PROJECT REPORT

IRWA - PART 3 - G102-2

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

Universitat Pompeu Fabra Barcelona

1. Introduction

This document is the Part 3 Project Report, prepared by team G102-2. Here, it is described all the necessary information about the elaboration of the code for this third part, including the assumptions, methods, and algorithms employed. Each subsection presents the reasoning behind the decisions made and is classified into two main parts and a conclusion.

The team developed the code using a Google Colaboratory file, which required importing libraries and the provided dataset. The project aims to create a ranking system that retrieves and sorts relevant tweets based on given queries, using both classical and customized scoring mechanisms. The ranking techniques tested include TF-IDF with cosine similarity, BM25, and a custom ranking method. Additionally, we explore a Word2Vec-based ranking approach to evaluate potential improvements in query-document relevance through semantic similarity.

The repository for the deliveries has been shared with the labs teacher Francielle Do Nascimento, but since it is a public repository, the link to get in is the following:

https://github.com/QuerZamora/IRWA_G102_2

IMPORTANT

Since Part 3 was developed and submitted before receiving feedback on Part 2, any potential issues from Part 2 may be carried forward here. We appreciate understanding on this matter, as certain aspects may have been influenced by incorrect elements from earlier phases. Moreover, since we are only asked to see the results of one query, in the report we are only going to see the results of one of it, however, in the Collab we can find the results of all our queries.

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

2. Ranking score

2.1 Ways of Ranking

2.1.1 a) TF-IDF + Cosine Similarity

The first ranking method implemented in this project uses TF-IDF (Term Frequency-Inverse Document Frequency) combined with cosine similarity to evaluate the relevance of each tweet to a given query. The TF-IDF scoring method is commonly used in information retrieval tasks to determine how significant specific words are within documents and across a collection of documents, based on their frequency. This method allows us to balance the importance of commonly occurring words, which may not provide much relevance to a query, against less frequent but potentially more informative terms.

```python
def rank_with_tfidf(query, documents):
    """
    Ranks documents based on a query using TF-IDF and cosine similarity.

    Parameters:
    - query: The query string.
    - documents: A list of document strings.

    Returns:
    - A list of (document, score) tuples, ranked by relevance to the query.
    """
    # Combine query and documents for TF-IDF computation
    all_texts = [query] + documents
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(all_texts)

    # Calculate cosine similarity between the query vector and document vectors
    query_vector = tfidf_matrix[0]
    doc_vectors = tfidf_matrix[1:]
    cosine_similarities = cosine_similarity(query_vector, doc_vectors).flatten()

    # Rank documents by similarity score
    ranked_docs = sorted(zip(documents, cosine_similarities), key=lambda x: x[1], reverse=True)
    return ranked_docs[:20]
```

The code implemented begins by combining the query and tweet documents for a comprehensive TF-IDF computation. Once the TF-IDF vector representations of the query and each document are calculated, we use cosine similarity to measure the closeness between the query vector and each tweet's vector. A higher cosine similarity score indicates a closer match between the query and the tweet, with numbers between 0 and 1.

For each query, the TF-IDF ranking function processes the tweet dataset and returns the top 20 tweets based on their cosine similarity scores.

