Github: https://github.com/dani-upf/IRWA-2022-u171404-u172940-u173482

Indexing and evaluation

In this part of the project we are going to create an inverted index and rank some queries with the tf-idf algorithm. Then, we will evaluate the ranking for well-known queries with some of the most common metrics in Information Retrieval. The source code can be accessed via the github repository, and the README.md from the previous delivery has been updated with comprehensive instructions on how to access the datasets we are using.

Indexing

The first thing we did was building an inverted index:for each processed tweet we enumerate the terms and store them in a dictionary of the form "term":[["doc_id"],array["positions_of_the_term"]. This is a dictionary created for each tweet. At the end of each iteration these are merged with the main index, so at the end we have the inverted index in the form of a dictionary ordered alphabetically.

{Term_id_1: [[document_1, term positions in doc1], [doc2, positions doc2]...],

Term_id_2: [[document_4, term positions in doc4], [doc6, positions doc6]...],

etc...}

Now we should implement the tf-idf algorithm for 5 proposed queries which are based on terms that might be interesting for the tweets content:

q1: "Last hurricane in Florida"

q2: Terrible hurricane north carolina"

q3: "Medical assistance Florida"

q4: "Last news report Ian"

q5: "Dealing with east coast tragedy"

Note that we also preprocessed the queries so they fit our processed model.

The smartest way of computing the tf-idf of the document terms would be only considering the terms that appear in the query, since by cosine similarity definition, if a term does not appear in a query its query tf-idf will be 0 and therefore this will have no effect on the cosine similarity. However, we did not know which queries we were going to use and wanted to play with various options so we decided to firstly compute all the tf-idf of the document terms.

To do it we iterated through all the terms in the inverted index and computed the document frequency (df), raw frequency (f) and total number of documents in the collection as len(mapping_dict). The rest was applying the tf-idf formula: $w_{i,j} = (1 + log f_{i,j}) * log(N/df_i)$ and we ended up with the vector representation of the documents (if a term had frequency 0 the result should be 0).

Then we did the same for the queries and after having the vector representation of them we were able to compute the cosine similarity.

For each of the queries we had the document vector representation and the query vector representation, therefore we only needed to apply the cosine similarity formula $cos(q,d) = (q * d)/|q||d|$ and rank them in descendent order to get the top K documents.

Evaluation

In this section we will evaluate our algorithm. First of all, we will present the 2 main evaluation components:

1. As judges, we will analyze our previous queries, giving a 1 if the tweet returned is relevant for the given query and 0 else.

   q1: “Last hurricane in Florida” [0 1 1 1 1 1 0 1 1 0 1]

   q2: Terrible hurricane north carolina" [1 1 1 1 1 1 1 1 1 0]

   q3: "Medical assistance Florida" [1 1 1 0 0 1 1 1 1 1]

   q4: "Last news report Ian" [1 0 1 1 1 1 1 1 1 1]

   q5: "Dealing with east coast tragedy" [1 1 1 1 1 1 0 1 0 1]

2. We are given some queries and ground truth files evaluating these queries.

As we saw in class, we need:

- A collection of documents (representative of the reality)
- A collection of information needs (representative of the needs we expect to see)
- Human relevance assessments

Therefore, we are assuming that these documents and queries are representative of reality. The evaluation metrics will be computed using only the 2nd component, that is the given queries and the ground truth file.

To compute each evaluation metric we will proceed as follows: for each query we have some evaluated documents, but these are not the same for each query. Therefore we will compute the top k cosine similarity ranking considering only the documents we have information about (if q1 had truths for doc1,doc4,doc10 and doc12, we would only rank these 4 documents and forget about the others, so the metrics can be computed).

After it we return the top 10 lists for each query and compute the different metrics using the ground truth information. The pandas dataframe “query_info” stores all the necessary information to compute these metrics, and it is structured in a convenient way for fitting the metric functions.

Evaluated Queries

Q1: Landfall in South Carolina

Q2: Help and recovery during the hurricane disaster
Q3: Floodings in South Carolina

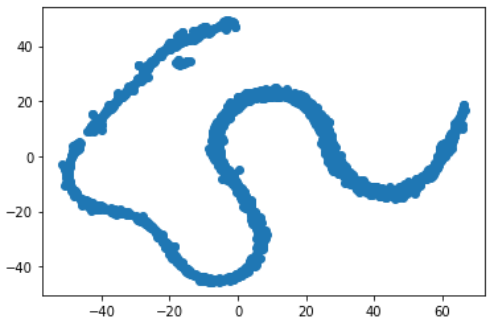
Queries	P@10	R@10	AvgP@10	F1-Score	MAP	MRR
Q1	1.0	0.5	1.0	0.6666	0.8598	1.0
Q2	0.9	0.45	0.8521	0.5	0.8598	1.0
Q3	0.8	0.4	0.8333	0.5333	0.8598	1.0

NDCG:

Q1: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Q2: [1.0, 0.5, 0.62, 0.68, 0.72, 0.75, 0.77, 0.78, 0.8, 0.81, 0.86, 0.86, 0.86, 0.86, 0.86, 0.86, 0.86, 0.86, 0.86, 0.86]
Q3: [0.0, 0.5, 0.62, 0.68, 0.72, 0.75, 0.68, 0.71, 0.73, 0.74, 0.80, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85]

Overall we have seen that the tf-idf algorithm is quite good (at least with this type of data and queries) and we have obtained very good results for each of the metrics, specially in precision since the gathered results are mainly relevant. However, there are still some caveats (which may be better approached by BM-25) like not considering position in text, semantics, etc so for example, some words with high tf-idf could not have any relation at all with the topic of the document.

Finally we choose Word2Vec to represent the tweets in a 2D scatter plot applying the T-distributed Stochastic Neighbor Embedding algorithm. We gather the tweets (already cleaned) of the json_processed array, put them through Word2Vec and take the vocabulary and fit it in the TSNE algorithm.



Word2Vec helps us embed the tweets in order to represent them as real number vectors. Then, TSNE helps us to visualize the high-dimensional data by converting similarities between data points to joint probabilities in the two-dimensional map. We cannot clearly define categories, but we can observe a correlation between the embeddings.