Mireia Pou Oliveras 251725
Iria Quintero García 254373
Javier González Otero 243078

Part 1: Text Processing and Exploratory Data Analysis

Introduction

The objective of this project is to develop a search engine.

We are dealing with a document corpus containing a set of Tweets related to the Indian farmers protests in 2021. The dataset contains **117407 JSON objects,** each of which follows the structure of the Tweet shown in the appendix of the statement.

Our first part of this project is structured in the following way:

- Part 0: Imports and Drive.
- Part 1: Text Preprocessing
- Part 2: Exploratory Data Analysis

In Part 0, we import all the necessary libraries and connect to Google Drive. In Part 1, we preprocess the tweets, following the guidelines in the project statement. Finally, in Part 2, we perform an analysis of the data to better understand its characteristics.

1. Data Pre-Processing

Text preprocessing is performed in 2 main parts: Data Load and Text Preprocessing Functions.

1. Data Load

Before preprocessing the data, we need to load it correctly. To load the data, we directly read the zip file and extract each tweet as a json object. Furthermore, we also ensure that considered tweets belong to the *tweet_documents_ids_map* file as stated in the statement with a dictionary that matches every tweet with its docID. That dictionary is built with the *load_docid_map_from_zip* function.

2. Text Preprocessing Functions

This part contains the functions used to preprocess the text. There are two main functions:

build_terms(line)

This function receives a document from the corpus (JSON) and processes the **content field** of the tweet, to return a list of tokens that can be used to make some analysis.

For that purpose, we make the following:

- We first convert the whole content to lowercase since capital letters are not important to conduct the majority of analyses.
- Then, we eliminate every symbol from the text that is not a lowercase letter or a number, (Hashtags and other relevant symbols are not lost as we will see in the next function).
- Finally, we split the line by words, so we obtain a list of tokens.

After tokenization has been done stop words are removed from the tokens list, concretely we eliminate all stop words from the English dictionary and from the *hinglish* one, which also contains Hindi words translated to English, since they are present in some documents. We also added the word *'amp'* to stop words list because we have found that when the symbol '&' is written in a Tweet, the word '&' is added to the document, since it is probably its symbol in the text format used for the JSONs.

Then we also eliminate all single-letter words, since we found them present in some documents. The reason behind this probably is that the json format of Tweets maintains the single letter commands to perform tasks such as newlines \n or tabulars \t.

Finally, to ease the analysis, we perform a stemming of the resulting tokens.

extract_tweet_info_with_docid

This function receives an entire document (Tweet in Json format) and a dictionary that maps Tweets IDs to DocIDs as specified in the file *tweet_documents_ids_map*. The function returns a dictionary containing DocID | TweetID | Tweet Content | Tokenized Tweet | Date | Hashtags | Likes | Retweets | Url from the tweet.

This function stores each returning field in a variable. If TweetID is not found among *tweet_documents_ids_map* file, then the tweet is discarded. To build the Tokenized Tweet, the function calls *build_terms* function.

Motivation for returning each of the fields:

DocID: We find it important to store it since it may be useful for posterior parts of the project, as it is stated in the statement, also to ensure that all tweets match a DocID from *tweet_documents_ids_map* file.

TweetID: To be able to identify uniquely every tweet.

Tweet Content: This field keeps the original text content from the tweet as it is, without performing any processing task. We find it interesting so when performing analyses we can be able to read the original content of the Tweet without modifications.

Tokenized Tweet: This field contains the most relevant information for the data analysis, as its label says, it has all relevant stemmed words from the tweet in a tokenized format. With this information we can perform many analysis tasks as we will see in the next part. It is also necessary to be able to perform TF-IDF and other tasks for developing the search engine.

We have decided to not incorporate hashtags here since it would make it difficult to analyze the text. For example when computing the number of words, the # symbol would make it more difficult to process them. To perform hashtag analyses we have created a proper field only for them, with which we can also increase the weight of a word if it appears on the hashtag list for searching purposes.

Date: This may also be useful to perform some temporal analyses. It is also interesting so one can be able to filter tweets temporarily, improving the efficiency of the search engine.

Hashtags: To be able to search for Hashtags, and to perform several analyses.

Likes: Mainly to see most liked tweets and to act as a filter.

Retweets: Same as Likes.

URL: Once a tweet has been found, we can search it by its URL.

