9OftsBizhY

Top 8 tweet: tweet_id = doc_30422 - score = 0.9559336620471013
Content: Why are Indian farmers protesting against the government? #FarmersProtest https://t.co/eMUGoXtabZ

Top 9 tweet: tweet_id = doc_40125 - score = 0.9557642493646018
Content: Shame on Indian government #GoBackModi #FarmersProtest https://t.co/md7soP8JOL

Top 10 tweet: tweet_id = doc_36211 - score = 0.9557540632813569
Content: #FarmersProtest look at all the bs the Indian government is doing. https://t.co/7KOggZw8pu

Compared with the previous result, we see some new documents in the list as the first one 'doc_11484' and others that have changed their position in the ranking. This tells us that there might be some documents with lower tf-idf scores, but appear to be more popular and, therefore, their score in this case is increased thanks to that. However, we can also observe that the documents retrieved, besides being more popular, are also relevant to the query.

Our scoring method is not perfect and has some pros and cons, which are the following:

- **Pros**
  - The balance between the cosine similarity and the popularity of the tweet ensures that high-engagement tweets aren't always prioritized unless they're also relevant to the query.
  - By giving higher weights for retweets we can capture real engagement, as shares imply a strong interest in the content.
  - We are also taking into account the number of likes which could directly affect the relevance of the tweet and it is important to take them into consideration.
- **Cons**
  - Although relevance has a higher weight, tweets with high engagement (likes, retweets) may still rank well even if they are not so relevant to the query. This bias could occasionally leave lower in the rank more relevant but less popular tweets.
  - Also, since we are dealing with likes and retweets to compute our score, someone that has a private account has less probabilities of receiving these interactions with other users and, therefore, less probabilities of increasing its popularity.

BM25

We have defined `bm25(query, inverted_index, total_tweets, clean_tweets)`. This function first computes the average length of all tweets and, then, for each term in the query we compute its DF and its IDF. Having this, for each document that contains the query term, we compute the TF of that term in the document and add the computation for the RSV

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

to that document. As we want to compare the results with the TF-IDF, we will use the same strategy and we will use only the common documents (documents that contain all the words in the query).

Finally, we sort the documents and we can observe that for a given query, for example, 'indian government' we get the following results:

Note that the output of the code is the top 20 tweets, but, for simplicity in this report we include the top 10.

Top 1 tweet: tweet_id = doc_29545 - score = 3.9028930433839264
Content: #FarmersProtest What a shame on Indian government. https://t.co/WgtT04W6ur

Top 2 tweet: tweet_id = doc_31188 - score = 3.9028930433839264
Content: The Indian government is facist #FarmersProtest

Top 3 tweet: tweet_id = doc_11484 - score = 3.9028930433839264
Content: Shame on Indian government #FarmersProtest https://t.co/khyXi7pNAy

Top 4 tweet: tweet_id = doc_38240 - score = 3.9028930433839264
Content: Shame on Indian Government 😔
#FarmersProtest https://t.co/XuLV671Gof

Top 5 tweet: tweet_id = doc_17769 - score = 3.9028930433839264
Content: Shame on Indian government
#FarmersProtest https://t.co/oW61P13nXH

Top 6 tweet: tweet_id = doc_36211 - score = 3.07862728150336
Content: #FarmersProtest look at all the bs the Indian government is doing.
https://t.co/7KOggZw8pu

Top 7 tweet: tweet_id = doc_27778 - score = 3.07862728150336
Content: indian government #BJP all #FarmersProtest target https://t.co/Qu3KuEkjnt

Top 8 tweet: tweet_id = doc_30422 - score = 3.07862728150336
Content: Why are Indian farmers protesting against the government?
#FarmersProtest https://t.co/eMUGoXtabZ

Top 9 tweet: tweet_id = doc_34728 - score = 3.07862728150336
Content: So disgusted by Indian government now !
#FarmersProtest #FreeDishaRavi

Top 10 tweet: tweet_id = doc_40124 - score = 3.07862728150336
Content: Shame on Indian government #FarmersProtest #GoBackModi https://t.co/97vElTIDkY

Comparing these results with the TF-IDF results we observe that both methods retrieve many of the same tweets in the top 10 list. However, the order is different between the two methods, indicating different weightings and influences on ranking scores.
Also, TF-IDF has tied scores for several tweets, as seen with a score of **1.9905...** for the top five tweets. This suggests that this method does not always distinguish between very similar

documents. On the other hand, BM25 has unique scores for each tweet, even when the content is similar, thanks to its consideration of document length and term saturation. This allows BM25 to differentiate better between very similar tweets, leading to a better ranking.

