Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

REPORT PART 2

Indexing and Evaluation

Github LINK and TAG: IRWA-2024-part-2

This second part of the project is based on the implementation of the first part, since we again use the processed tweets and the created tokens. Therefore, we will again use the code that is in the **Text Processing** section changing only one thing inside the `build_terms` function. That is, we have added a line to remove apostrophes because we encountered that when analyzing words like *people's,* we were not creating the token as expected. After this change, we have proceeded to create the inverted index, implement the TF-IDF algorithm to retrieve the ranked documents and use some evaluation techniques on different queries.

We are now going to explain each section of the code and what it contains.

Inverted Index

In this part we have first defined 3 functions:

- `def get_token_tweets(dic):` We have created this function to return a dictionary with the form doc_id → tokens. Therefore, it takes as input the dictionary that we have with all the preprocessed tweets *get_tweets* to keep only the clean tweets, that is, the tokens for each tweet. We have created this function to be able to be able to work easily when computing the inverted index and the tf-idf algorithm

- `def create_index(dic):` This function will allow us to create the inverted index. It takes as input a dictionary of tweets and returns another dictionary of the form term → list_of_documents. It computes, for each word in the tweets in which documents appears. This is done by taking the tokens of each document and going through these tokens and, for each term, we check if it has already appeared or not and we append the word with the current document id. As an example,

```
index, title_index = create_index(get_tweets)
print(index['farmer'])
```

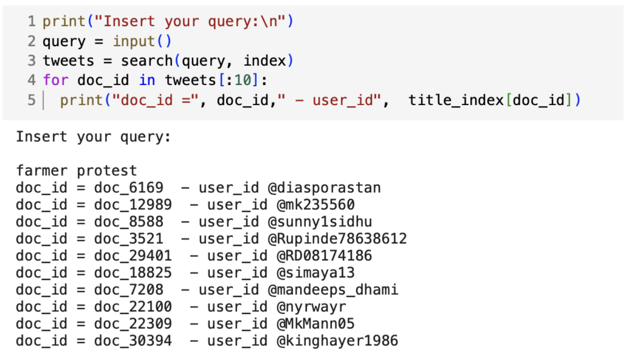
```
['doc_1', 'doc_5', 'doc_13', 'doc_14', 'doc_17', 'doc_22', 'doc_26', 'doc_41', 'doc_47', 'doc_59', 'doc_61',
```

- `def search(query, index):` This function will perform the search of documents given a query and the inverted index. Since we are dealing with conjunctive queries (AND), we need to retrieve the documents that contain ALL the words that form the query. First, we do this by creating a set with the documents that are in the inverted index of the first term of the query because, if not, when performing the intersection with the rest of documents, it will give an empty set. When we have the initial set, we go through the following terms in the query, take their inverted index and perform and & operation. All this will give us the common documents for the words that form the query.

Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

For example, if we take as input the query ‘farmer protest’, we can see all the documents where both words of the query appear.



```
print("Insert your query:\n")
query = input()
tweets = search(query, index)
for doc_id in tweets[:10]:
  print("doc_id =", doc_id," - user_id",  title_index[doc_id])
```

```
Insert your query:

farmer protest
doc_id = doc_6169  - user_id @diasporastan
doc_id = doc_12989  - user_id @mk235560
doc_id = doc_8588  - user_id @sunny1sidhu
doc_id = doc_3521  - user_id @Rupinde78638612
doc_id = doc_29401  - user_id @RD08174186
doc_id = doc_18825  - user_id @simaya13
doc_id = doc_7208  - user_id @mandeeps_dhami
doc_id = doc_22100  - user_id @nyrwayr
doc_id = doc_22309  - user_id @MkMann05
doc_id = doc_30394  - user_id @kinghayer1986
```

Inverted Index TF-IDF

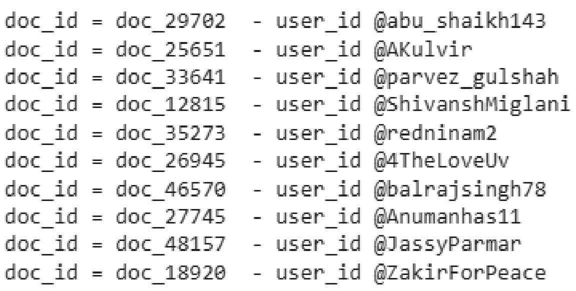
For this part we have created a function `def inverted_index_tfidf(query, inverted_index, total_tweets, clean_tweets)` This function creates a dictionary with the documents where the query appears and its ranked score.

To do so it first gets the query and applies the function build_terms to clean the words in the query; it also uses the function `search(query, inverted_index)` to find the documents where the query appears. Then it computes the tf-idf score for each word of the query in the documents where it appears and it adds the scores to the dictionary. When the dictionary has all the documents and its scores, we order it in descending order to have the ranked tweets.

