Part 3: Ranking

Abril Masgrau Turró, Jorge Martínez Rosa and Enric Riba Sáez
Information Retrieval & Web Analytics
University Pompeu Fabra

Ranking

0) Introduction

First of all, we have changed some of our queries of the previous part because we did the union of the documents having some of the query words, but in fact, it should be the intersection (they should contain all the words of the query). Consequently, some of the sets are empty or have very few documents, which affects the search results of the queries. Then, these are our new queries (only Q3 and Q5 are new):

- Q1 = United States
- Q2 = trump russia
- Q3 = Russia attack
- Q4 = World War
- Q5 = Nazi

Now, about the goal of the lab, we have to find, sort and show the 20 documents with the highest score with regard to our queries.

1)

To do the ranking, we have done it in two different ways: using TF-IDF + cosine similarity (rank_scores_tf_idf) and using a formula created by us using some popularity criteria + cosine similarity (rank_scores_popularity).
To do this, we first compute the TF-IDF+cosine_similarity as we did in the previous part, but then, we involve the popularity of the tweet by the following formula:

total_score = previous_score*lambda + popularity_score*(1-lambda).

Where popularity_score is:

popularity_score = log(likes+retweets+1)

With lambda=0.5, we get scores that affect the total score.

With this new ranking method, we combine two different scores and with the lambda parameter, we modify the weight of each score. So they differ that TF-IDF doesn't consider popularity and our method does.

The **pros** of doing this new method is that now that we have added the popularity, the ranking is more useful for cases where we are interested not only in relevant documents but also on popular ones.
This score is also affected by trends or behaviors of the population so it adapts to the moment in time.
Also it has the parameter lambda that can be adjusted to give more weight to the different scores.
It also has the pros that the TF-IDF have plus the ones mentioned.

The **disadvantages** of our method are that popularity doesn't always mean relevance. A tweet can be very popular with many likes and retweets but not be relevant. Also, the term popularity may be ambiguous as there are many metrics to compute popularity, not only likes but also comments, shares… .

2)

These are the top-10 lists of documents for the 5 queries (top-20 in Jupyter notebook):

Q1 = United States

doc_id	1847	1747	613	3645	647	3931	630	3665	429	2635
predicted_score	1.753	1.105	0.996	0.981	0.838	0.81	0.7956	0.718	0.7158	0.669

Q2 = trump russia

doc_id	3089	859	3447	3464	3429	3399	2660	3813	2613	3073
predicted_score	0.8682	0.862	0.833	0.817	0.795	0.768	0.753	0.7451	0.7156	0.6573

Q3 = Russia attack

doc_id	249	1419	2367	1244	111	1404	3535	2032	322	491
predicted_score	1.2806	1.233	1.160	1.060	0.957	0.914	0.8830	0.876	0.8700	0.849

Q4 = World War

doc_id	1540	222	204	3414	1632	2333	1644	2648	1462	3088
predicted_score	0.9477	0.919	0.857	0.809	0.739	0.7381	0.7029	0.6712	0.6581	0.6480

Q5 = Nazi

doc_id	2954	3708	3286	1162	2722	136	876	2439	468	75
predicted_score	1.7750	0.923	0.862	0.579	0.554	0.5366	0.5183	0.5054	0.4831	0.4748

As we can see, common terms in queries related to the Ukraine-Russian War have higher similarity scores between documents terms.

We have used the code of the previous deliverable part to represent the tweet2vectors for the top-20 documents of every query. The only thing we have modified is the value of the variable "perplexity" which represents the number of nearest neighbors, and the new value is 5.

Now, we show here the plots obtained with the 5 queries:

Q1 = United States

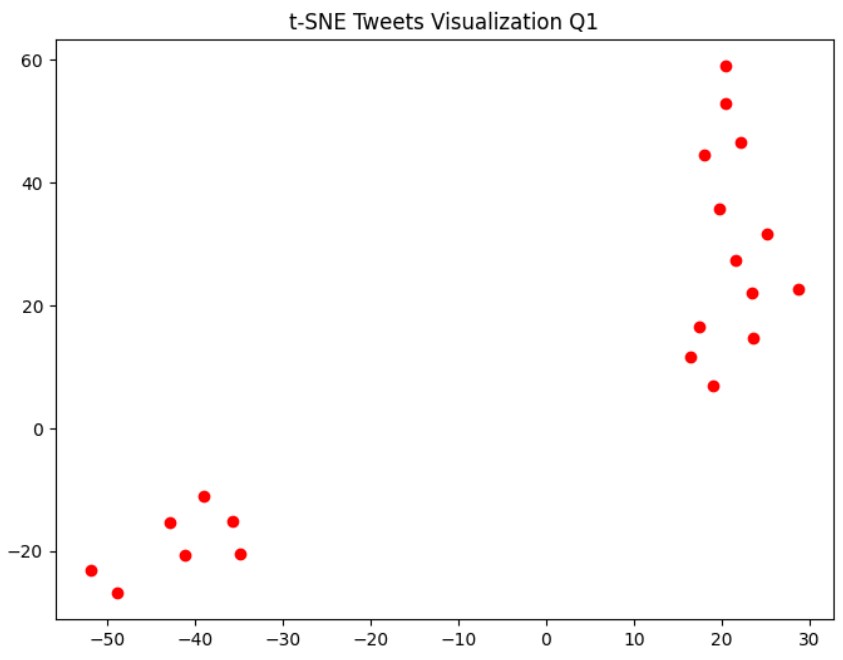

Figure 1: Scatter plot t-SNE query 1

As we can see, there are only two clusters of terms that are very far from each other (one of them with negative values in both axes and the other one with positive values).