How the Rankings Differ when using TF-IDF and BM25

Regarding Term Frequency, in TF-IDF the score is directly proportional to term frequency. If a term appears more frequently in a document, its score increases. This is why documents that have high term frequencies for the query terms may rank higher in TF-IDF ranking. BM25, however, limits the influence of TF by introducing the k1 parameter. Documents in BM25 ranking with less term frequency may still rank similarly against those with very high term frequencies, as it does not guarantee a higher score.

Regarding document length normalization, in TF-IDF longer documents with frequent terms can sometimes receive very high scores if they have high term frequency, but BM25 includes document length normalization via the b parameter. This helps prevent longer documents from having an advantage in the ranking.

Pros and Cons of using them

- TF-IDF is effective for small datasets or when simplicity is required as it is more straightforward to compute. It is easier to interpret because it directly reflects the frequency of terms in both the document and the collection, which is good to understand term relevance. However, it lacks length normalization, which can favor longer documents that contain query terms more frequently, even if they are not so relevant.
- BM25 implements document length normalization, which is better when dealing with documents of different lengths. Also, k1 parameter helps prevent terms that appear a lot of times in one document from changing the ranking excessively. However, this ranking is more complex to implement and as its effectiveness depends on the values of k1 and b, an incorrect adjustment can lead to bad performance.

Word2Vec + cosine similarity

Now, in this part we are going to create a tweet ranking model using Word2Vec vectors, where each tweet is represented by an averaged vector of its word embeddings.

What we do at the beginning is to train the Word2Vec model using the `clean_tweets`, generating vectors of 50 dimensions that captures the relationships between words. Once we have this, we have created the function `def calculate_tweet_vector(model, tweet)` that calculates a vector for every tweet making an average of the vectors of its words. Then the function `def top20(query)` retrieves the most relevant 20 tweets for the query by calling the 'tfidf_cosine' function to have a ranking based on similarity between the query and document vector. After calling this function, we compute the tweet vector of each tweet in the top 20 ranked documents and, then, to reduce dimensionality and simplify the calculus of the similarity, we apply PCA (Principal Component Analysis) to this vectors, making them in an space of 3 dimensions. Then the query is converted into an averaged vector using the function 'calculate_tweet_vector' defined before and it is transformed with

PCA to align it with the vectors of the tweets. Finally we compute the cosine similarity between the query and every tweet in the reduced space, rank the tweets by relevance and select the top 20.
So when applying it, we observe that for a given query, for example 'indian government', we get the following results:

Top 1 tweet: tweet_id = doc_30422 - score = 5.124883327002816
Content: Why are Indian farmers protesting against the government?
#FarmersProtest https://t.co/eMUGoXtabZ

Top 2 tweet: tweet_id = doc_29545 - score = 2.617013692821783
Content: #FarmersProtest What a shame on Indian government. https://t.co/WgtT04W6ur

Top 3 tweet: tweet_id = doc_11484 - score = 2.617013471444465
Content: Shame on Indian government #FarmersProtest https://t.co/khyXi7pNAy

Top 4 tweet: tweet_id = doc_17769 - score = 2.6170134714444635
Content: Shame on Indian government
#FarmersProtest https://t.co/oW61P13nXH

Top 5 tweet: tweet_id = doc_38240 - score = 2.617013471444463
Content: Shame on Indian Government 😔
#FarmersProtest https://t.co/XuLV671Gof

Top 6 tweet: tweet_id = doc_27778 - score = 1.3521834453904433
Content: indian government #BJP all #FarmersProtest target https://t.co/Qu3KuEkjnt

Top 7 tweet: tweet_id = doc_31188 - score = 0.6419620845731623
Content: The Indian government is facist #FarmersProtest

Top 8 tweet: tweet_id = doc_14237 - score = 0.09851665984703722
Content: Shame on Indian government #FarmersProtest #DPstopIntimidatingFarmers
https://t.co/9OftsBizhY

Top 9 tweet: tweet_id = doc_42096 - score = 0.09214460995373694
Content: I am appalled by the ignorance of the Indian government. #FarmersProtest