Now we have selected 5 different queries to test our functions `def search(query, index)` and `def inverted_index_tfidf(query, inverted_index, total_tweets, clean_tweets)`. and see the differences between the two function’s outputs. The search function shows, in this case, the first 10 documents where the query appears, in contrast, the inverted_search_tfidf shows the top 10 documents with higher tf-idf score. Our fives queries are:

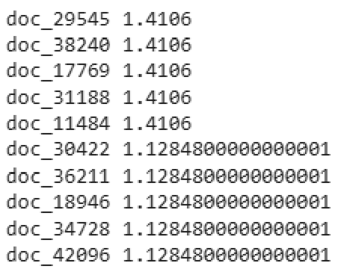
1. “*indian government*”

`def search`



```
doc_id = doc_29702  - user_id @abu_shaikh143
doc_id = doc_25651  - user_id @AKulvir
doc_id = doc_33641  - user_id @parvez_gulshah
doc_id = doc_12815  - user_id @ShivanshMiglani
doc_id = doc_35273  - user_id @redninam2
doc_id = doc_26945  - user_id @4TheLoveUv
doc_id = doc_46570  - user_id @balrajsingh78
doc_id = doc_27745  - user_id @Anumanhas11
doc_id = doc_48157  - user_id @JassyParmar
doc_id = doc_18920  - user_id @ZakirForPeace
```

`def inverted_index_tfidf`



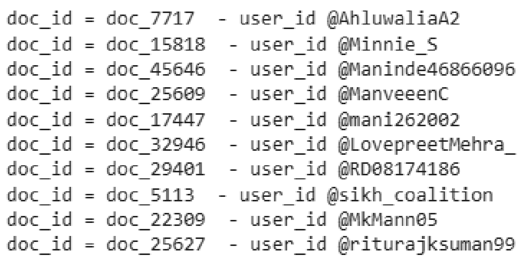
```
doc_29545 1.4106
doc_38240 1.4106
doc_17769 1.4106
doc_31188 1.4106
doc_11484 1.4106
doc_30422 1.1284800000000001
doc_36211 1.1284800000000001
doc_18946 1.1284800000000001
doc_34728 1.1284800000000001
doc_42096 1.1284800000000001
```

Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

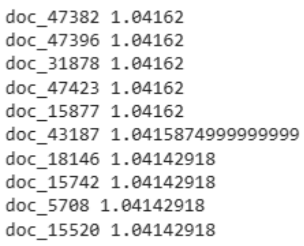
2. “support farmers”

`def search`



```
doc_id = doc_7717  - user_id @AhluwaliaA2
doc_id = doc_15818  - user_id @Minnie_S
doc_id = doc_45646  - user_id @Maninde46866096
doc_id = doc_25609  - user_id @ManveeenC
doc_id = doc_17447  - user_id @mani262002
doc_id = doc_32946  - user_id @LovepreetMehra_
doc_id = doc_29401  - user_id @RD08174186
doc_id = doc_5113  - user_id @sikh_coalition
doc_id = doc_22309  - user_id @MkMann05
doc_id = doc_25627  - user_id @riturajksuman99
```

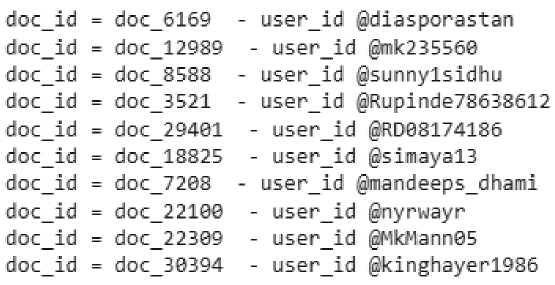
`def inverted_index_tfidf`



```
doc_47382 1.04162
doc_47396 1.04162
doc_31878 1.04162
doc_47423 1.04162
doc_15877 1.04162
doc_43187 1.0415874999999999
doc_18146 1.04142918
doc_15742 1.04142918
doc_5708 1.04142918
doc_15520 1.04142918
```

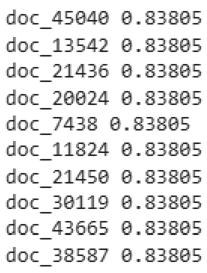
3. “farmer protest”

`def search`



```
doc_id = doc_6169  - user_id @diasporastan
doc_id = doc_12989  - user_id @mk235560
doc_id = doc_8588  - user_id @sunny1sidhu
doc_id = doc_3521  - user_id @Rupinde78638612
doc_id = doc_29401  - user_id @RD08174186
doc_id = doc_18825  - user_id @simaya13
doc_id = doc_7208  - user_id @mandeeps_dhami
doc_id = doc_22100  - user_id @nyrwayr
doc_id = doc_22309  - user_id @MkMann05
doc_id = doc_30394  - user_id @kinghayer1986
```

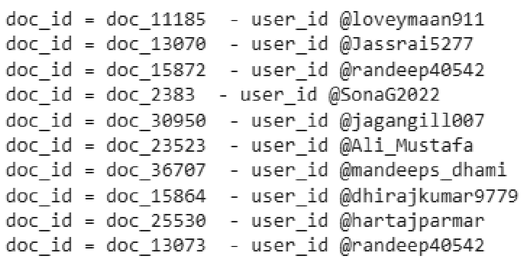
`def inverted_index_tfidf`



```
doc_45040 0.83805
doc_13542 0.83805
doc_21436 0.83805
doc_20024 0.83805
doc_7438 0.83805
doc_11824 0.83805
doc_21450 0.83805
doc_30119 0.83805
doc_43665 0.83805
doc_38587 0.83805
```

