

Summarizing Microblogs during Emergency Events: A Comparison of Extractive Summarization Algorithms

Problem Statement:

Summarizing Microblogs during Emergency Events: A Comparison of Extractive Summarization Algorithms

Abstract:

Microblogging sites, notably Twitter, have become important sources of real-time situational information during emergency events. Since hundreds to thousands of microblogs (tweets) are generally posted on Twitter during an emergency event, manually going through every tweet is not feasible. In such scenario, it is critical to summarize the microblogs (tweets) and present informative summaries to the people who are attempting to respond to the disaster. Hence, summarization of microblogs posted during emergency events has become an important problem in recent years. Several summarization algorithms have been proposed in literature, both for general document summarization, as well as specifically for summarization of microblogs. However, to our knowledge, there has not been any systematic analysis on which algorithms are more suitable for summarization of microblogs posted during disasters. In this work, we evaluate and compare the performance of three extractive summarization algorithms in the application of summarizing microblogs posted during emergency events.

Apart from comparing the performances of the algorithms, we also find significant differences among the summaries produced by different algorithms over the same input data.

Introduction:

Summarization of microblogs posted during emergency situations is an important and practical problem. People at the scene of a disaster post information about the disaster on microblogs in real time. Emergency responses during natural or man-made disasters use information available on social media platforms such as Twitter. Volunteers and other support personnel generate reports or summaries of the relevant tweets posted via Twitter that the responders can then use to address the needs of people located in disaster areas. Such manual intervention may not scale given the volume of data produced within a short time interval during a disaster. Moreover, different stakeholders and responders need information at varying levels of granularities. Some stakeholders may want to obtain overall situational updates for a given day as a short summary or report (high-level information need) or specific updates for a particular class, such as ‘infrastructure damage’, ‘shelter needs’ etc. (class-specific information need).

Related Work:

In this work, we evaluate and compare the performance of three extractive summarization algorithms in the application of summarizing microblogs posted during emergency events.

Apart from comparing the performances of the algorithms, we also find significant differences among the summaries produced by different algorithms over the same input data –using the rouge score.

And also, apart from Summarizing tweets in real-time and extracting useful information from the summarized data, we have extended our project to be more abled-friendly by adding Speech recognition for the summarized data.

Future-Course Work:

Recently, Cyclone Fani that made a landfall in Puri last week had left a trail of destruction in many parts of Odisha, but its management by the government has emerged as a global example of how natural disasters can be handled. In the beginning it was just the storm but has taken the form of depression, it had become a cyclone. Meteorologists accurately predicted its path and announcement has been made days before that it would head straight up the Bay of Bengal and make landfall in Odisha. Therefore, we intend to extend our project to NDRF(Natural Disaster Relief team)- for implementation and upgradation of the system for the same.

Summarization of microblogs:

We outline the extractive summarization algorithms that we considered for comparison in the present work. Note that some of these algorithms were originally proposed for summarization of a single document, where the sentences of the given document are ranked according to some importance measure, and then few important sentences are selected for the summary. These algorithms can be easily applied to summarization of a set of microblogs, where each microblog is analogous to a sentence.

We have used:

1.LUHN

2.LEXRANK and

3.LSA

Summarization Algorithms:

1) LUHN:

# Luhn’s algorithm works on the perception that some words in a document are descriptive of its content, and the sentences that express the most significant information in the document are the ones that contain many such descriptive words close to each other. The words that occur often in a document are likely to be associated with the main topic of the document. However, an exception to this observation is stop-words. Hence, Luhn proposed the idea of stop-words such as determiners, prepositions and pronouns, which do not have much value in informing about the topic of the document. So he suggested removing these words from consideration.?

# Luhn identified descriptive words using empirically determined high and low frequency thresholds. The high thresholds Filtering out considering the words that occur very frequently throughout the article. Similarly the low thresholds filtering out considering the words that occur too infrequently. The remaining words in the document are the descriptive words, which indicate that content which is important.

# On a sentence level, a ‘significance factor’ is computed for each sentence, which could be calculated for a sentence by bracketing the significant words in the sentence, squaring the number of significant words and then dividing by the total number of words. Sentences are identified as important and included in the summary based on the significance factor values.

