Information Retrieval and Web Analytics: Part 3 Project Report

Nicolas Vila, David Pérez, Guillem Escriba

November 2023

1 Scoring

The goal of the ranking score is to get the top-20 documents that are related to the query. Our proposed solution lies in the intersection of state-of-the-art transformer based embeddings and using likes and retweets retrieved from the vectors. The main improvement here is the usage of transformer-based embeddings which capture the semantics of the tweet in a richer way than any other method we have used is able to do. It can capture context, similarity of words that are not exactly the same but share the same semantics, etc. In addition to that, we add a normalized score based on likes and retweets that further enhances the retrieved tweets. The way we are currently using the embeddings is by building an embedding index mapping tweet information to their corresponding embedding. This is only done once, and can be saved as a .csv file for future use. The user can provide any query and our model will compute the embedding of the given query and perform a cosine similarity with all tweets. On top of that cosine similarity, we add another score based on normalized likes and retweets. The specific information is in embeddings_score.py. We have to say that finding the proper values to weight the likes and retweets was done by trial and error -while ensuring that they were on a comparable scale to the cosine similarity-. We have found the results of this retrieval method qualitatively better than the other methods we have tried.

The cons of this model are that it doesn't ensure that all the terms in the query will be present in the tweet -which we don't believe it is necessarily a bad thing- and that to compute the embeddings we need to use a GPU, given that transformer models are slow on cpu. The results for the queries retrieved by this scoring method are stored in our GitHub. As future work we would propose to add the days since the posting of the tweet, in such a way that it would penalize older tweets.

When it comes to testing both rankings, we tried several queries, such as 'Ukraine' or 'missile attacks'. For the second one we received the most insightful results. Since our score takes into consideration the context and the nuances of the text. For example, our score considers tweets with 'launch' or 'strikes' instead of attack or other synonyms or types of missiles. Since our score considers the semantics, we get more realistic rankings because it is not excluding all the tweets that have not the same exact words. Also, our score weighs the tweets according to the likes or retweets, so we notice how the top ranked tweets from the traditional score have almost no likes or retweets, but the top ranked tweets according to our score contain several likes and retweets. We have chosen low weights, since otherwise they could bias the ranking by choosing the most popular tweets over the rest of them, which from our point of view does not necesarily mean more relevant tweets.

2 Word2Vec

The implementation of our 'Word2Vec' rankings, or rather 'Tweet2Vec' works as following. We first compute a data structure that stores the tweet vector for each tweetId, in that way, we don't need to recompute the vectors for each query. Secondly, there is a loop that asks the user for an input. For each given input, we obtain the terms of the query that are in our index, and retrieve only those documents that contain all the terms. Subsequently, we retrieve a subset of our vector data structure to retrieve only those tweets that matched all terms in the query, and perform cosine similarity only with that subset. Finally, we write the top 20 results in a text file.

Between all of our 5 queries, we only got matching tweets for the first one, since it was the only one where all the query terms - present in our index- were also in the tweet that was retrieved. This makes sense because in our case, our queries are a bit large for this rather simple model of retrieving tweets. When a query is too long, it gets increasingly more rare that the model will return any matching query. That is because of what I just mentioned, which is that we only retrieve tweets that have all the terms of the query -present in the index-.

3 Understanding the Advantages and Disadvantages of Transformer-Based Retrieval Engines Compared to Simpler Models

When considering transformer-based embeddings for information retrieval, exemplified by models like RoBERTa, OpenAI's embeddings, or Llama v2, they offer a robust approach to text embeddings due to their complexity and ability to capture nuances and meaning in smaller pieces of text, such as tweets. These models are extremely good at capture meaning inside a paragraph, which is something that a model like word2vec could never perform, because it computed a vector for each word, without taking into account context. However, for longer texts, these models may struggle to process the entire meaning in a single embedding. This increased complexity, while immensely beneficial, comes with associated costs compared to traditional embeddings.

On one hand, running transformer-based embeddings at scale requires substantial computational power, particularly for state-of-the-art models with billions of parameters. Additionally, fine-tuning such models for specific tasks can be a non-trivial and costly endeavor. Fine-tuning billion-parameter models involves significant expenses, especially when fine-tuning all parameters in the Language Model (LM). Some approaches, like LoRA, focus on fine-tuning only the most relevant part of the model for a specific task, typically less than 1

On the other hand, simpler models like word2vec are more cost-effective to run but exhibit lower performance compared to transformer-based embeddings, limiting their applicability to various tasks.

Finding the right balance or trade-off for a specific use case is crucial. The decision depends on the trade-offs between the advantages and disadvantages of each model, considering factors such as computational requirements, fine-tuning complexity, and overall performance.

There is a trend toward using pre-built models provided by companies like OpenAI, making complex models more accessible. However, the affordability of utilizing these models for fine-tuning or other specific needs remains a challenge. While the availability of complex models may increase through platforms like OpenAI, the cost implications for individual fine-tuning of open-source models can still be very expensive.