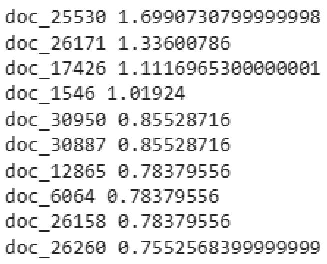
4. “agricultural sector”

`def search`



```
doc_id = doc_11185  - user_id @loveymaan911
doc_id = doc_13070  - user_id @Jassrai5277
doc_id = doc_15872  - user_id @randeep40542
doc_id = doc_2383  - user_id @SonaG2022
doc_id = doc_30950  - user_id @jagangill007
doc_id = doc_23523  - user_id @Ali_Mustafa
doc_id = doc_36707  - user_id @mandeeps_dhami
doc_id = doc_15864  - user_id @dhirajkumar9779
doc_id = doc_25530  - user_id @hartajparmar
doc_id = doc_13073  - user_id @randeep40542
```

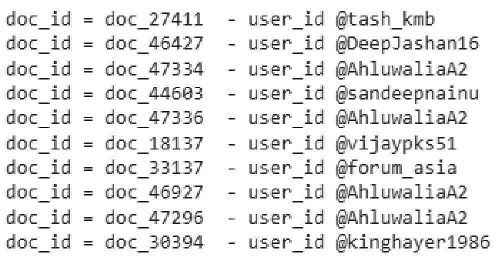
`def inverted_index_tfidf`



```
doc_25530 1.6990730799999998
doc_26171 1.33600786
doc_17426 1.1116965300000001
doc_1546 1.01924
doc_30950 0.85528716
doc_30887 0.85528716
doc_12865 0.78379556
doc_6064 0.78379556
doc_26158 0.78379556
doc_26260 0.7552568399999999
```

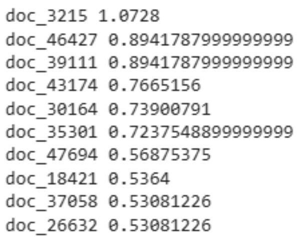
5. “indian rights ”

`def search`



```
doc_id = doc_27411  - user_id @tash_kmb
doc_id = doc_46427  - user_id @DeepJashan16
doc_id = doc_47334  - user_id @AhluwaliaA2
doc_id = doc_44603  - user_id @sandeepnainu
doc_id = doc_47336  - user_id @AhluwaliaA2
doc_id = doc_18137  - user_id @vijaypks51
doc_id = doc_33137  - user_id @forum_asia
doc_id = doc_46927  - user_id @AhluwaliaA2
doc_id = doc_47296  - user_id @AhluwaliaA2
doc_id = doc_30394  - user_id @kinghayer1986
```

`def inverted_index_tfidf`



```
doc_3215 1.0728
doc_46427 0.8941787999999999
doc_39111 0.8941787999999999
doc_43174 0.7665156
doc_30164 0.73900791
doc_35301 0.7237548899999999
doc_47694 0.56875375
doc_18421 0.5364
doc_37058 0.53081226
doc_26632 0.53081226
```

Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

Evaluation

Functions for evaluation techniques

Before evaluating our queries, we have defined the functions to compute all the evaluation techniques that were asked.

- **Precision@K -** `def precision_at_k(doc_score, y_score, k=10)`
  It computes the precision for the top *k* results, indicating the proportion of relevant documents (label = 1) among the top *k* (k high scores) retrieved documents.

- **Recall@K -** `def recall_at_k(doc_score, y_score, k=10)`
  It measures recall within the top *k* results, dividing the amount of relevant documents (label=1) that are included in the top *k* retrieved documents by the total number of relevant documents.

- **Average Precision@K -** `def avg_precision_at_k(doc_score, y_score, k=10)`
  Computes the average precision across relevant documents in the top *k* results. It tells how much the relevant documents are concentrated in the highest ranked predictions

- **F1-Score@K -** `def avg_precision_at_k(doc_score, y_score, k=10)`
  It computes the F1-score for the top k results by combining precision and recall using the formula F1= $2 \cdot \frac{precision*recall}{precision+recall}$. It indicates overall effectiveness over k.

- **Mean Average Precision -** `def map_at_k(search_res, k=10)`
  It computes the mean average precision over all the queries that we are evaluating, computing the average precision of each queries and then computing the mean.

- **Mean Reciprocal Rank -** `def map_at_k(search_res, k=10)`
  Computes the mean reciprocal rank across all the queries that we are evaluating by averaging the reciprocal ranks of the first relevant document found in each query.

- **Normalized Discounted Cumulative Gain -** `ndcg_at_k(doc_score, y_score, k=10)`
  It computes the normalized DCG at *k* by dividing the DCG by the ideal DCG, giving a score between 0 and 1 to represent ranking quality.

