Ingrid Sancho Aventin - 218033
Vladimir Trukhaev - 219328
Information Retrieval & Web Analysis

Universitat Pompeu Fabra Barcelona

Final Project Part 1: Text Processing

Introduction

This part of the final project consisted in processing a given text file corpora from the World Health Organization Twitter account (*dataset_tweets_WHO.txt*). This had to be done in order to retrieve the tweets tweeted by @WHO.
We had to process the given text file and extract the tweet text and pre-process it. This consisted in removing any stop-words the tweet text may have, tokenizing it, removing any punctuation that appeared in it, stemming it, lowering the capital letters and so on.
As a submission we have this PDF file (*IRWA-2021-final-project-U162442-U161809-part-1*) and the following GitHub repository where you can find the code we have used to accomplish all the tasks required for the submission: https://github.com/vladimirtrukhaev/IRWA_Project. In said GitHub repository you can find a branch called *"IRWA-2021-final-project-part-1"* (https://github.com/vladimirtrukhaev/IRWA_Project/tree/IRWA-2021-final-project-part-1) where the code for this part of the project, this PDF file as well as the provided text file are located.

Code Explanation

Libraries and Reading Files

Firstly we had to load the needed libraries (we mostly used `nltk`), update/download the Stop Words list (`nltk.download('stopwords')`) and proceed by loading the *dataset_tweets_WHO.txt* text file.

Tweet Dictionary

After loading all the necessary files and libraries we created a dictionary that has as keys the Tweet IDs and as values the Text of said Tweets. To do so we take the keys from the original *dataset_tweets_WHO.txt* text file and append them into a list called `keylist`. Then we create a dictionary called `my_dict` where we set the keys with the previously created list. We then iterate through the original dataset once again in order to get the full Tweet Text and add it as a value to our dictionary. This process can be seen in *Figure 1* below.

```
#reading data
doc = "dataset_tweets_WHO.txt"
with open(doc, 'r') as file:
    data = json.load(file)

#initializing dictionary "my_dict" where value is the tweet text and key its id
keylist = []
for key in data:
    keylist.append(key)


my_dict = {}

for i in keylist:
    my_dict[i] = None

for key in data:
    tweet = []
    for i in data[key]["full_text"]:
        tweet.append(i)
    tweet1 = "".join(tweet)
    my_dict[key] = tweet1
```

Fig. 1. Creating a dictionary with the Tweet IDs as Keys and Tweet Texts as Values.

Lowering

We then proceed by lowering all the cases in the tweet texts. This is one of the most important steps in Natural Language Processing (NLP). Information and information obviously mean the same however, when working in the Vector Space Model these two words will be differently represented due to the upper case I.

In this case this is a pretty easy and straightforward process that can be done iterating through each key in the dictionary using `d[key] = d[key].lower()` (*Figure 2*).

```
def lowering(d):
    """
    Transforming tweet text (values in dictionary) in lowercase

    Argument:
    d -- dictionary where tweets are stored as values

    Returns:
    d -- dictionary with tweets transformed to lowercase as values
    """
    for key in d:
        d[key] = d[key].lower()
    return d
```

Fig. 2. Lowering all the cases for all the Tweet Texts.

Cleaning

Next we need to remove all non alphanumeric characters so that there are no weird characters, punctuation and/or accents in the output text.

We decided to keep the hashtag character (#) since it is used to denote the topic one is tweeting about and users often use them to search topics. We consider the hashtag information especially useful when talking about implementing a search engine.

In our case this is done using the famous *regex* library and coding it so it only leaves the alphanumeric characters as well as the hashtags in our output (`[^A-Za-z0-9 #]`). Again, we iterate through our dictionary keys and for each key value we run the following line: `["".join(re.sub(r'[^A-Za-z0-9 #]', ' ', i) for i in d[key])]` which can be seen in *Figure 3*.

```
def cleaning(d):
    """
    Removing anything that is not alphanumeric

    Argument:
    d -- dictionary where tweets are stored as values

    Returns:
    d -- dictionary with tweets without any non alphanumeric character
    """
    for key in d:
        d[key] = ["".join(re.sub(r'[^A-Za-z0-9 #]', ' ', i) for i in d[key])]
    return d
```

Fig. 3. Cleaning all Tweet Texts from anything that is not alphanumeric or a hashtag.

Tokenizing

We now continue by tokenizing the Tweet Texts we have as our values in our dictionary. Tokenizing basically means splitting each sentence into words that can also be called tokens when speaking about NLP. This process also removes all punctuation which we already did in the previous step when we cleaned our data.
This tokenization process helps us interpret the meaning of the whole text by just looking at the sequence of the tokens which give us more information about the context. At first we used the `nltk.tokenize` package however, after seeing it splits not only words but hashtags too we decided to use the Python function `.split()` as seen in *Figure 4*.

```
def tokenize(d):
    """
    Tokenizing the tweets, in other words, splitting text by "words"

    Argument:
    d -- dictionary where tweets are stored as values

    Returns:
    d -- dictionary with lists of words as values
    """
    for key in d:
        for sentence in d[key]:
            d[key] = sentence.split()
    return d
```

Fig. 4. Tokenizing all the Tweet Texts using .split().

