Report Part 3: Ranking.

The goal of the part 3 is find all the documents that contain all the words in the query and sort them by their relevance with regard to the query. For each query we will obtain the top.10 ranking. We used the same queries from part 2. That are the following:

- "Indian government response to farmers".
- "International support for farmers".
- "Demands of farmers' protests".
- "Police action during farmers protest".
- "Schedules and locations of demonstrations and protests".

As we did in part 2, we create a new .ipyn file, enabling us to work in separate files for each part of the project. This approach speeds up execution and makes the results clearer to follow in each section. As consequence, we had to add some functions from part 1 and 2, as process_tweet() or rank_documents(). On other hand, we import the pickle, i.e. the serialized data from create_index_tfidf from part 2. We need term frequencies, document frequencies, the idf values and index for each term in the collection.

In this part we use the dataset of processed tweets that contains the following information for each tweet: Tweet, Date, Hashtags, Hashtags_count, Likes, Retweets, Url, and docId.

1. Score in ranking.

1.1. How the ranking differs when using TF-IDF and BM25.

- Query "Indian government response to farmers".

In the TF-IDF results, the top 20 documents have docIds such as doc_3234, doc_27784, etc. In contrast, the BM25 results show pageIds like doc_13543, doc_30422, etc. There are no direct matches between the most relevant documents returned by the two systems.

For TF-IDF, the scores range from 11.8045 to 6.065, with the majority of scores clustered between 7.0 and 7.5. This shows a broader range of variability in the scores, with some documents having significantly higher scores. In BM25, the scores range from 6.018 to 4.7309, and the distribution is more concentrated around the value 5.47, with a smaller spread in scores. TF-IDF, therefore, shows more variability in its score distribution compared to BM25.

TF-IDF returns a much larger set of documents in total, with 15,989 documents retrieved, from which the top 20 are selected. On the other hand, BM25 retrieves 2,719 documents in total, also selecting the top 20 results from this smaller set.

- Query "International support for farmers".

In the TF-IDF results, the top 20 documents have docIds such as doc_859, doc_21846, doc_12950, etc. In contrast, the BM25 results show pageIds like doc_859, doc_21846, doc_38762, etc. Only documents doc_859 and 21846 appear in both ranks.

In TF-IDF, the scores range from 10.9298 to 6.3614, with the majority of the scores clustered between 7.3 and 8.0. This shows a wider range of higher scores, indicating that TF-IDF has a more varied distribution of relevance across the top documents. In BM25, the scores range from 7.6717 to 3.7902, with a concentration of scores around 4.0 to 5.0. BM25 shows a more uniform distribution, with most scores grouped within a smaller range.

TF-IDF returns a much larger number of documents in total, with 15,767 documents retrieved for the query. In contrast, BM25 retrieves only 443 documents for the same query. Both algorithms focus on the top 20 results.

- Query "Demands of farmers' protests".

The overlap between the two algorithms is notable. Specifically, the following documents appear in both the top 20 results of TF-IDF and BM25, but in different positions inside each rank: doc_15908, doc_17929, doc_43415, doc_887, doc_43810, doc_43799, doc_43790, doc_18273, doc_17894 and doc_30921.

In TF-IDF, the scores range from 12.1176 to 6.0588, with the majority of score between 6 and 7. In BM25, the scores range from 6.5705 to 4.0196, and the majority of the scores are clustered between 4.0 and 5.5. The BM25 distribution is more concentrated with less variability in the scores compared to TF-IDF.

TF-IDF retrieves a larger set of documents overall, with a total of 14,911 documents considered for the search. In contrast, BM25 retrieves only 496 documents in total.

- Query "Police action during farmers protest".

The following documents appear in ranks from TF-IDF and BM25 but in different positions: doc_32026, doc_14464, doc_11674, doc_18802, doc_34650 and doc_32162. The of documents are different.

In the TF-IDF results, the scores range from 11.4668 to 5.7334, with most scores concentrated between 5.7 and 7.1, indicating a relatively higher score spread. In the BM25 results, the scores range from 7.8763 to 3.3481, with the distribution being much more concentrated around scores between 3.3 and 4.6. This shows a more compact range of scores in BM25, with less variability in document relevance compared to TF-IDF.

The TF-IDF algorithm retrieves a much larger set of documents, considering a total of 15,935 documents in its ranking for the query "Police action during farmers protest." On the other hand, BM25 retrieves a smaller set, with 1,448documents.

- "Schedules and locations of demonstrations and protests".

The overlap between the two algorithms' results shows that some documents, such as doc_43892, doc_46453, and doc_45516, appear in both rankings, though there are substantial differences in their ranking order.