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

2.1.2 b) Our Score + Cosine Similarity

Approach and Procedure

For this implementation, we will use Item-based Collaborative Filtering, adapted to the context of tweets and vocabulary. This method considers items (words in tweets) that a user has already "consumed" (that appear in relevant tweets) and recommends other items that have been rated similarly.

```python
def get_top20_our_score(query, tweets_df):
    """
    query: text of the query
    tweets_df: DataFrame of tweets, must include 'Likes', 'Retweets', and 'Tweet' columns

    This function calculates a custom score based on popularity
    (Likes and Retweets) and cosine similarity with the query, and returns the
    20 most relevant tweets.
    """
    # Calculate the popularity score for each tweet
    tweets_df['nota_tweet'] = tweets_df.apply(calcular_nota, axis=1)

    # Filter the tweets with a minimum score of 3
    df_important_tweets = tweets_df[tweets_df.nota_tweet >= 3].copy()

    # Filter the tweets that contain all the words in the query
    query_words = set(query.lower().split())
    df_important_tweets['contains_query'] = df_important_tweets['Tweet'].apply(lambda x: query_words.issubset(set(x.lower().split())))
    df_filtered = df_important_tweets[df_important_tweets['contains_query']].copy()

    if df_filtered.empty:
      print('No important docs found')
      return df_filtered

    # Create the TF-IDF representation for the set of tweets and the query
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(df_filtered['Tweet'])

    # Vector of the query
    query_tfidf = tfidf_vectorizer.transform([query])

    # Calculate the cosine similarity between the query and the tweets
    cosine_scores = cosine_similarity(query_tfidf, tfidf_matrix).flatten()
    df_filtered['cosine_similarity'] = cosine_scores

    # Define the custom score (Your-Score) combined with the cosine similarity
    df_filtered['our_score'] = df_filtered['nota_tweet'] * df_filtered['cosine_similarity']

    # Sort and display the top-20 according to the Your-Score
    top_20_custom = df_filtered.sort_values(by='our_score', ascending=False).head(20)

    return top_20_custom[['Tweet', 'Likes', 'Retweets', 'nota_tweet', 'cosine_similarity', 'our_score']]
```

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

Universitat Pompeu Fabra Barcelona

Recommendation Process

To recommend words that might be important in a search query, we follow these steps:

1. Calculate the Popularity of Each Tweet:

We calculate a popularity score for each tweet using the formula:

$$\text{tweet\_mark} = \log(\text{Likes} + \text{Retweets} + 1)$$

where:

Likes: The total number of likes on the tweet.

Retweets: The total number of retweets on the tweet.

We filter tweets with a minimum popularity score to keep only the most relevant ones.

2. Filter Tweets Based on the Query:

We filter tweets that contain all the words in the query, allowing us to focus on the tweets most relevant to that query.

3. Vector Representation of Tweets and Query with TF-IDF:

We convert the text of the tweets into a TF-IDF vector matrix, which assigns weights to words based on their frequency in each tweet and their rarity across the entire set of tweets.

4. Cosine Similarity Between the Query and Tweets:

We calculate the **cosine similarity** between the query vector and the vectors of each filtered tweet. This helps identify tweets whose word composition is most similar to the query.

5. Calculate the Custom Score (Your-Score):

We combine the popularity score and cosine similarity into a custom metric, called our_score, calculated as:

$$\text{our\_score} = \text{tweet\_mark} * \text{cosine\_similarity}$$

This score weighs tweets that are not only popular but also relevant to the query in terms of word composition.

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

6. Top 20 Recommended Tweets:

Finally, we sort the tweets by **our_score** and select the top 20 as recommendations, providing a list of tweets that are both popular and relevant to the query.

This approach allows us to identify the tweets that best respond to a query in terms of thematic relevance and popularity. Additionally, by calculating cosine similarity with the TF-IDF model, we prioritize tweets whose vocabulary more closely resembles that of the user's query.

Pros and **Cons**

The Our Score + Cosine Similarity method combines popularity (likes and retweets) with semantic relevance of tweets using cosine similarity and TF-IDF. Its advantages include a balanced recommendation between popularity and relevance, effective filtering of irrelevant tweets, and ease of implementation. However, it relies too much on popularity, which can skew results towards popular content that may not be highly relevant. Additionally, it is limited to exact word similarity, failing to capture nuances such as synonyms or context, and it doesn't consider the user's previous interactions, reducing personalization.

