Report Part 3: Ranking

1. INTRODUCTION

The objective of this phase of the project is to build a retrieval and evaluation system for a collection of tweets. This system will index tweets and rank them based on their relevance to specific queries using different scoring methods: TF-IDF, BM25, and a custom score that considers tweet popularity (likes and retweets). Ranking methods will be evaluated and compared analyzing their strengths and limitations. Additionally, tweets will be represented using word2vec embeddings, with an exploration of alternative representations like doc2vec or sentence2vec to enhance thematic grouping and interpretation.

2. RANKING

2.1 TF-IDF + cosine similarity

The rank_tweets_tfidf function takes as input the query terms (terms), the IDs of tweets that contain at least one of those terms (tweet_ids), the inverted index (index), the TF-IDF scores for each tweet (tfidf_scores), and the lengths of each tweet vector (tweet_lengths).

First, a query vector (query_vector) is created, where each position represents a query term, and its TF-IDF weighting is calculated. For each relevant tweet (tweets that contain any of the query terms), a tweet vector (tweet_vectors) is generated with the corresponding TF-IDF scores for the query terms.

Then, the cosine similarity between the query vector and each tweet vector is calculated. Tweets are sorted in descending order based on their cosine similarity scores, and the IDs of the relevant tweets are returned.

Finally, for each query (query_terms), the system identifies tweets that contain at least one of the query terms and calls the rank_tweets_tfidf function to obtain tweets ranked by relevance.

2.2 Popularity Score + cosine similarity

In this part, we have created a new metric to rank tweets based on popularity metrics (likes and retweets) and cosine similarity. The idea is to prioritize tweets that are both relevant to the query and popular among users.

First we calculate the popularity score based on likes and retweets and we normalize this score between 0 and 1. Then, we compute the cosine similarity between the query vector and each tweet vector based on the tf-idf weights. The final score is a combination between the cosine similarity and the popularity score, ensuring that tweets that are relevant are popular are in a high position in the ranking.

This approach is useful since it considers not just the relevance and also considers the social significance. However, the downside can be that popularity metrics may not always align with the quality or relevance of content, leading to potential bias towards more viral, but less relevant, tweets.

2.3 BM25

The rank_tweets_bm25 function receives the query terms (terms), the IDs of tweets containing at least one of those terms (tweet_ids), the inverted index (index), the tweet lengths (tweet_lengths), the average tweet length in the collection (avg_doc_len), and the BM25 tuning parameters, k1 and b.

First, it calculates the IDF for each term in the query. The IDF of each term is calculated using the BM25 formula, which considers the total number of tweets (total_tweets) and the document frequency of the term (doc_freq).

Next, for each tweet in tweet_ids, a score is initialized to accumulate the tweet's relevance to the query terms. For each term in the query, if the tweet contains that term, the term frequency in the tweet (term_freq) is calculated, and the BM25 TF component is applied. The BM25 score for the term is calculated by multiplying the term's IDF by the TF component, and this score is added to the tweet's total score.

Tweets with a positive score are stored in the tweet_scores dictionary. The tweets are then sorted in descending order based on their score. The function returns a list of tweet IDs ordered by relevance, from highest to lowest score.

For each query (query_terms), the system identifies the set of tweets containing at least one of the query terms. Then, it calculates the average tweet length (avg_doc_len) to use in the BM25 calculation. The rank_tweets_bm25 function is called to obtain the tweets ranked according to their relevance to the query.

2.4 Comparison between TF-IDF and BM25

The main difference between TF-IDF and BM25 in document ranking lies in how they handle term frequency and document length, which leads to distinct outcomes in the relevance of results for queries.

Differences in Ranking Between TF-IDF and BM25

- *Handling of Term Frequency*
  -TF-IDF: Treats term frequency linearly, meaning that the more a term appears in a document, the higher the score. However, this can lead to diminishing returns in capturing term importance as frequency increases since there is no normalization.

  -BM25: Uses a more refined term frequency component with a saturation effect, which means that after a certain point, the impact of term frequency on the score diminishes. This better reflects the real-world scenario where a term appearing 10 times is not necessarily ten times more relevant than if it appeared once.

In our results, BM25 gave more nuanced scores to tweets with repeated terms, meaning it did not over-prioritize overly frequent terms within the same tweet.

- *Document Length Normalization*
  <u>-TF-IDF:</u> Lacks explicit document length normalization. This can lead to long documents being ranked higher if they naturally contain more query terms, even if those terms are less relevant.