Q2 = trump russia

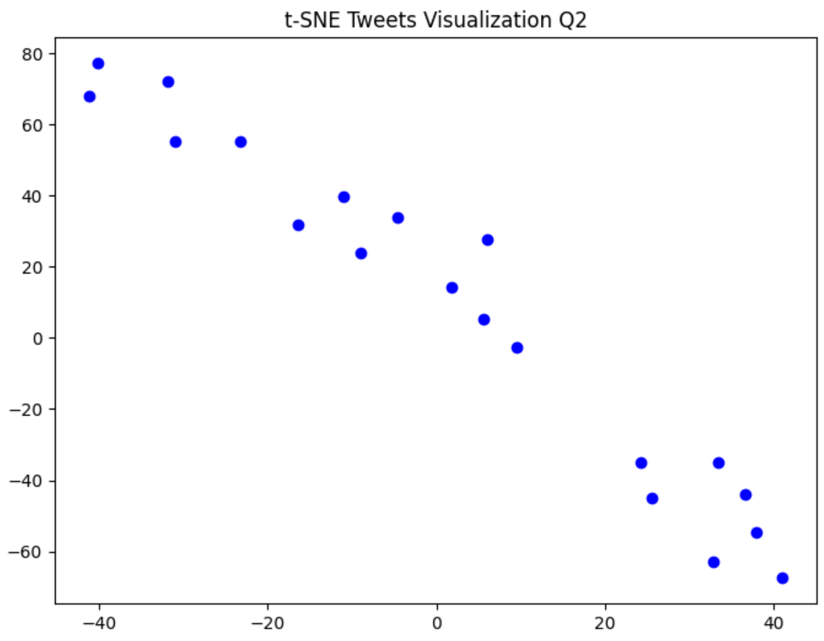

Figure 2: Scatter plot t-SNE query 2

As we can see, there are only three clusters of terms that seem to be in the same line or have a similar distribution.

Q3 = Russia attack

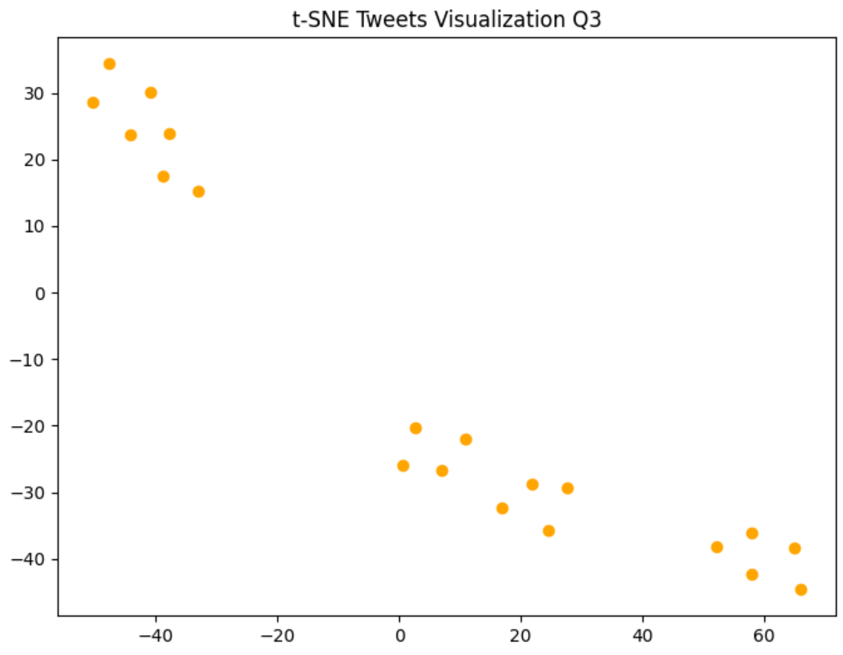

Figure 3: Scatter plot t-SNE query 3

As we can see, there are only three clusters of terms that seem to follow a similar pattern or distribution, where two of them are closer and the other one is isolated (the first ones have positive values in one axis while having negative values in the other, and the last cluster has the opposite).