Result

Query: protest farmer

	index	Tweet	Likes	Retweets	nota_tweet	cosine_similarity	our_score
0	32032	keep support peac farmer protest	3679	1554	8.562931	0.234164	2.005129
1	34406	indian largest protest world support farmer	706	720	7.263330	0.205200	1.490438
2	305	great illustr protest farmer protest delhi	235	50	5.655992	0.237512	1.343368
3	97922	farmer protest germani	149	62	5.356586	0.250239	1.340428
4	34197	indian largest protest world support farmer	232	254	6.188264	0.205200	1.269834
5	116255	farmer protest turn point india time	1055	439	7.309881	0.155920	1.139759
6	64112	countless human right org farmer amp activist ...	2781	1201	8.289791	0.125611	1.041290
7	34472	largest protest world support farmer	31	47	4.369448	0.230600	1.007593
8	91123	farmer protest germani	28	13	3.737670	0.250239	0.935312
9	93623	farmer protest across india	21	16	3.637586	0.254981	0.927517
10	34298	protest constitut right delhi polic wrong char...	68	84	5.030438	0.180918	0.910096
11	44296	indian farmer protest explain part 3	86	43	4.867534	0.180694	0.879534
12	34291	indian largest protest world support farmer	24	46	4.262680	0.205200	0.874704
13	105169	guy pleas keep use follow hashtag farmer prote...	3604	1355	8.509161	0.102638	0.873364
14	46951	farmer protest imag bihar	109	36	4.983607	0.174514	0.869707
15	33581	indian largest protest world support farmer	22	37	4.094345	0.205200	0.840161
16	33673	indian largest protest world support farmer	33	24	4.060443	0.205200	0.833205
17	98048	per report khattar govt plan bring law penalis...	1153	395	7.345365	0.108057	0.793716
18	49232	farmer start protest	15	5	3.044522	0.260172	0.792099
19	30746	support farmer protest repeal farm law	22	28	3.931826	0.199572	0.784682

These results show the top 20 tweets for the query "protest farmer," ranked by a custom score that prioritizes both popularity and relevance to the query. The nota_tweet (popularity score) is calculated based on the number of likes and retweets, with higher values indicating more popular tweets. Additionally, the cosine_similarity measures how closely the content of each tweet matches the words in the query. The our_score combines these two factors, giving higher weight to tweets that are both popular and contextually relevant.

Thus, the tweets shown here are the ones that are not only highly liked and retweeted but also contain relevant terms from the search query. This approach ensures that the most relevant and engaging tweets for the query "protest farmer" are recommended.

2.1.3 c) BM25

BM25 ranking function is one of the most effective relevance-based ranking models in Information Retrieval. It is designed to retrieve and rank documents based on their relevance to a given query. BM25 calculates a relevance score for each document by analyzing term frequency, document length, and other variables.

```python
def rank_with_bm25(query, documents, k1=1.5, b=0.75):
    """
    Ranks documents based on a query using BM25.

    Parameters:
    - query: The query string.
    - documents: A list of document strings.
    - k1: BM25 hyperparameter for term frequency scaling.
    - b: BM25 hyperparameter for document length normalization.

    Returns:
    - A list of (document, score) tuples, ranked by BM25 relevance to the query.
    """
    # Tokenize the query and documents
    query_terms = query.lower().split()
    doc_terms = [doc.lower().split() for doc in documents]

    # Compute document frequencies (df) and average document length
    doc_lengths = [len(doc) for doc in doc_terms]
    avg_doc_length = np.mean(doc_lengths)
    df = Counter(term for doc in doc_terms for term in set(doc))  # Document frequencies (unique terms in each doc)

    # Total number of documents (N)
    N = len(documents)

    # Compute BM25 score for each document
    def idf(term):
        """Inverse Document Frequency for a given term."""
        return math.log((N - df.get(term, 0) + 0.5) / (df.get(term, 0) + 0.5) + 1.0)

    def bm25_score(doc, query_terms):
        """Compute BM25 score for a single document and query."""
        score = 0
        doc_counter = Counter(doc)  # Term frequencies in the document
        for term in query_terms:
            if term in doc_counter:
                tf = doc_counter[term]
                score += idf(term) * (tf * (k1 + 1)) / (tf + k1 * (1 - b + b * len(doc) / avg_doc_length))
        return score

    # Rank documents by BM25 score
    bm25_scores = [(doc, bm25_score(doc, query_terms)) for doc in doc_terms]
    ranked_docs = sorted(bm25_scores, key=lambda x: x[1], reverse=True)

    return ranked_docs[:20]
```