  <u>-BM25:</u> Adjusts scores based on document length, using the average document length as a reference. Short tweets that are highly relevant to the query terms tend to rank higher, while longer tweets with diluted term relevance may rank lower.

  In our results, teh BM25 results where tweets that were shorter but highly relevant to the query appeared higher on the list.

Pros and Cons of TF-IDF and BM25

- *TF-IDF*
  <u>-Pros:</u> Simpler to compute and often sufficient for basic text retrieval tasks and works well when document lengths are relatively uniform. Term frequency does not vary significantly.

  <u>-Cons:</u> Lacks length normalization, making it less effective for datasets with varying document lengths. It can overemphasize documents with high term frequency without accounting for diminishing relevance.

- *BM25*
  <u>-Pros:</u> More sophisticated, as it incorporates both term saturation and document length normalization, leading to more accurate rankings. Well-suited for datasets with varying document lengths, like tweets, where short yet relevant documents may hold more importance.

  <u>-Cons:</u> Slightly more computationally intensive due to additional parameters (k1 and b) and normalization steps. Requires tuning of k1 and b parameters to achieve optimal performance, which can be time-consuming.

In conclusion, TF-IDF may work well for simpler, uniform datasets, while BM25 provides a more flexible and accurate approach for complex text corpora with diverse document lengths, as demonstrated in our tweet ranking results.

3. TOP-20 DOCUMENTS

We aim to identify the top 20 most relevant tweets for each of our 5 queries. Relevance is determined by generating vector representations of tweets and queries using Word2Vec, with cosine similarity used to measure the closeness between each query and the tweets.

First, “tweets” is extracted from processed_tweets, this prepares the data for the Word2Vec model to capture semantic relationships between words.

A *Word2Vec* model is trained using the list of tweets, transforming each word into a fixed-size vector (here, 100 dimensions) to capture semantic relationships. Specifically:

- vector_size=100: Defines the size of each word vector.
- window=5: Indicates the number of context words considered around each word.
- min_count=1: Specifies the minimum number of occurrences for a word to be included in the model.

The tweet_to_vec function processes each tweet to calculate its representative vector. To achieve this:

- It extracts individual word vectors (if they exist in the Word2Vec vocabulary).
- If no known words are found in a tweet, a zero vector is returned.
- If the tweet contains known words, it averages their vectors, resulting in a single vector representing the general meaning of the tweet.

Using “tweet_to_vec” we generate vectors for the queries using the Word2Vec model, allowing queries and tweets to be compared in the same vector space. For each query, the code calculates the cosine similarity between the query vector and each tweet vector, to be sorted in descending order where the top 20 tweets will be selected. Finally, for each query, the code displays the top 20 most relevant tweets, along with their original text and similarity score.

4. BETTER REPRESENTATIONS THAN WORD2VEC

While Word2Vec is useful for learning semantic representations at the word level, it doesn't always capture the full context of a text, especially for shorter texts like tweets. In the case of tweets, where the context and relationships between words are key to understanding the meaning, more advanced models are needed that consider the entire message, not just individual words. Two notable alternatives to improve tweet representation are Doc2Vec and Sentence2Vec, which aim to overcome the limitations of Word2Vec by providing richer, more context-sensitive representations for documents or sentences.

Doc2Vec

Doc2Vec is an extension of Word2Vec that learns vector representations not just for individual words but also for entire documents or sentences, making it especially useful for tasks where the full meaning of a tweet is more important than the individual words.

- Advantages of Doc2Vec for Tweets

  Captures the full context of the tweet: Doc2Vec learns a unique representation for each document (or tweet), which is ideal for short texts like tweets, where meaning is derived from the combination of words rather than individual words.

  Useful for tasks like classification or information retrieval: If the goal is to classify tweets or retrieve the most relevant ones for a query, Doc2Vec can be effective because it understands the full meaning of the tweet, not just the relationship of isolated words.

- Disadvantages of Doc2Vec

  Requires large text corpora: To train properly, Doc2Vec needs a large corpus. If the data is limited or the model is not well-trained, it may not capture the full semantic range of the tweets.

  Static representations: While it considers the context of the tweet, Doc2Vec's representations are still static. This means it can't adapt to the changing context of a tweet or the specific meaning of a word in a broader context.

Sentence2Vec

Sentence2Vec is a more recent approach that builds vector representations for entire sentences, capturing the full semantic meaning of the sentence by using more advanced techniques like transformer-based models (such as BERT, RoBERTa, and ALBERT). These models revolutionized natural language processing (NLP) by considering contextualized embeddings, meaning the representation of a word (or sentence) can change depending on the surrounding words.