Ground truth file for evaluation

We were given a file *evaluation_gt* with a subset of documents, 2 queries and the ground truth files for each query to perform evaluation with the functions explained above. This file had 3 columns: doc_id, query_id, label.
To be able to perform evaluation, we added the score for each query and document computed by the `inverted_index_tfidf` function

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

Our Ground Truth Dataframe

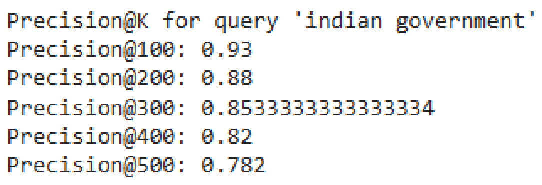
In this section we have created our own ground-truth dataframe. The criteria we used to label de documents is defined by the function `def words_together(query, document, clean_tweets)` this function checks if the whole query appears together in the document, if it is the case then ir labels de document with 1, otherwise if the words of the query do not appear one after the other in the document it labels the document with 0.

Once we have the criteria defined we create the dataframe. We have defined the five queries we used above in a dictionary and then we iterate through them to get the ranked tweets of the queries using the function `inverted_index_tfidf`. We label the documents of the queries using the function we explained above, and then we append the results to the list. Once we have all the information stored in the list we create the data frame.

Evaluation of our Queries

Precision@K

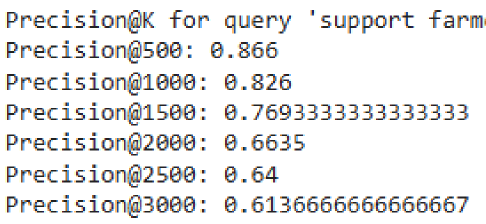
- **Query 1 → "*indian government*":**
  To test the precision of the first query we have defined `k = [100,200,300, 400, 500]`



```
Precision@K for query 'indian government'
Precision@100: 0.93
Precision@200: 0.88
Precision@300: 0.8533333333333334
Precision@400: 0.82
Precision@500: 0.782
```

  As seen in the picture, the precision decreases as the k value increases, it drops from 0.93 to 0.782, this indicates that the top 100 documents (first value of k) are highly relevant, but as we consider more documents (k = 500) less proportion of documents are relevant.

- **Query 2 → "*support farmers*":**
  To test the precision of the second query we defined `k = [500,1000,1500, 2000,2500,3000]`

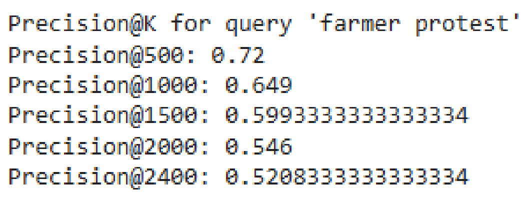


```
Precision@K for query 'support farmers'
Precision@500: 0.866
Precision@1000: 0.826
Precision@1500: 0.7693333333333333
Precision@2000: 0.6635
Precision@2500: 0.64
Precision@3000: 0.6136666666666667
```

  In this second query as in the first one, the precision decreases as k increases what suggests that the most relevant documents are within the first 500 and adding more documents reduces the precision.

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

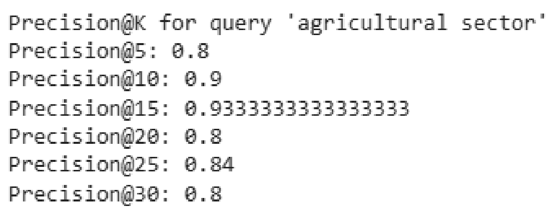
- **Query 3 → "*farmer protest*":**
  To test the precision of the third query we defined `k = [500,1000,1500,2000,2400]`



```
Precision@K for query 'farmer protest'
Precision@500: 0.72
Precision@1000: 0.649
Precision@1500: 0.5993333333333334
Precision@2000: 0.546
Precision@2400: 0.5208333333333334
```

  In this query, precision decreases as k increases what also suggest that relevant documents are concentrated within the first documents.

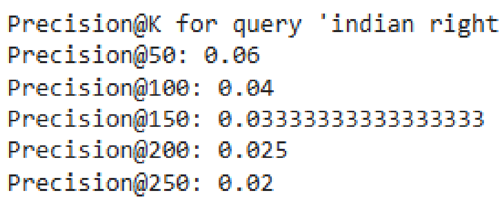
- **Query 4 → "*agricultural sector*":**
  To test the precision of the fourth query we defined `k = [5,10,15,20,25,30]`



```
Precision@K for query 'agricultural sector'
Precision@5: 0.8
Precision@10: 0.9
Precision@15: 0.9333333333333333
Precision@20: 0.8
Precision@25: 0.84
Precision@30: 0.8
```

  In this query precision states relatively high for all k which suggest that the relevant documents are well-ranked in the top positions.