The function takes a query and a set of documents, applying the BM25 algorithm to rank documents based on their relevance to the query. This function uses two main hyperparameters, “k1” controls term frequency scaling. Higher k1 value makes term frequency (TF) more important, while lower values reduce TF importance. “b” adjusts document length normalization.

The function follows these steps to rank the documents. First, both the query and documents are tokenized. Document Frequency (DF) is then calculated, counting the number of documents containing each unique term, which is essential for computing the Inverse Document Frequency (IDF). To normalize scores, especially for longer documents, the function calculates the average document length. For each query term, the IDF is determined using the formula:

$$IDF = log(\frac{N - df(term) + 0.5}{df(term) + 0.5} + 1.0)$$

where N represents the total number of documents. Then we apply BM25 scoring, where each document receives a relevance score based on the term frequency, document length, and IDF of query terms within the document. Finally, the function sorts the documents by their BM25 scores in descending order and returns the top 20 most relevant results.

Some results:

Query: protest farmer

	Tweet	Rating
0	[farmer, protest, four, month, farmer, protest...	6.114822
1	[farmer, protest]	5.919814
2	[farmer, protest]	5.919814
3	[farmer, protest]	5.919814
4	[farmer, protest]	5.919814
5	[farmer, protest]	5.919814
6	[farmer, protest]	5.919814
7	[farmer, protest]	5.919814
8	[farmer, protest]	5.919814

These results demonstrate the BM25 model's ability to prioritize documents that match the query in term frequency with highly relevant documents achieving higher BM25 scores. This ranking approach is effective for returning relevant tweets with terms that align well with the query context.

2.1.4 Comparing TF-IDF and BM25

Query: protest indian

	Tweet	Rating
0	indian protest	1.000000
1	indian protest	1.000000
2	indian protest	1.000000
3	indian protest	1.000000
4	indian protest	1.000000
5	indian protest	1.000000
6	indian protest	1.000000
7	indian protest	1.000000
8	indian protest	1.000000
9	indian protest	1.000000
10	indian protest	1.000000
11	indian farmer protest	0.912791
12	indian	0.750460
13	indian farmer protest govern	0.748612
14	indian farmer peac protest	0.708037
15	protest	0.660916
16	indian farmer protest matter british indian	0.656116
17	indian farmer protest matter british indian	0.656116
18	indian farmer	0.644583
19	amitshah indian farmer protest u	0.620198

Query: protest indian

	Tweet	Rating
0	[indian, protest]	8.406879
1	[indian, protest]	8.406879
2	[indian, protest]	8.406879
3	[indian, protest]	8.406879
4	[indian, protest]	8.406879
5	[indian, protest]	8.406879
6	[indian, protest]	8.406879
7	[indian, protest]	8.406879
8	[indian, protest]	8.406879
9	[indian, protest]	8.406879
10	[indian, protest]	8.406879
11	[indian, farmer, protest]	7.905432
12	[indian, farmer, protest, matter, british, ind...	7.899865
13	[indian, farmer, protest, matter, british, ind...	7.899865
14	[indian, farmer, peac, protest]	7.460437
15	[indian, farmer, protest, govern]	7.460437
16	[punyaab, farmer, indian, n, everi, person, pr...	7.409171
17	[indian, farmer, protest, matter, british, ind...	7.314073
18	[uk, protest, support, indian, farmer]	7.062871
19	[indian, farmer, right, peaceful, protest]	7.062871

TF-IDF (left) vs BM25 (right)

The ranking methods TF-IDF and BM25 differ in how they assess and prioritize relevance, leading to notable differences in the ranking outcomes.