2) LEXRANK:

# Lex-rank is stochastic graph-based method for computing relative importance of textual units in a document. In this method, a graph is generated which is composed of all sentences in the input corpus. Each sentence is represented as a node, and the edges denote similarity relationships between sentences in the corpus. An intra-sentence cosine similarity measure is used as edge weight in the graph representation of sentences by considering every sentence as bag-of-words model. A connectivity matrix or similarity matrix is generated using the similarity measure, which can be used as a similarity graph between sentences. A thresholding mechanism is applied (i.e.edges having weights below the threshold are removed) to extract the most important sentences from the resulting similarity matrix. ?

3)LSA:

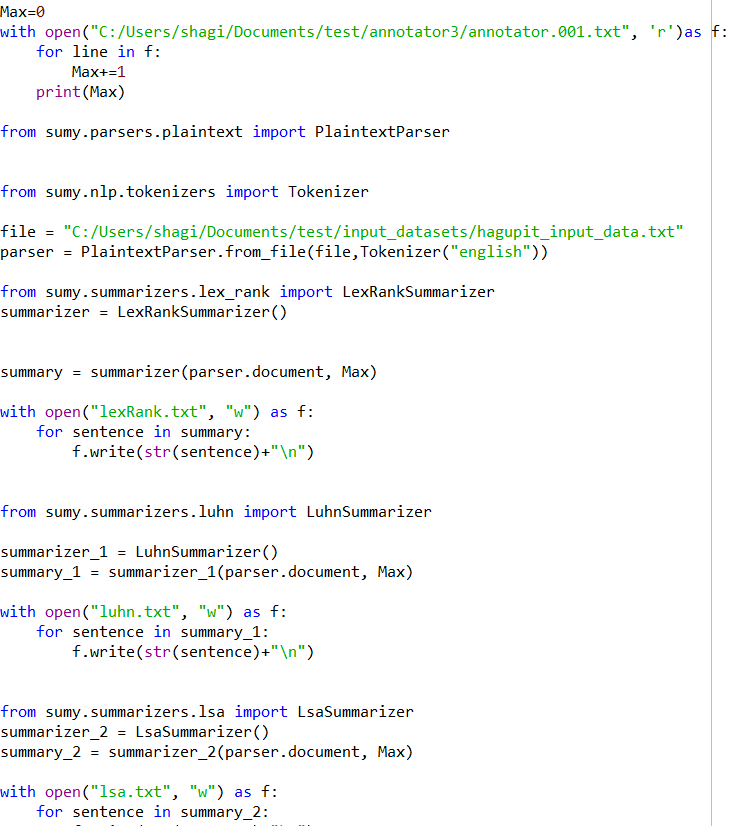
# LSA(latent semantic analysis) is a generic extractive text summarization method to identify semantically important sentences for generating the summary. It is an unsupervised method of deriving vector space semantic representations from large documents, and does not need any training or external knowledge. Considering context of the input document, LSA extracts information such as which words are used together and which common words are seen in different sentences. High number of common words among sentences means that the sentences are more semantically related. LSA is based on mathematical technique which is named Singular Value Decomposition (SVD) that is used to find out the inter-relations between sentences and words. The input text document is first converted into a matrix where each row represents a word and each column represents a sentence. Each cell value represents the importance of the word. SVD is then applied on this matrix to select the sentences to generate the summary.

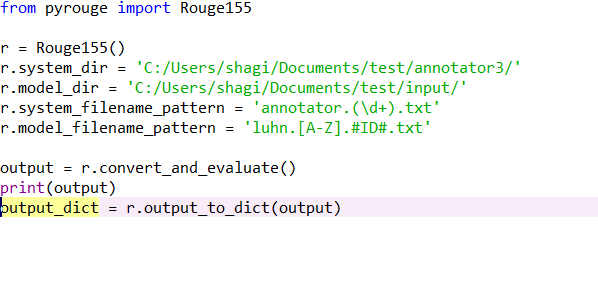
* We have developed our on datasets with the Tweepy api.
* We have used Speech Recognition for abled-friendly.
* We have extracted tweets and summarized on real-time tweets, and have tested on several other datasets.