In TF-IDF, the scores range from 22.7138 to 8.2594 for the top 20 documents, with a broad range of values. This shows a relatively high variability in how the documents are scored. In BM25, the scores range from 8.9934 to 1.4527 for the top 20 documents, there is a notable concentration of scores between 2.0 and 3.5.

The TF-IDF algorithm retrieves a total of 4,394 documents for the query, from which the top 20 are selected. In contrast, BM25 retrieves only 16 documents, making it a much smaller set from which the top 20 are selected.

1.2. Pros and cons of using TF-IDF and BM25.

In general terms. The following properties are advantages and disadvantages of each ranking algorithm.

TF-IDF's advantages:

- Simplicity: Easy to implement and understand, providing a basic method to quantify the significance of terms.
- Relevance Scoring: Weighs terms based on frequency and rarity, highlighting the most informative terms for a document.
- Dimensionality Reduction: Filters out common stop words, optimizing data for tasks like clustering and classification.
- Effective for Search: Proven effectiveness in information retrieval, helping rank documents by query relevance.
- Data-Agnostic: Does not assume a specific data distribution, ensuring adaptability to various text types.

Disadvantages of TF-IDF:

- Context Ignorance: Fails to recognize the context of terms, treating polysemous words and synonyms as unrelated, which can lead to misinterpretations.

- Document Length Sensitivity: Prone to biases from longer documents due to higher term frequencies, complicating comparisons between documents of different lengths.
- Lack of Semantic Understanding: Does not capture the underlying meaning of words or phrases, limiting content comprehension.
- Sparse Representation: Produces high-dimensional, sparse vectors, which can be inefficient for some machine learning algorithms.
- Static Nature: Generates fixed document representations that do not adapt to shifts in language or term usage over time.

BM25's advantages:

- Dynamic ranking: Unlike the static nature of TF-IDF, BM25 adjusts its ranking based on the distribution of terms within the collection, making it more adaptable to different types of documents and queries.
- Effective for long queries: The ranking function tends to perform better than TF-IDF for longer queries as it addresses the issue of term saturation and considers the overall document length.
- Adjustable Parameters: BM25 allows fine-tuning through its parameters k_1 and and b (offering flexibility to improve performance based on specific datasets.

BM25's disadvantages:

- No semantic understanding: BM25 does not consider the semantic meaning of the query terms or the documents, which means it may not be able to capture the full context of the search.
- No personalization: BM25 treats all users' queries equally, which may not provide personalized results for individual users.
- Resource-Intensive Calculations: BM25 can be computationally intensive due to the need for repeated calculations of term frequencies and document length normalization across the collection, especially for large datasets.

Related to our queries and scoring, BM25 is better for queries needing precise, contextually rich, and highly relevant results (e.g., "Indian government response to farmers" and "Police action during farmers protest") In contrast, TF-IDF is better for broader or exploratory searches where a larger number of documents can provide more varied perspectives or extensive information (e.g., "Demands of farmers' protests" and "Schedules and locations of demonstrations and protests").

1.3. Our score.

I n order to build our score, we first examine the distributions of the Likes, Retweets, and Hashtags_count columns, which are key features in determining the popularity of each tweet on the Twitter social network. We use a function to detect outliers and provide a statistical description of each feature to achieve this goal.

W e note that the Likes and Retweets columns are highly skewed and contain extreme outliers. We decide to apply a logarithmic transformation because it effectively reduces the impact of outliers and compresses the wide range of values in both columns, making them more comparable and easier to work with.

On the other hand, the Hashtags_count column has a more compact distribution, meaning the values are closely grouped together, with less difference between the 25th and 75th percentiles. It also has a less extreme skew compared to Likes and Retweets. In this case, applying a logarithmic transformation could over-compress the values and distort their interpretation, making it unnecessary.

Afterward, we attempted to apply Min-Max scaling to normalize the values of all three columns within the range of 0 to 1. The goal of this normalization was to ensure that all columns were on the same scale, making them easier to compare when calculating the final popularity score.

However, the issue arose when we combined the popularity score with the tf-idf calculation, which we considered essential for ranking and therefore included in the final score. The main challenge was that tf-idf is not scaled, meaning it doesn't fall within the 0 to 1 range like the popularity values after Min-Max normalization. As a result, when both factors (popularity and tf-idf) were combined in the final score for each document, the weight of the popularity factor was significantly reduced. Its normalized values became much smaller compared to the tf-idf values, leading to an underrepresentation of popularity in the final ranking and not properly reflecting its impact.

Instead of continuing to modify the process to adjust both the popularity weight and the tf-idf weight, which would have required several model modifications and would not necessarily solve the problem, we decided to keep the popularity calculation without applying Min-Max Scaling. By doing so, the popularity factor already had an appropriate weight in the final formula, as its values were not excessively compressed after the logarithmic normalization.