Removing Stopwords

Now that we have the tweets splitted by words we need to erase those words that do not have meaning, which are called “stopwords”. We recognized, by looking at the input data, at least three languages (English, Spanish and French). We know this is not an ideal situation, we would rather have them all in one language, by translating or removing the minority language tweets. Nevertheless we decided to do the stopwords erasing process three times, one for each language. Even though this might be disadvantageous (considering one language can have a meaningful word deleted because it coincides with another language’s stopword) we considered that, taking into account the amount of data the negative impact would be negligible.
In order to implement this, we used `stopwords.word(`*`language`*`)` where language is the desired language to get the stopwords from and ran through a predefined list of languages (Figure 5).

```
def stpwords(d):
    """
    Removing stopwords, which are very common words that do not contain meaning

    Argument:
    d -- dictionary where list of words of tweets are stored as values

    Returns:
    d -- dictionary with lists of words as values, now with no stopwords
    """
    languages = ["english", "spanish", "french"]
    for language in languages:
        stop_words = set(stopwords.words(language))
        for key in my_dict:
            my_dict[key] = [word for word in my_dict[key] if word not in stop_words]
    return d
```

Fig. 5. Erasing Spanish, English and French stopwords from our data.

Stemming

As the last step we have to stem each token, which means to keep the root (core meaning) of each word. In order to implement this, we use the imported algorithm `PorterStemmer` from `nltk.stem` package. We tried to implement this process for each of the languages, but after being unsuccessful, we decided to consider the most used language in these tweets (English). We considered the negative impact of not having the core words of a small sample subset (tweets in other languages) negligible. The code for this part can be seen in *Figure 6* below.

```
def stemming(d):
    """
    Stemming tweets, which means to keep only the "root" of each word

    Argument:
    d -- dictionary where list of words of tweets are stored as values

    Returns:
    d -- dictionary with lists of words as values, now stemmed words
    """
    stemmer = PorterStemmer()
    for key in my_dict.keys():
        my_dict[key] = [stemmer.stem(word) for word in my_dict[key]]
    return d
```

Fig. 6. Stemming our Tweet Texts.

Conclusion

This Text Processing process transforms unstructured data into structured one. In our case we started with a large amount of data from Twitter and we transformed it into lowercase, stemmed it, left only alphanumeric data (allowing only hashtags besides letters and numbers), splitted it into words which are stored as lists in a dictionary with identifiers as keys and so on (*Figure 7*). With structured data it will be much easier to build an effective search engine.

```
In [10]: print(my_dict)
{'0': ['intern', 'day', 'disast', 'risk', 'reduct', '#openwho', 'launch', 'multi', 'tier', 'core', 'curriculum', 'help',
'equip', 'compet', 'need', 'work', 'within', 'public', 'health', 'emerg', 'respons', 'start', 'learn', 'today', 'amp', '#
ready4respons', 'http', 'co', 'hbffof0xkl', 'http', 'co', 'fgzy22rwu'], '1': ['#covid19', 'shown', 'health', 'emerg', 'di
sast', 'affect', 'entir', 'commun', 'especi', 'weak', 'health', 'system', 'vulner', 'popul', 'like', 'migrant', 'indigen
', 'peopl', 'live', 'fragil', 'humanitarian', 'condit', 'http', 'co', 'jpuqpnu0v1'], '2': ['intern', 'day', 'disast', 'ri
sk', 'reduct', 'better', 'respond', 'emerg', 'countri', 'must', 'invest', 'health', 'care', 'system', 'achiev', 'gender',
'equiti', 'protect', 'marginalis', 'group', 'ensur', 'readi', 'amp', 'equit', 'access', 'suppli', 'strong', 'amp', 'resil
i', 'health', 'system', 'http', 'co', '5nalyjiymp'], '3': ['rt', 'whoafro', 'congratul', 'algeria', '#algeria', '16th', '
countri', '#africa', 'reach', 'mileston', 'fulli', 'vaccin', '10', 'pop'], '4': ['rt', 'opsom', 'si', 'completament', 'va
cunado', 'pued', 'contraer', 'covid', '19', 'importa', 'si', 'vacunado', 'si', 'todav', 'esperando'], '5': ['rt', 'whosea
ro', '#inform', 'come', 'sourc', 'verifi', 'share', '#checkbeforeyoushar', 'http', 'co', 'vr'], '6': ['rt', 'drtedro', 'm
ust', 'appreci', 'role', 'privat', 'sector', 'play', '#covid19', 'respons', 'develop', 'vaccin', 'short'], '7': ['rt', 'd
rtedro', 'human', 'fail', 'miser', 'vaccin', 'injustic', 'debat', 'whether', 'intellectu', 'properti', 'right', 'waiv'],
'8': ['rt', 'drtedro', '#covid19', 'major', 'impact', 'peopl', '#mentalhealth', 'thank', 'doctorja', 'whophilippin', 'ded
ic', 'work'], '9': ['rt', 'drtedro', 'broad', 'administr', 'booster', 'dose', 'unfair', 'unjust', 'amp', 'immor', 'time',
'#healthwork', 'amp', 'risk', 'peopl', 'mani'], '10': ['rt', 'drtedro', '1951', 'henrietta', 'lack', 'die', '#cervicalcan
c', 'age', '31', '8', 'month', 'diagnosi', 'although', 'life'], '11': ['rt', 'drtedro', 'donat', 'enough', 'deliv', '#vac
cinequ', 'end', '#covid19', 'pandem', 'need', 'stronger', 'leadership', 'ramp'], '12': ['rt', 'whoafghanistan', 'nine', '
flight', 'arriv', '#afghanistan', 'total', '1', '8', '6', 'metric', 'tonn', 'health', 'suppli', 'sinc', 'august', '30'],
```

Fig. 7. Output of running the data through all of the aforementioned processes.