TF-IDF (Term Frequency-Inverse Document Frequency) ranks tweets based on word frequency within the tweet relative to its rarity across the entire corpus. In the results above, TF-IDF gives a high similarity score of 1.0 to tweets containing both "indian" and "protest," but it also assigns progressively lower scores to tweets with variations of these terms and other additional terms. TF-IDF's reliance on frequency can make it less sensitive to differences in term density within documents, which sometimes leads to similar scores for documents of varying lengths, provided they contain the query terms.

BM25, by contrast, is a probabilistic ranking model that not only considers term frequency and document frequency but also adjusts based on document length, rewarding shorter, query-dense tweets. This means BM25 generally assigns higher scores to tweets where query terms are more concentrated and frequent. For example, tweets containing only the essential query terms “indian” and “protest” have the highest BM25 score (8.40), while tweets with additional context score slightly lower. BM25’s ability to scale with document length and term saturation generally allows it to better handle longer texts with relevant terms dispersed throughout.

Pros and **Cons**

- TF-IDF Pros: Simple, computationally efficient, and effective for short, structured text. However, it lacks length normalization, which can make it less effective with documents of varying lengths.
- TF-IDF Cons: It may miss context, as it doesn’t account for term saturation or document length, potentially overvaluing infrequent terms and underweighting important frequent terms.
- BM25 Pros: More flexible in handling different text lengths and term frequencies, making it robust for ranking documents of varying lengths, like tweets with diverse content. This makes it highly effective in search tasks and suited for real-world datasets.
- BM25 Cons: Slightly more computationally intensive due to its length normalization and saturation adjustments.

In summary, BM25 often provides more nuanced rankings, especially beneficial for longer and more complex documents, whereas TF-IDF can be effective for simpler, shorter texts, but may lack the same level of adaptability.

2.2 Top-20

Top-20 list of documents for each of the 5 queries, using word2vec + cosine similarity:
We want to find tweets that match each query (for example, "Indian protest" or "support for farmers") by comparing the meaning of the query to that of each tweet. To do this, we’ll use Word2Vec, which converts words into numerical vectors that capture the “meaning” of each word in a space of numbers.

1. Load a Word2Vec model:

First, we need a Word2Vec model, which is like a dictionary that provides a vector (a set of numbers) for each word. These numbers capture each word’s meaning in relation to others.

2. Convert tweets to vectors ("tweet2vec"):

Word2Vec gives a vector for each word, but we need a single vector for each tweet. To do this, we take the vector for each word in the tweet and compute an average of these vectors. The result is a vector that represents the overall meaning of the complete tweet.

3. Convert the query to a vector:

Just as we did for the tweets, we take each word in the query, obtain its vector with Word2Vec, and calculate the average. This way, we convert the entire query into a single vector.

4. Measure similarity:

Once we have the vector for the query and the vector for each tweet, we calculate the cosine similarity between them, which tells us how similar they are in terms of meaning. Then, we can sort the tweets by their similarity to the query and return the 20 most similar.

```python
# train a Word2Vec model on the preprocessed tweet data
def train_word2vec_model(tweets):
    # split tweets into words
    processed_tweets = [tweet.split() for tweet in tweets]

    # train Word2Vec model
    model = Word2Vec(sentences=processed_tweets, vector_size=100, window=5, min_count=1, workers=4)

    return model

tweets = tweet_info_df['Tweet'].tolist()
model = train_word2vec_model(tweets)

# tweet to a vector by averaging its word vectors
def tweet2vec(tweet, model):
    words = tweet.split()
    word_vectors = []

    for word in words:
        if word in model.wv:
            word_vectors.append(model.wv[word])  #word vector from the model

    if len(word_vectors) > 0:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(model.vector_size)

# rank documents using Word2Vec and cosine similarity
def rank_with_word2vec(query, documents, model):
    query_vector = tweet2vec(query, model)
    doc_vectors = [tweet2vec(doc, model) for doc in documents]

    cosine_similarities = cosine_similarity([query_vector], doc_vectors).flatten()

    ranked_docs = sorted(zip(documents, cosine_similarities), key=lambda x: x[1], reverse=True)
    return ranked_docs[:20]
```

