IRWA FINAL PROJECT

PART 2

INDEXING AND EVALUATION

Jordi Guillén: u198641
Anira Besora: u189647
Ana Cereto: u199767
Marina Castellano: u188311

Github/ Tag: https://github.com/Marinagrup2/IRWA.git

1. Indexing

We start by preprocessing the tweet data with the functions we did in the Part 1 of the project. We also added to our `preprocess_tweet` function the repetition of hashtags we mentioned on part1 so those have more weight and the importance of the hashtags is present.

From the dictionary that contains only the keys we want to keep (plus the separated hashtags), we created the function `create_index_tfidf` where we build the inverted index storing for each term the doc_ids of where you can find it and the position in this doc to then, can calculate the term frequency and idf as we saw in class. From the function we defined in class, we modified the title index to return the keys we kept from the tweets, we added the preprocessing step and added the page id as the doc_id from the key “id”.

To rank these documents we start by computing the norm of the query we want to search with the frequency of each term. For each term of the query, we compute the tf*idf and generate the doc_vectors for the matching docs that have that term. Finally we return the ranking of the docs for that query after calculating the scores of those.

With the `search_tf_idf` function we search which docs have all the terms of the query and we execute the ranking function of those docs so the user can see with their query which tweets are the higher ranked.

Queries:

To create our queries, firstly we printed the top 20 most frequent words which are mostly the hashtags we repeated to showcase the importance and some other words
['farmersprotest', 'farmer', 'india', 'releasedetainedfarm', 'indiabeingsilenc', 'farmersmakeindia', 'disharavi', 'support', 'mahapanchayatrevolut', 'repealonlywayahead', 'protest', 'pagdisambhaljatta', 'amp', 'standwithfarm', 'modiignoringfarmersdeath', 'railrokoforfarm', 'istandwithfarm', 'msplawforallcrop', 'modi', 'freenodeepkaur']

From this words, we decided on the following queries:

1. Farmers protest
2. Farmers rights
3. India is being silenced
4. Detained farmers
5. Support farmers revolution

2. Evaluation

Based on the queries we have selected, we have analyzed:

- ➔ Precision
- ➔ Recall
- ➔ Average precision
- ➔ f1-measure
- ➔ Mean Average Precision (MAP)
- ➔ Mean Reciprocal Rank (MRR)
- ➔ Normalized Discounted Cumulative Gain (NDCG)

The ground truth we have set was a bit random. The results show that:

```
Python
Metrics for query 'Farmers protest': {'precision@k': 0.4, 'recall@k': 0.25,
'avg_precision@k': 0.6527777777777777, 'f1_score@k': 0.3076923076923077,
'rr@k': 1.0, 'ndcg@k': 0.8421}

Metrics for query 'Farmers rights': {'precision@k': 0.3, 'recall@k':
0.3333333333333333, 'avg_precision@k': 0.4761904761904762, 'f1_score@k':
0.3157894736842105, 'rr@k': 0.5, 'ndcg@k': 0.6546}

Metrics for query 'India is being silenced': {'precision@k': 0.5,
'recall@k': 0.2, 'avg_precision@k': 0.4323809523809524, 'f1_score@k':
0.28571428571428575, 'rr@k': 0.3333333333333333, 'ndcg@k': 0.6189}

Metrics for query 'Detained farmers': {'precision@k': 0.4, 'recall@k': 0.25,
'avg_precision@k': 0.6, 'f1_score@k': 0.3076923076923077, 'rr@k': 1.0,
'ndcg@k': 0.8036}

Metrics for query 'Support farmers revolution': {'precision@k': 0.5,
'recall@k': 0.2, 'avg_precision@k': 0.5053968253968254, 'f1_score@k':
0.28571428571428575, 'rr@k': 0.5, 'ndcg@k': 0.6812}

Mean Average Precision (MAP): 0.5333492063492064
```

Analyzing queries individually we observe:

- Precision varies between 0.3 and 0.5, showing some inconsistency in retrieving relevant documents.
- Recall values are generally low (ranging from 0.2 to 0.33), suggesting that a significant number of relevant documents are being missed.
- Ranking effectiveness, however, seems quite high in all queries.

In conclusion, our random declarations for the ground truth were not very precise. The Mean Average Precision (MAP) score is approximately 53.3%, which indicates that the retrieval effectiveness is slightly above average, but still not highly accurate. However, the system is effective at ordering relevant documents higher in the results list.

Scatter plot

We selected the TF-IDF vector representation with the TfidfVectorizer, which transforms a collection of documents (in our case the tweets content) in a matrix of the TF-IDF values. With this matrix, we apply T-SNE, where we select the 2 components of the plot and change the number of iterations to 500 (the default value is 1000 so we chose half) to optimize the model. Finally, we display the scatter plot with `tweets_tsne[:, 0]` and `tweets_tsne[:, 1]` where we can see the different tweets, each represented as a dot. These tweets seem to be concentrated in the middle, forming a cluster, which shows that they share the same topic and are kind of similar. Also, as we triplicated the hashtags their weight is higher than other words, so this increase in the weight of some words that some tweets have in common results in the closeness in the scatter plot, making them similar, as we can see in the following picture