Within the calculation of the popularity score, we decided to assign different weights to each factor: Likes, Retweets, and the number of Hashtags. These weights were defined based on

how important each of these factors is in determining the popularity and visibility of a tweet on Twitter:

- 0.5 to Retweets: Retweets are one of the most important factors, as they reflect the reach and spread of the tweet across users' networks. A Retweet typically indicates that the tweet has had significant impact and is more visible to other users.
- 0.3 to Likes: Likes also indicate popularity but have slightly less weight than Retweets. A Like shows that the tweet was appreciated, but it doesn't necessarily mean it has been shared or spread as much as in the case of Retweets.
- 0.2 to Hashtags: Hashtags are an important tool for increasing the visibility of tweets on Twitter, as they allow users to categorize and group content. This makes it easier for users interested in specific topics to discover tweets. However, Hashtags have less impact on popularity than Retweets or Likes, so they were assigned a lower weight.

As a summary, we combine the factors: tf-idf plus the popularity score, based on Likes, Retweets, and Hashtags_count, multiplying each by its respective weight to compute the final score for the top-20 ranking.

2. Return a top-20 list of documents for each of the 5 queries, using word2vec + cosine similarity.

In this part the implement two different functions search_word2vec() and rank_documents_word2vec().

The search_word2vec() function processes a user query to find documents containing all query terms by intersecting document sets retrieved from an inverted index. After preprocessing and gathering relevant docId, it calls rank_documents_word2vec(), which ranks these documents using Word2Vec-based semantic similarity.

The rank_documents_word2vec() function builds document vectors from preprocessed tweets, computes an average vector for each document and the query, and uses cosine similarity to score and sort the documents by relevance. Both functions work together to efficiently identify, and rank documents based on the semantic meaning of the query, even handling cases where terms are missing.

Finally, we executed the search_word2vec() function for the defined queries. The highlights of the results obtained are as follows:

- Query "Indian government response to farmers".

2719 documents were retrieved, with doc_8866 having the highest score of 10.88 and doc_35055 having the lowest score of 10.75 in the top 20.

- Query: "International support for farmers".

443 documents were retrieved, with doc_9046 having the highest score of 12.86 and doc_6920 having the lowest score of 12.50 in the top 20.

- Query: "Demands of farmers' protests".

496 documents were retrieved, with doc_10292 having the highest score of 13.89 and doc_48036 having the lowest score of 13.56 in the top 20.

- Query: "Police action during farmers protest".

1448 documents were retrieved, with doc_1719 having the highest score of 15.44 and doc_14502 having the lowest score of 14.88 in the top 20.

- Query: "Schedules and locations of demonstrations and protests".

16 documents were retrieved, with doc_32308 having the highest score of 14.73 and doc_45516 having the lowest score of 12.94 in the top 20.

3. Can you imagine a better representation than word2vec? Justify your answer.

Transformer-based embeddings, such as those produced by models like BERT (Bidirectional Encoder Representations from Transformers) or GPT (Generative Pre-trained Transformer), have transformed the field of natural language processing. They offer significant advancements over traditional embeddings like Word2Vec, Doc2Vec, and Sentence2Vec. While the use of transformer-based embeddings for information retrieval and ranking tasks offers numerous improvements, it also introduces certain complexities compared to traditional embeddings.

Improvements:

- Contextual Awareness: Transformers capture the context of words by considering their surrounding words in the text, leading to a deeper understanding of word meanings and nuances. This significantly enhances retrieval accuracy.
- Semantic Matching: The ability to understand the specific context in which a word appears allows transformers to identify synonyms, antonyms, and other contextually related terms, thus improving semantic matching.
- Multi-modal Capability: They can handle multi-modal inputs, such as text, images, and audio, enhancing their versatility across different types of data.

- Extensive Training: Since transformers are trained on large-scale datasets, they are more robust and can handle a broad range of topics and languages effectively.

Complexities:

- High Computational Requirements: Transformers demand significant computational resources for inference, which can pose challenges for real-time or large-scale retrieval tasks.
- Deployment Challenges: Due to their large model sizes, deploying transformer-based systems can be cumbersome and requires considerable memory and storage capacity.
- Increased Latency: The inherent complexity of these models can lead to delays in the retrieval process, making them less ideal for time-sensitive applications.
- Data and Resource Intensive: Leveraging transformers effectively often requires access to large datasets, pre-trained models, and substantial computational power, which can be prohibitive.

Despite the substantial improvements in contextual and semantic understanding offered by transformer-based embeddings—making them ideal for enhancing retrieval outcomes across diverse applications—their high computational demands, large size, and complexity can hinder smooth integration. Thus, whether to use traditional embeddings like Word2Vec or transformer-based models should be determined by the specific requirements and resource availability of the intended retrieval application.

Link to the GitHub repository: https://github.com/enekotrevi/IRWA-2024