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

Universitat Pompeu Fabra Barcelona

Results:

Query: protest indian

	Tweet	Rating
0	indian protest	1.000000
1	indian protest	1.000000
2	indian protest	1.000000
3	indian protest	1.000000
4	indian protest	1.000000
5	indian protest	1.000000
6	indian protest	1.000000
7	indian protest	1.000000
8	indian protest	1.000000
9	indian protest	1.000000
10	indian protest	1.000000
11	indian farmer protest	0.958700
12	indian farmer protest matter british indian	0.947590
13	indian farmer protest matter british indian	0.947590
14	protest india	0.945592
15	stevehank largest crowd ever seen protest exce...	0.940071
16	uk protest support indian farmer	0.938611
17	indian farmer protest govern	0.933822
18	indian farmer peac protest	0.931044
19	garba gujarati way protest protest	0.929111

Query: protest farmer

	Tweet	Rating
0	farmer protest	1.000000
1	farmer protest	1.000000
2	farmer protest	1.000000
3	farmer protest	1.000000
4	farmer protest	1.000000
5	farmer protest	1.000000
6	farmer protest	1.000000
7	farmer protest	1.000000
8	farmer protest	1.000000
9	farmer protest	1.000000
10	farmer protest	1.000000
11	farmer protest	1.000000
12	farmer protest	1.000000
13	farmer protest	1.000000
14	farmer protest	1.000000
15	farmer protest	1.000000
16	farmer protest	1.000000
17	farmer protest	1.000000
18	imanikaurrai farmer protest	0.999989
19	jashanlambi malkitsaggu99 farmer protest	0.999937

Query: support farmer

	Tweet	Rating
0	support farmer	1.0
1	support farmer	1.0
2	support farmer	1.0
3	support farmer	1.0
4	support farmer	1.0
5	support farmer	1.0
6	support farmer	1.0
7	support farmer	1.0
8	support farmer	1.0
9	support farmer	1.0
10	support farmer	1.0
11	support farmer	1.0
12	support farmer	1.0
13	support farmer	1.0
14	support farmer	1.0
15	support farmer	1.0
16	support farmer	1.0
17	support farmer	1.0
18	support farmer	1.0
19	support farmer	1.0

Query: free india

	Tweet	Rating
0	british empir arrest freedom fighter today ind...	0.966668
1	gujarati kill india	0.963422
2	murder democrat india	0.962276
3	india free democraci anymor	0.961724
4	call freedom speechexpress india	0.959731
5	mother india reboot 202021	0.959429
6	india system oppress export free	0.954395
7	babagloc karunanundi india judiciari rotten ce...	0.953333
8	youngster india mute	0.952057
9	call free india lost freedom mani year freedom...	0.951659
10	freedom india	0.951601
11	free countri	0.950625
12	inde india freedom goe die	0.950238
13	new india nazi india old thing india	0.949517
14	india treat elderli fought countryshameless	0.948889
15	vulner citizen india suffer	0.948603
16	india still democrat nation	0.948595
17	enemi india	0.947851
18	india regress danger level	0.947760
19	minor safe india	0.947549