- **Query 5 → "*indian right*":**
  To test the precision of the fifth query we defined `k = [50,100,150,200,250]`

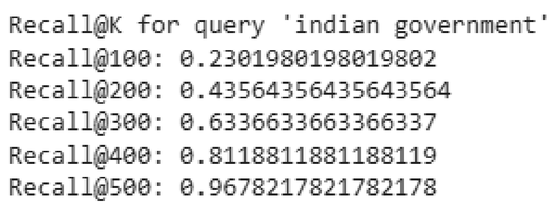


```
Precision@K for query 'indian right'
Precision@50: 0.06
Precision@100: 0.04
Precision@150: 0.033333333333333333
Precision@200: 0.025
Precision@250: 0.02
```

  In this query precision is very low what indicates that there are few relevant documents in the dataset talking about this query.

Recall@K

- **Query 1 → "*indian government*":**
  To test the recall of the first query we have defined `k2 = [100,200,300, 400, 500]`



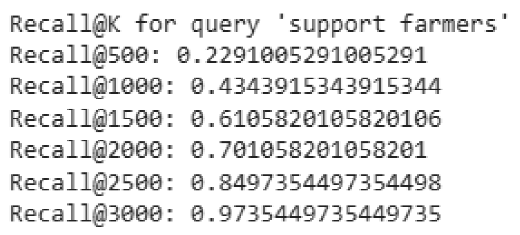
```
Recall@K for query 'indian government'
Recall@100: 0.2301980198019802
Recall@200: 0.43564356435643564
Recall@300: 0.6336633663366337
Recall@400: 0.8118811881188119
Recall@500: 0.9678217821782178
```

upf. Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

In this query recall increases as k rises, this indicates that whenever more documents are considered more relevant documents appear, so we can find a good proportion of relevant documents within the first 500.

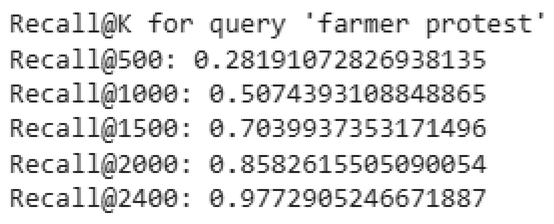
- **Query 2 → "*support farmers*":**
  To test the recall of the second query we defined `k2 = [500,1000,1500, 2000,2500,3000]`



```
Recall@K for query 'support farmers'
Recall@500: 0.2291005291005291
Recall@1000: 0.4343915343915344
Recall@1500: 0.6105820105820106
Recall@2000: 0.701058201058201
Recall@2500: 0.8497354497354498
Recall@3000: 0.9735449735449735
```

In this query happens as in the first one, as we considered a higher number of documents more relevant ones appear. Recall indicates effectiveness, nevertheless precision might suffer as k increases.

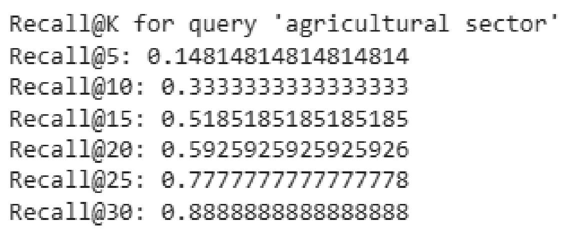
- **Query 3 → "*farmer protest*":**
  To test the recall of the third query we defined `k2 = [500,1000,1500, 2000,2500,3000]`



```
Recall@K for query 'farmer protest'
Recall@500: 0.28191072826938135
Recall@1000: 0.5074393108848865
Recall@1500: 0.7039937353171496
Recall@2000: 0.8582615505090054
Recall@2400: 0.9772905246671887
```

The values of the recall of this query increases as k increases, in the same way as query 1 and 2.

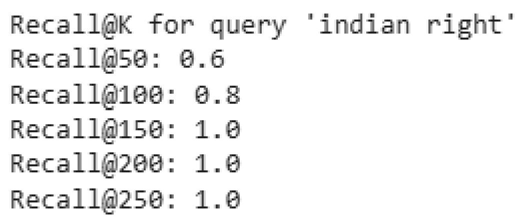
- **Query 4 → "*agricultural sector*":**
  To test the recall of the fourth query we defined `k2 = [5,10,15,20,25,30]`



```
Recall@K for query 'agricultural sector'
Recall@5: 0.14814814814814814
Recall@10: 0.3333333333333333
Recall@15: 0.5185185185185185
Recall@20: 0.5925925925925926
Recall@25: 0.7777777777777778
Recall@30: 0.8888888888888888
```

The recall also increases for higher values of k which in this case, given the values, indicates that few relevant documents are well-ranked, a higher number of results means a significant portion of relevant documents although they may not be concentrated in the very top rank.

- **Query 5 → "*indian right*":**
  To test the recall of the fifth query we defined `k3 = 300`

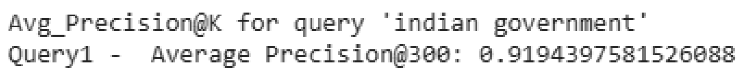


```
Recall@K for query 'indian right'
Recall@50: 0.6
Recall@100: 0.8
Recall@150: 1.0
Recall@200: 1.0
Recall@250: 1.0
```

In this query recall increases until we reach k = 150 where it achieves recall=1, this means that all relevant documents for this query are included within the first 150.