Top 10 tweet: tweet_id = doc_18946 - score = -0.43689440518788036
Content: Indian government has to do better! Seriously. #FarmersProtest

Top 11 tweet: tweet_id = doc_36211 - score = -0.49062644761468366

Content: #FarmersProtest look at all the bs the Indian government is doing.
https://t.co/7KOggZw8pu

Top 12 tweet: tweet_id = doc_15947 - score = -0.9413180409449402
Content: Indian Government works for the super rich .. not for the Indian people
#ReleaseDetainedFarmers
#FarmersProtest https://t.co/rDtcga8mAw

Top 13 tweet: tweet_id = doc_31710 - score = -1.1176120390683564
Content: #IamAgainstModiGovt
#farmersprotest
Shame on indian government https://t.co/RMwUNfYMV1

Top 14 tweet: tweet_id = doc_8470 - score = -1.372189071180373
Content: #farmersprotest. @KanganaTeam @PMOIndia shame on Indian government
https://t.co/YERIeLznPS

Top 15 tweet: tweet_id = doc_40124 - score = -1.846063157361935
Content: Shame on Indian government #FarmersProtest #GoBackModi https://t.co/97vElTIDkY

Top 16 tweet: tweet_id = doc_40125 - score = -1.84606318945095
Content: Shame on Indian government #GoBackModi #FarmersProtest https://t.co/md7soP8JOL

Top 17 tweet: tweet_id = doc_13788 - score = -1.9624124789517385
Content: This is heartbreaking 💔 Indian government literally has no shame. #FarmersProtest
https://t.co/w5k09WEJj7

Top 18 tweet: tweet_id = doc_34728 - score = -2.1148998628616904
Content: So disgusted by Indian government now !
#FarmersProtest #FreeDishaRavi

Top 19 tweet: tweet_id = doc_14748 - score = -2.3731434231448243
Content: @ambedkariteIND Omg , shame on you Indian government #FarmersProtest

Top 20 tweet: tweet_id = doc_41580 - score = -3.2765221181549977
Content: Why Indian celebrities defend the government after Rihanna tweeted #FarmersProtest
https://t.co/dCGwRR6Avv

Can you imagine a better representation than Word2Vec?

Sentence2Vec and Doc2Vec are two representations that can perform better than Word2Vec, however in the context of tweets we can say that the most optimal approach is Sentence2Vec since it is particularly useful to analyze shortest texts which is the case. Analyzing tweets with Doc2Vec can be broader and with Word2Vec can be a lack of context.

Having a deeper understanding of Doc2Vec we can see some advantages and disadvantages compared with Word2Vec and Sentence2Vec:

- **Pros:** it generates embeddings of the whole tweet, capturing the meaning of the entire text and making it perfect for tasks like ranking and similarity. It is more accurate than Word2Vec as it analyzes the whole tweet, not just the words so it takes more context, that is especially useful in cases when the tweet can be ambiguous.
- **Cons:** they are slower to train, especially when the data is large. Also, it needs significant data to have more accurate outputs as it can overfit smaller datasets which can be the case of small tweets that do not have that much context, in this case Doc2Vec may be less accurate than Sentence2Vec.

Therefore, by looking at the advantages and disadvantages of Sentence2Vec compared to Word2Vec and Doc2Vec we can see this is the most accurate approach:

- **Pros:** Sentence2Vec is highly effective in capturing the context of a whole sentence, which is more precise and gives more context than Word2Vec. Sentence2Vec captures both the meaning of individual words and their relationship within the tweet. This analysis is more efficient than ones for Word2Vec and Doc2Vec in the context of tweets as it manages to process better shortest texts making it computationally lighter and also faster to train.
- **Cons:** Sentence2Vec may struggle when analyzing tweets that may be longer or that have more complexity than usual and thus the accuracy of the output can be affected.

We have implemented a Sentence Transformer and Doc2Vec models to observe their results with the query ‘indian government’ as an example. After observing the results, we can conclude that Sentence2Vec (sentence transformers) outperforms Doc2Vec. Sentence2Vec has higher cosine similarity scores and better ranking performance and this shows that it is better for tasks where semantic relationships between short texts (like tweets) and queries is important. For instance, the highest-ranking tweet in Doc2Vec (doc_11484) has a score of 0.0898 even though we have seen in the previous ranking examples that it was always one of the most relevant tweets to the query.

Then, as stated above, Sentence2Vec would be the best choice for this particular tweet ranking task.