Query: universitat pompeu fabra

	Tweet	Rating
0	world progress indian polic govt still tri tak...	0.0
1	kisanektamorcha farmer constantli distroy crop...	0.0
2	reallyswara rohinisgh watch full video	0.0
3	mandeeppunia1 watch full video	0.0
4	mandeeppunia1 watch full video	0.0
5	anoth farmer malkeet singh mahilpur hoshiarpur...	0.0
6	akshaykumar hi tell boss thank	0.0
7	taapse watch full video	0.0
8	despit increas must increas petroleummin pmoindia	0.0
9	mockeri menac charg	0.0
10	taapse watch full video	0.0
11	what left hear modi lol	0.0
12	shekhargupta khanthefatima mainabisme give new...	0.0
13	anoth farmer malkeet singh mahilpur hoshiarpur...	0.0
14	pride britain chariti hero help poor india far...	0.0
15	hi im common man pleas find time elect make fa...	0.0
16	pride britain theerasingh	0.0
17	sarahwoodwrit vivianavigil know tiger wood acc...	0.0
18	omg modi tri replac everyon els histori presen...	0.0
19	biggest enemi indian peopl zeenew aajtak repub...	0.0

1. Query: "protest indian":

The top tweets here are exactly "indian protest," which directly matches the query and has a similarity score of 1.000. This means they contain both "indian" and "protest," appearing in the most relevant order. As we move down the list, we see variations like "indian farmer protest" or "protest india," which still include both query words but in slightly different contexts. The similarity decreases from 1.000 to 0.929, suggesting that these tweets are considered slightly less relevant than the first few tweets but remain thematically related.

2. Query: "protest farmer":

The top results here are exclusively "farmer protest," which perfectly matches the query words and has a full similarity score of 1.000. The next results are variations on the same theme, like "imanikaurrai farmer protest" or "jashanlambi malkitsaggu99 farmer protest," which contain the exact query words but add extra names or words. The similarity remains high across all tweets, around 0.999, showing they are strongly related to "farmer protest."

3. Query: "support farmer":

All the top 20 tweets are exactly "support farmer," perfectly matching the query with a similarity of 1.000. There are no additional context variations, which suggests that these exact words are the only highly relevant tweets in the dataset for this query.

4. Query: "free india":

The results for this query are more varied, as there is no exact match for "free india" alone. The top results include tweets with phrases like "british empire arrest freedom fighter today india" and "gujarati kill india," which contain words associated with freedom or the political context in India but in broader contexts. Similarities here are lower, starting at 0.966 and gradually dropping to 0.947, indicating that while the tweets are somewhat relevant to "free India" or "freedom in India," the match is less direct than in previous queries.

5. Query: "universitat pompeu fabra":

For this query, all results show a similarity of 0.0. This means there are no tweets in the dataset that contain this phrase or related words. The term "universitat pompeu fabra" seems unrelated to the main themes in the dataset, which appears to focus primarily on protest and political issues in India.

Overall Conclusion:

- Direct match queries ("protest indian," "protest farmer," "support farmer"): Word2Vec works well in identifying tweets with exact or near-exact matches, prioritizing tweets that contain the keywords in the same context as the query.
- Broader thematic queries ("free india"): In these cases, Word2Vec provides tweets generally relevant to the topic but without an exact match. It still finds related contexts, though with slightly lower similarity scores.
- Queries not represented in the dataset ("universitat pompeu fabra"): Results with a similarity of 0.0 indicate that no tweets are related to this particular query within the dataset.

This shows that Word2Vec is useful for identifying direct matches and related themes but may struggle if the query topic isn't represented in the dataset or if the query is very specific and unrelated to the dominant themes in the tweets.

2.3 Better than Word2vec?

Yes, there are several representations that build upon or extend the concept of Word2Vec, such as Doc2Vec and Sentence2Vec. Each of these models has specific use cases, strengths, and limitations compared to Word2Vec. Here's an overview of these representations, their pros, and cons, along with justifications:

1. Word2Vec:

Word2Vec represents words as dense, fixed-length vectors based on their context within a corpus. It uses techniques like Continuous Bag of Words (CBOW) and Skip-gram to learn these representations.

Pros:

- Efficient: Quick to train and scales well to large datasets.
- Contextual similarity: Captures semantic relationships between words effectively, enabling operations like "king" - "man" + "woman" ≈ "queen".