Q4 = World War

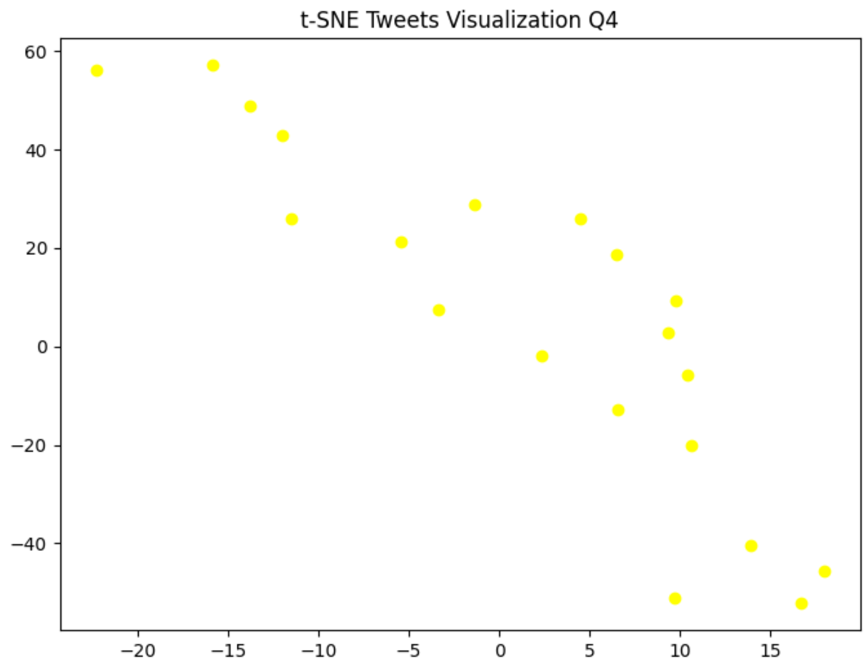

Figure 4: Scatter plot t-SNE query 4

As we can see, there are no well-defined clusters of terms, or they are a little bit sparse. Although, they seem to be in the same line or have a similar distribution.

Q5 = Nazi

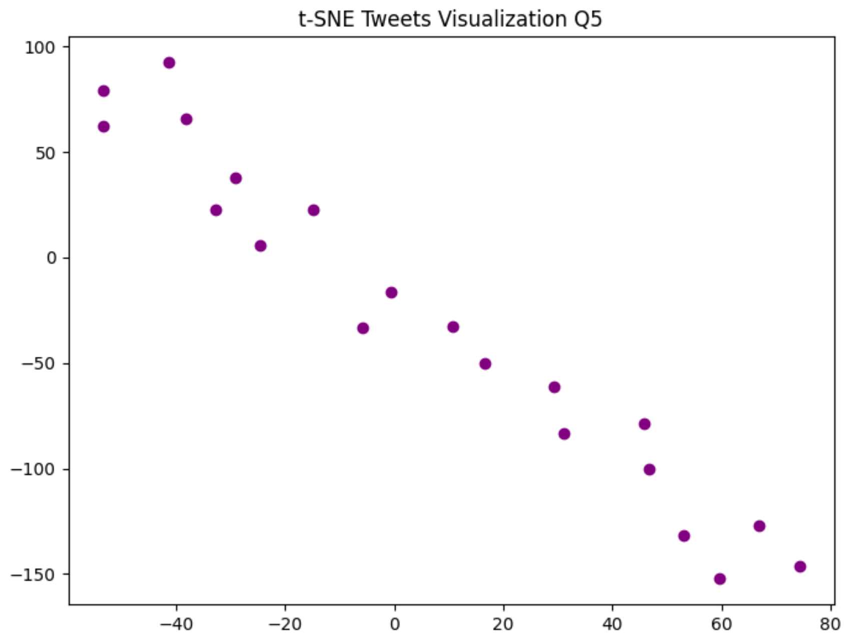

Figure 5: Scatter plot t-SNE query 5

As we can see, there are no well-defined clusters of terms, or they are a little bit sparse. Maybe, you can identify six small clusters that are next to each other. Although, they seem to be in the same line or have a similar distribution.

3) In the context of Information Retrieval, where cosine similarity between vectors is often employed to retrieve relevant documents, how might transformer-based embeddings (such as those from BERT or ROBERTa) enhance or complicate the retrieval process compared to traditional embeddings like word2vec? Consider factors like capturing context within tweets, computational overhead, the potential need for fine-tuning, and the richness of embeddings in representing the nuances of short texts like tweets.

Word2Vec models generate embeddings that are context-independent (each word is represented by a single numeric vector). Different senses of the word (if any) are combined into one single vector.
Word2Vec embeddings do not take into account the word position.

However, the BERT model generates embeddings that allow us to have multiple (more than one) vector (numeric) representations for the same word, based on the context in which the word is used. This variation arises from the context in which the word is employed, highlighting the context-dependent nature of BERT embeddings.
BERT model explicitly takes as input the position (index) of each word in the sentence before calculating its embedding

References:

Gupta, L. (2022, 21 septiembre). Difference between Word2Vec and BERT | the startup.Medium.
https://medium.com/swlh/differences-between-word2vec-and-bert-c08a3326b5d1

GitHub Repository: JorgeMRPOO/IRWA-2023-part-1 (github.com)