- Advantages of Sentence2Vec for Tweets

  The main advantage of Sentence2Vec is its ability to generate dynamic representations based on context. This means that the meaning of a word (and thus the meaning of a sentence) can vary depending on the surrounding words. For example, the word "bank" will have different representations depending on whether it’s used in the context of a “financial institution” or “the edge of a river.”

Through transformer models like BERT, Sentence2Vec captures not just the meaning of individual words but also the relationships between them within a sentence. This is crucial for tasks like sentiment analysis, sarcasm detection, or semantic textual similarity, which are common in tweets.

Transformer-based models can be fine-tuned on a specific corpus, like tweets, allowing the model to learn the unique characteristics of this type of text.

- Disadvantages of Sentence2Vec

  Transformer-based models like BERT require significant computational resources, especially for training and inference. This can be a challenge when working with large amounts of tweets.

  Pre-trained models like BERT or RoBERTa are quite large (several gigabytes), which can make loading and inference slow, making them difficult to implement in real-time applications or on platforms with limited resources.

  While pre-trained models are powerful, fine-tuning for tweets requires an adequate amount of domain-specific data. If the dataset is small, fine-tuning may not significantly improve performance over the general-purpose pre-trained models.

Comparison between Doc2Vec vs Sentence2Vec

Sentence2Vec, based on transformer models, has a clear advantage in capturing dynamic context. Tweets are highly context-sensitive, and Sentence2Vec's ability to generate contextualized representations makes it much more suitable for tasks where semantic nuances matter, such as sentiment analysis or tweet classification. Doc2Vec, while useful, cannot capture these subtleties as effectively.

Doc2Vec is more efficient computationally, especially when dealing with limited resources. If the goal is a simple task like basic tweet classification, Doc2Vec may be preferred. However, for more complex tasks or when precise semantic representation is needed, Sentence2Vec is superior.

Sentence2Vec generally offers a richer and more nuanced understanding of the content of a tweet, considering not only the individual words but also how they interact within the full context of the sentence. Doc2Vec, on the other hand, focuses on the broader context of the tweet but its representations are static and unable to adjust to the immediate context of the sentence.

Conclusion

While Word2Vec is useful for learning word-level representations, both Doc2Vec and Sentence2Vec offer better alternatives when it comes to tweets. Doc2Vec improves by learning document-level vectors, making it useful for tasks where the full context of the tweet matters. However, Sentence2Vec, using transformer models like BERT, provides more dynamic and contextualized representations that are especially helpful for tasks requiring deep semantic understanding, such as classification or sentiment analysis of tweets. Despite the computational challenges, Sentence2Vec holds the greatest potential for obtaining precise, context-sensitive representations, especially in the modern era of tweet analysis.

5. CONCLUSION

The tweet ranking system successfully retrieves and ranks tweets based on their relevance to specific queries using a combination of TF-IDF, BM25, and popularity scores. The evaluation of these ranking methods demonstrates their strengths in handling term frequency, document length, and the social significance of tweets. Additionally, the use of word embeddings, particularly Word2Vec, enhances the thematic grouping of tweets, while exploring alternative representations like Doc2Vec and Sentence2Vec reveals their potential for improving context-sensitive understanding. Overall, the system delivers accurate and meaningful rankings, with opportunities for further improvement through advanced embedding techniques.

6. FUTURE WORK

Future work could focus on fine-tuning embedding models like BERT or RoBERTa for better context-sensitive tweet representations. Popularity metrics could be refined by incorporating additional social engagement indicators, and query expansion techniques could improve retrieval accuracy. Optimizing BM25 parameters and exploring learning-to-rank models would enhance ranking performance. Real-time processing capabilities could be improved to handle large tweet volumes efficiently. The system could be expanded to support multilingual tweets by using models like mBERT or XLM-R. Additionally, incorporating stream processing techniques would allow for near-instantaneous ranking and retrieval. These improvements would lead to a more accurate, scalable, and context-aware tweet ranking system.

GITHUB Repository: *https://github.com/davidobrero/IRWA-2024-G102-10*
All related with Part 3 can be found inside the folder IRWA-2024-PART-3 containing:

- This report (IRWA-2024-PART-3-REPORT)
- The code (IRWA-2024-PART-3-CODE)

It is possible to open the code with Google Colab, connecting the data to Google Drive.