Avg_Precision@K

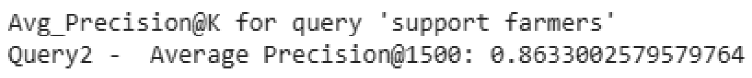
- **Query 1 → "*indian government*":**
  To test the average precision of the first query we have defined `k3 = 300`



```
Avg_Precision@K for query 'indian government'
Query1 -  Average Precision@300: 0.9194397581526088
```

In this case average precision is a high value which indicates that relevant documents are normally distributed across the top 300 documents.

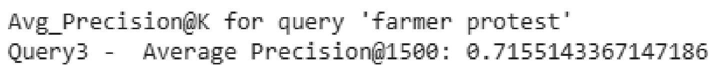
- **Query 2 → "*support farmers*":**
  To test the average precision of the second query we defined `k3 = 1500`



```
Avg_Precision@K for query 'support farmers'
Query2 -  Average Precision@1500: 0.8633002579579764
```

This average precision is also a high value, although lower than for query 1, this also demonstrates a good ranking performance, showing again that relevant documents are relatively well-distributed within the top 1500.

- **Query 3 → "*farmer protest*":**
  To test the average precision of the third query we defined `k3 = 1500`



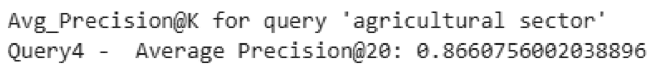
```
Avg_Precision@K for query 'farmer protest'
Query3 -  Average Precision@1500: 0.7155143367147186
```

This value is also lower than from query 1 and 2 what suggest that relevant documents are not as well distributed as the documents of queries 1 and 2.

Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

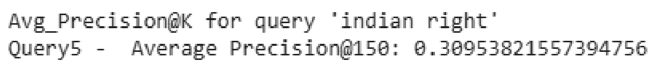
- **Query 4 → “*agricultural sector*”:**
  To test the average precision of the fourth query we defined `k3 = 20`



```
Avg_Precision@K for query 'agricultural sector'
Query4 -  Average Precision@20: 0.8660756002038896
```

  In this case average is also higher so we can assume a performance really similar to the performance of query 2.

- **Query 5 → “*indian right*”:**
  To test the average precision of the fifth query we defined `k3 = 150`

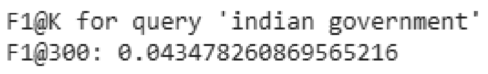


```
Avg_Precision@K for query 'indian right'
Query5 -  Average Precision@150: 0.30953821557394756
```

  This average precision, in contrast, has a lower value which means that relevant documents are less frequent within the first 150 results.

F1-Score@K

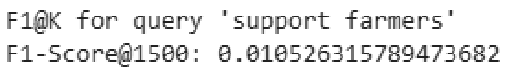
- **Query 1 → “*indian government*”:**
  To test the F1 score of the first query we have defined `k4 = 300`



```
F1@K for query 'indian government'
F1@300: 0.043478260869565216
```

  This low value shows that although precision and recall values are high, they may not be well-balanced.

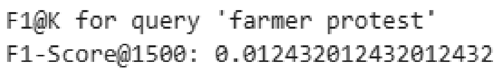
- **Query 2 → “*support farmers*”:**
  To test the F1 score of the second query we defined `k4 = 1500`



```
F1@K for query 'support farmers'
F1-Score@1500: 0.010526315789473682
```

  This value is really low which suggests a poor performance, this low value can indicate that the ranking does not effectively prioritize relevant documents.

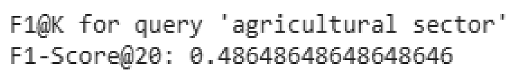
- **Query 3 → “*farmer protest*”:**
  To test the F1 score of the third query we defined `k4 = 1500`



```
F1@K for query 'farmer protest'
F1-Score@1500: 0.012432012432012432
```

This low value suggest a similar situation to the query 2, balanced between precision and recall is quite weak.

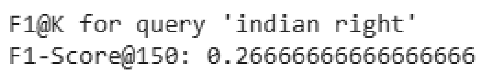
- **Query 4 → "*agricultural sector*":**
  To test the F1 score of the fourth query we defined `k4 = 20`



```
F1@K for query 'agricultural sector'
F1-Score@20: 0.48648648648648646
```

  This value is relatively higher than the other queries, suggesting a better balanced between precision and recall, making it more effective than the other queries.

- **Query 5 → "*indian right*":**
  To test the F1 score of the fifth query we defined `k4 = 150`

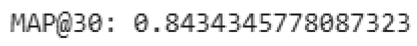


```
F1@K for query 'indian right'
F1-Score@150: 0.26666666666666666
```

  This value, although is still low, is higher than the f1 score for queries 1, 2 and 3, so we can deduce that it has a moderated balance between the precision and the recall.