After applying these two functions to all the elements from the corpus, a total of **48429 elements** were retrieved, coinciding with the number of documents registered in *tweet_documents_ids_map* file. Below there is an example of a processed tweet:

```
Unset
{'DocID': 'doc_0',
'TweetID': 1364506249291784198,
'Original Tweet': 'The world progresses while the Indian police and Govt are
still trying to take India back to the horrific past through its tyranny.
\n\n@narendramodi @DelhiPolice Shame on you. \n\n#ModiDontSellFarmers
\n#FarmersProtest \n#FreeNodeepKaur https://t.co/es3kn0IQAF',
'Tokenized Tweet': ['world', 'progress', 'indian', 'polic', 'govt', 'india',
'back', 'horrif', 'past', 'tyranni', 'narendramodi', 'delhipolic', 'shame',
'modidontsellfarm', 'farmersprotest', 'freenodeepkaur',
'httpstcoes3kn0iqaf'],
'Date': '2021-02-24T09:23:35+00:00',
'Hashtags': ['#ModiDontSellFarmers', '#FarmersProtest', '#FreeNodeepKaur'],
'Likes': 0,
'Retweets': 0,
'Url': 'https://twitter.com/ArjunSinghPanam/status/1364506249291784198'}
```

2. Exploratory Data Analysis

The “Exploratory Data Analysis” is performed in 5 different steps: Basic Statistics, Word Cloud, Retweeted Tweets, Entity Recognition and Other interesting statistics.

1. Basic statistics

In this part, we compute the basic statistics about the dataset.

First, we identify the **top N words**, using a function called print_top_n_words, where we basically find the most frequent words in the dataset and print them. To try the function we use n=10, and find that the most common word is `farmersprotest (50106)`.

Secondly, we display the **word distribution**, with a function called word_count_distribution. This function reuses the previous function to get the top n words and then performs a plot to show the distribution.

Finally we also show the **average sentence length**, with a function called average_sentence_length, which calculates the average number of words per sentence by summing all sentences lengths and dividing by the number of sentences.

2. Word cloud

In this second part, we generate a **word cloud** for the most frequent words, where the size of each word in the cloud reflects its frequency. For the one displayed, we have used the 50 most frequent words.

3. Retweeted tweets

In the third part, we find the **tweets with the highest number of retweets**, with a function called top_retweeted_tweets. This function sorts the tweets by the number of retweets in descending order, and prints the top n.

4. Entity Recognition

In the fourth part, we perform **entity recognition**, with a function called entity_recognition. In essence, this task involves detecting and categorizing named entities (such as people, organizations, etc.). We believe it's important to perform it for two reasons:

- It helps understand the type of information in the dataset.
- It may help in future tasks, such as searching for specific labels (search for organizations, locations, ...).

To perform entity recognition, we use a pre-trained NLP model and apply it to the original tweet to extract the entities.

5. Other interesting statistics

In this last part, we show other statistics that we find interesting.

a) Hashtag analysis

We start by showing **the most common hashtags** and most **common hashtags cooccurrences**. We believe this is really interesting because we can identify the most "trending topics" and analyze the connections between the different topics.

To show the most common hashtags, we have created a function called most_common_hashtags, that counts all the hashtags and gets the n most common. To show the most common hashtags

cooccurrences, we created another function called hashtag_cooccurrence, which counts the occurrences by pairs of hashtags, and returns de top n.

Finally, we also made a network of the hashtags cooccurrences to show them visually.

b) Most retweeted users

After performing this hashtag analysis, we continue by finding the **most retweeted users**, instead of the most retweeted tweets as before. We believe that knowing which users have been retweeted the most is important for identifying the sources of the content, as these users are also the most influential people.

To make this analysis, we created a function called most_retweeted_users, which counts the retweets per user and gets the topN retweeted users. Then, it also makes a plot comparing the number of followers of the user vs the number of retweets the tweet has. This plot is made to see if there is any correlation between the two variables (results show that there is!).

Before using this function, it's important to remark that we needed to add some fields to the processed tweet. In particular, we added the username and the number of followers. This is done in a function called extract_extended_tweet_info_with_docid.

c) Evolution of tweets over time.

As a final analysis, we decided to show a plot with the evolution of **tweets over time**, that is, the number of tweets by date. To do it, we created the function plot_tweets_over_time, that simply counts the number of tweets for each date.

The resulting plot shows what we suspected: At the beginning there are lots of tweets, but as the days pass the quantity decreases.