Cons:

- Lacks contextual awareness: The same word has one fixed vector, regardless of its meaning in different contexts.
- Sentence and document representation: Requires post-processing (e.g., averaging word vectors) to create representations for larger text units, which can be crude and lose nuanced meaning.

2. Doc2Vec:

Doc2Vec, an extension of Word2Vec, is designed to create dense, fixed-length vector representations of entire documents. It does this by incorporating a unique document ID that acts as an additional vector trained alongside the word vectors.

Pros:

- Captures document-level semantics: Represents the global context of a document, making it useful for tasks like document classification, topic modeling, and information retrieval.
- Better than averaging: Unlike simply averaging word vectors, Doc2Vec learns representations that consider the sequence and structure of words in a document.

Cons:

- Training complexity: More complex to train and requires more computational resources compared to Word2Vec.
- Data dependence: The quality of vectors can be heavily dependent on the training corpus, potentially requiring extensive training data for optimal results.

3. Sentence2Vec:

Sentence2Vec is used to create vector representations for individual sentences. Approaches vary from simple (e.g., averaging word embeddings) to more sophisticated methods, like using pre-trained models (e.g., Universal Sentence Encoder or BERT).

Pros:

- Sentence-level representation: Provides vectors that capture the meaning of sentences, which can be more contextually nuanced than Word2Vec or Doc2Vec.
- Use in similarity and matching: Useful for tasks like sentence similarity, semantic search, and chatbot responses.

Cons:

- Loss of granularity: May overlook fine-grained word-level nuances, especially with simpler methods like averaging.
- Complexity with contextual models: More advanced models, such as those based on transformers, require substantial computational power and may be overkill for simpler applications.

Clàudia Morales - u199906
Roger Sola - u199780
Queralt Zamora - u199903
G102-2

Comparative justification:

Is it a Better Representation? Yes, in many cases, Doc2Vec and Sentence2Vec can provide better representations than Word2Vec depending on the task.

- **Doc2Vec** is better for document-level tasks as it captures context more holistically, while Word2Vec's scope is limited to individual words.

- **Sentence2Vec i**s beneficial for sentence-level applications where understanding the flow and semantics of sentences is critical. For more complex, context-dependent representations, transformer-based models like **BERT** (Bidirectional Encoder Representations from Transformers) surpass Word2Vec by producing different embeddings for the same word in different contexts, enhancing performance in tasks like question answering and sentiment analysis.

Some advanced alternatives:

- BERT and Transformer Models:

  Pros: Highly contextual embeddings, ability to represent polysemy (same word with different meanings).

  Cons: High computational requirements, slower training and inference.

Conclusion

The choice between Word2Vec, Doc2Vec, Sentence2Vec, and more advanced models like BERT depends on the specific application:

- For word-level tasks, Word2Vec is efficient and straightforward.
- For document classification or retrieval, Doc2Vec offers richer context.
- For sentence-level tasks, Sentence2Vec or transformer-based models like BERT provide more nuanced and context-aware embeddings.

Overall, Doc2Vec and Sentence2Vec extend the capabilities of Word2Vec by incorporating more complex contexts and structures, making them preferable for applications that require representations beyond isolated words.

3. Conclusions

In Part 3 of the IRWA project, we explored different ranking methods to retrieve relevant tweets for specific queries. We compared three main approaches: TF-IDF with cosine similarity, BM25, and a custom ranking model that combined tweet popularity with relevance. TF-IDF effectively ranked tweets based on word frequency but without taking in count document length normalization, while BM25 provided a more precise ranking, factoring in document length and term frequency saturation, resulting in better relevance for longer tweets.

In the custom model we added popularity to relevance, prioritizing tweets with both high engagement and matching terms. We also tested Word2Vec for semantic ranking but found it less effective.

In summary, BM25 is the most robust for relevance-based ranking. However, TF-IDF and custom scores offer efficiency for simpler cases.