Mean Average Precision

This value is not calculated for each query but for all queries together so we define `k5 = 30` and we compute the mean:

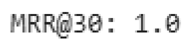


```
MAP@30: 0.8434345778087323
```

This value suggest that on average, the ranking system performs well across all the queries, so we can deduce that we can find the most relevant documents well ranked at the top positions ensuring the effectiveness of the system identifying relevant documents.

Mean Reciprocal Rank

This values is also calculated for all queries together so we define `k6 = 30`



```
MRR@30: 1.0
```

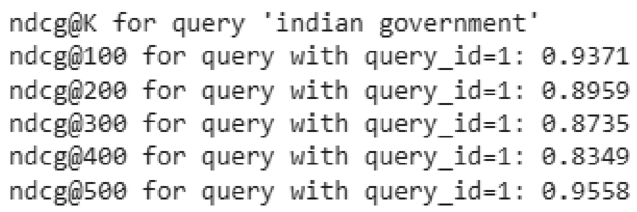
This value indicates that whenever we make a query the system successfully identifies and ranks a relevant document as the first result.

Universitat Pompeu Fabra Barcelona

Elena Barrio 252060
Laura Juan 254094
Rocio Gilabert 251217

Normalized Discounted Cumulative Gain

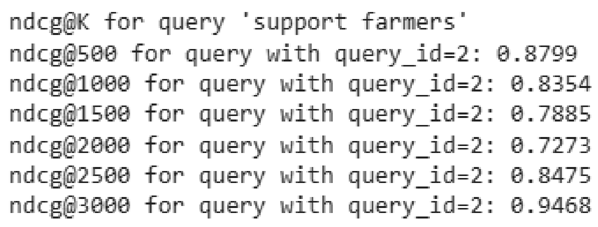
- **Query 1 → "*indian government*":**
  To test the normalized discounted cumulative gain of the first query we have defined
  `k7 = [100,200,300, 400, 500]`



```
ndcg@K for query 'indian government'
ndcg@100 for query with query_id=1: 0.9371
ndcg@200 for query with query_id=1: 0.8959
ndcg@300 for query with query_id=1: 0.8735
ndcg@400 for query with query_id=1: 0.8349
ndcg@500 for query with query_id=1: 0.9558
```

  Values for all k's are relatively close to 1 indicating that the ranking is very effective, since relevant documents appear near the top of the list regardless of the values of K.

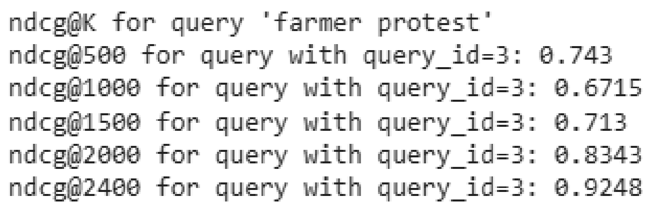
- **Query 2 → "*support farmers*":**
  To test the normalized discounted cumulative gain of the second query we defined
  `k7 = [500,1000,1500, 2000,2500,3000]`



```
ndcg@K for query 'support farmers'
ndcg@500 for query with query_id=2: 0.8799
ndcg@1000 for query with query_id=2: 0.8354
ndcg@1500 for query with query_id=2: 0.7885
ndcg@2000 for query with query_id=2: 0.7273
ndcg@2500 for query with query_id=2: 0.8475
ndcg@3000 for query with query_id=2: 0.9468
```

  Although this values are also high, they decrease a bit more for some values of K (ndcg = 0.72 for k = 2000), this small decline suggests that for that specific number of k, we can find less relevant documents. However since the value increases again as k grows, we can say that the ranking for this query is also effective.

- **Query 3 → "*farmer protest*":**
  To test the normalized discounted cumulative gain of the third query we defined
  `k7 = [500,1000,1500, 2000,2400]`

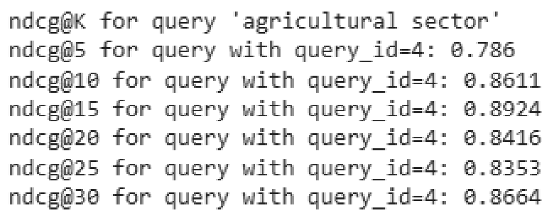


```
ndcg@K for query 'farmer protest'
ndcg@500 for query with query_id=3: 0.743
ndcg@1000 for query with query_id=3: 0.6715
ndcg@1500 for query with query_id=3: 0.713
ndcg@2000 for query with query_id=3: 0.8343
ndcg@2400 for query with query_id=3: 0.9248
```

For query 3 we can appreciate an increasing trend in the ndcg values as k increases, what suggests that relevant documents become better represented in larger sets of documents.

- **Query 4 → "*agricultural sector*":**
  To test the normalized discounted cumulative gain of the fourth query we defined

```
k7 = [5,10,15,20,25,30]
```

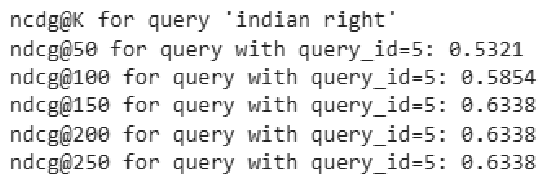


```
ndcg@K for query 'agricultural sector'
ndcg@5 for query with query_id=4: 0.786
ndcg@10 for query with query_id=4: 0.8611
ndcg@15 for query with query_id=4: 0.8924
ndcg@20 for query with query_id=4: 0.8416
ndcg@25 for query with query_id=4: 0.8353
ndcg@30 for query with query_id=4: 0.8664
```

In query 4 values barely have fluctuations for the different values of k so we can say that the query has a good initial ranking and that relevant documents are well-distributed.

- **Query 5 → "*indian right*":**
  To test the normalized discounted cumulative gain of the fifth query we defined

```
k7 = [50,100,150,200,250]
```



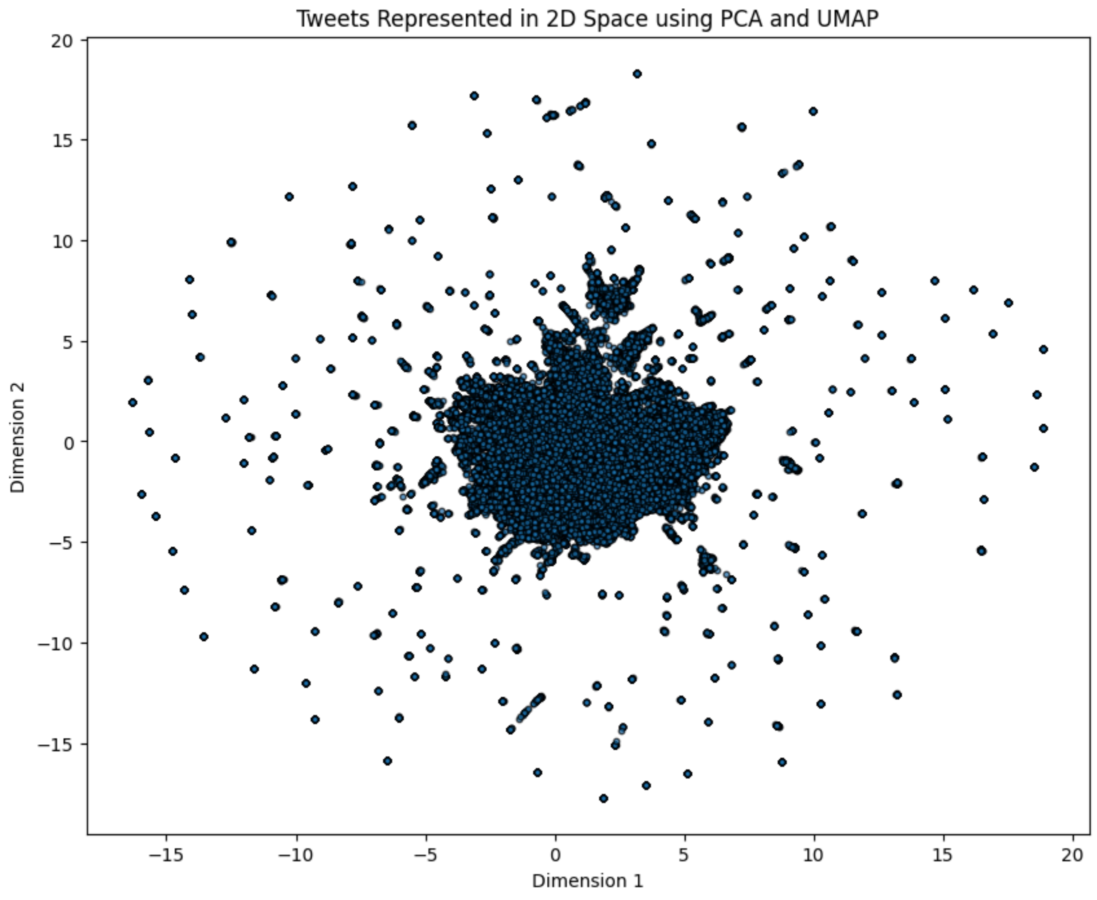
```
ncdg@K for query 'indian right'
ndcg@50 for query with query_id=5: 0.5321
ndcg@100 for query with query_id=5: 0.5854
ndcg@150 for query with query_id=5: 0.6338
ndcg@200 for query with query_id=5: 0.6338
ndcg@250 for query with query_id=5: 0.6338
```

Values for query 5 are relatively lower than for the other queries; this suggests that relevant documents are not really good prioritized in the ranking.

Vector Representation

The analysis aims to visualize the vector representations of tweets using dimensionality reduction techniques. To do so we will be using UMAP instead of T-SNE because of its computational complexity, UMAP has faster computation, efficient memory use, scalability, and better global structure preservation for large datasets.

This analysis uses Word2Vec to convert words into high-dimensional vectors and then averages them to create a unique representation for the tweet. To be able to visualize them, PCA reduces tweets vectors to 50 dimensions. Then we use UMAP to reduce data to 2D, to be able to visualize it. Finally we compute a 2D scatter plot where each point represents a tweet.



This structure shows how the tweets make kind of a cluster around a certain common theme, providing a powerful tool for exploratory data analysis.