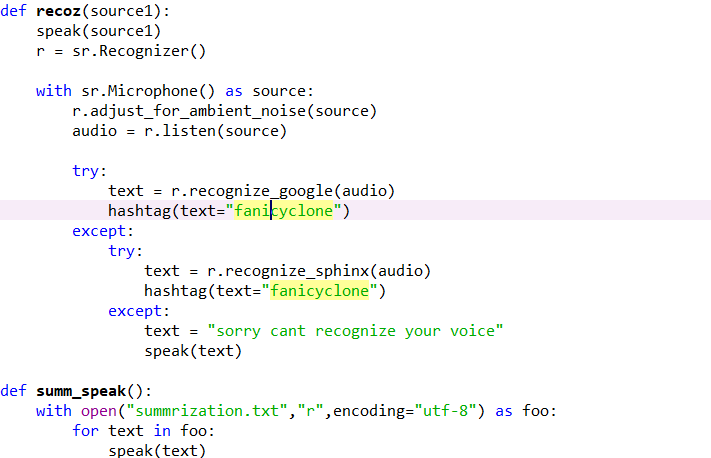
Our work on the recent FANI CYCLONE:

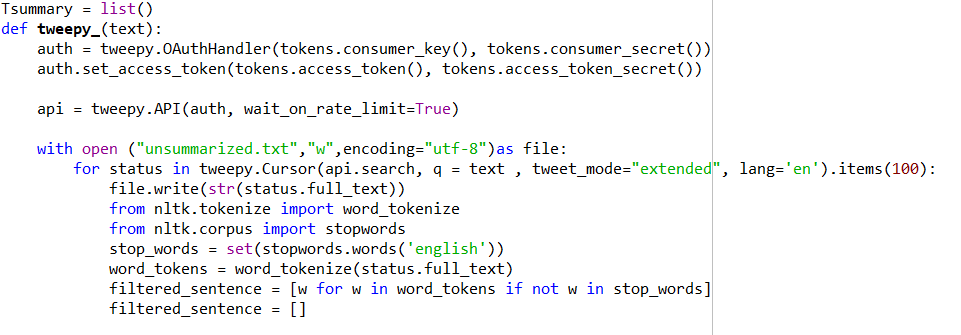
OUR WORK:

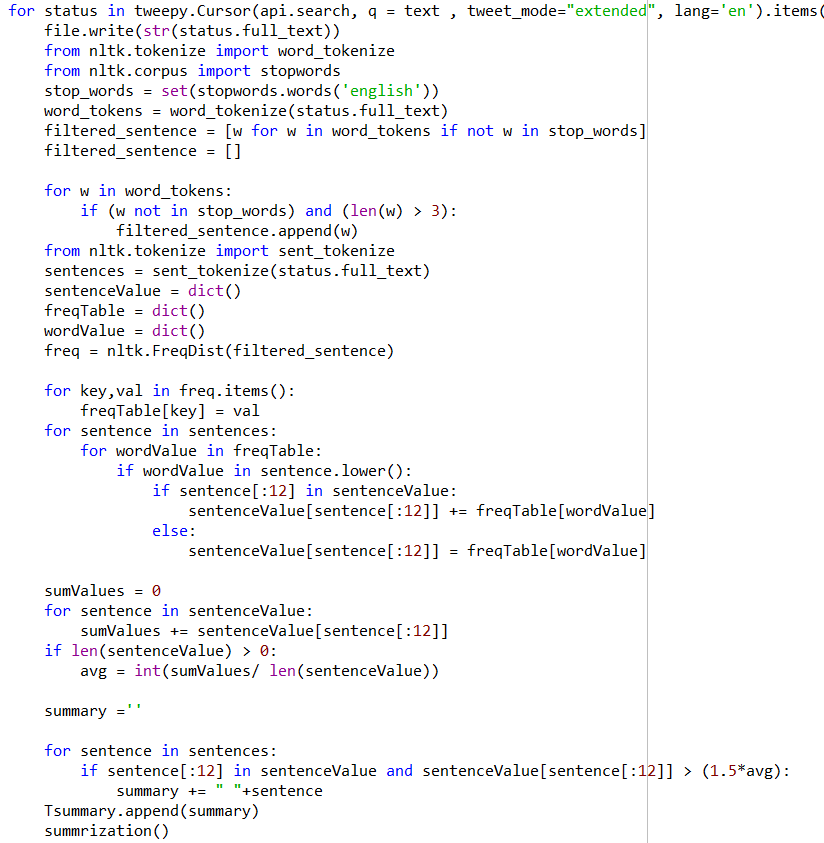
Code:

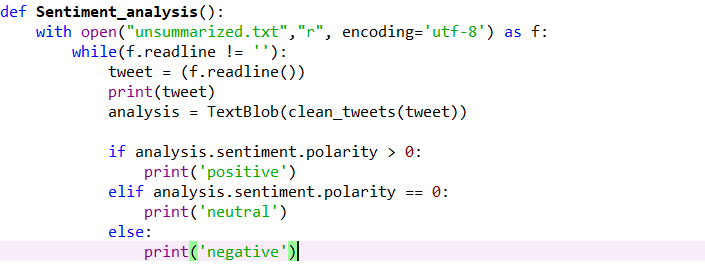






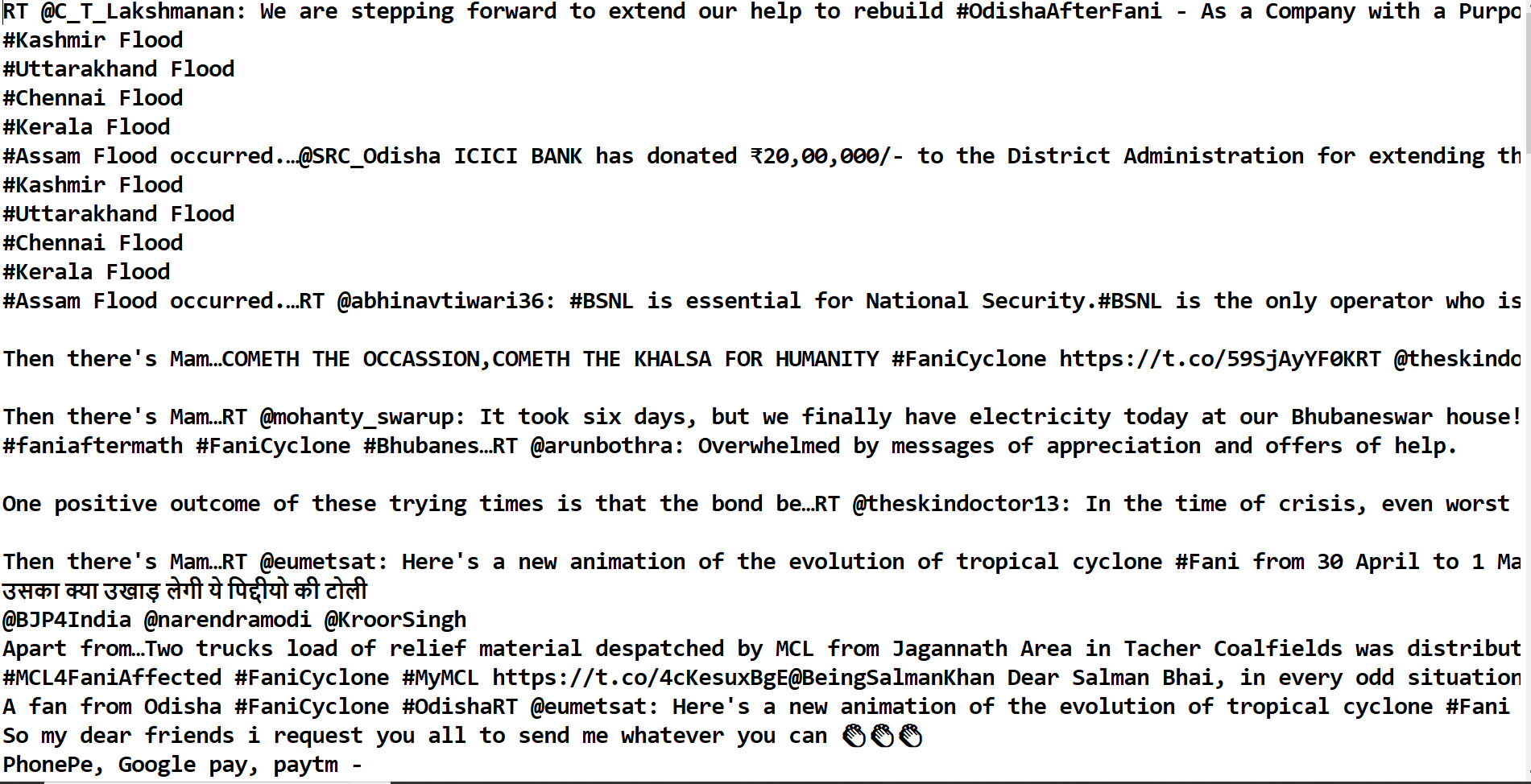


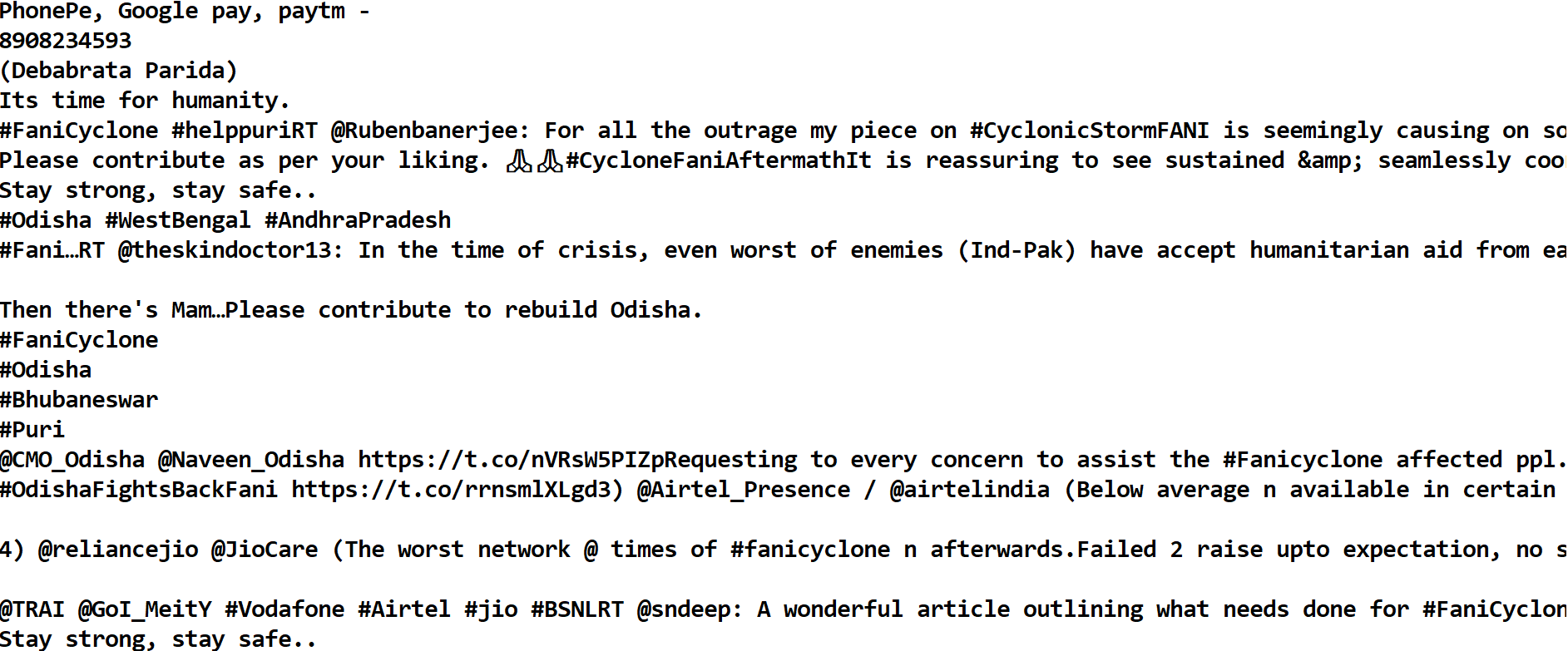




------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Generating Unsummarized Tweets (DATASETS):

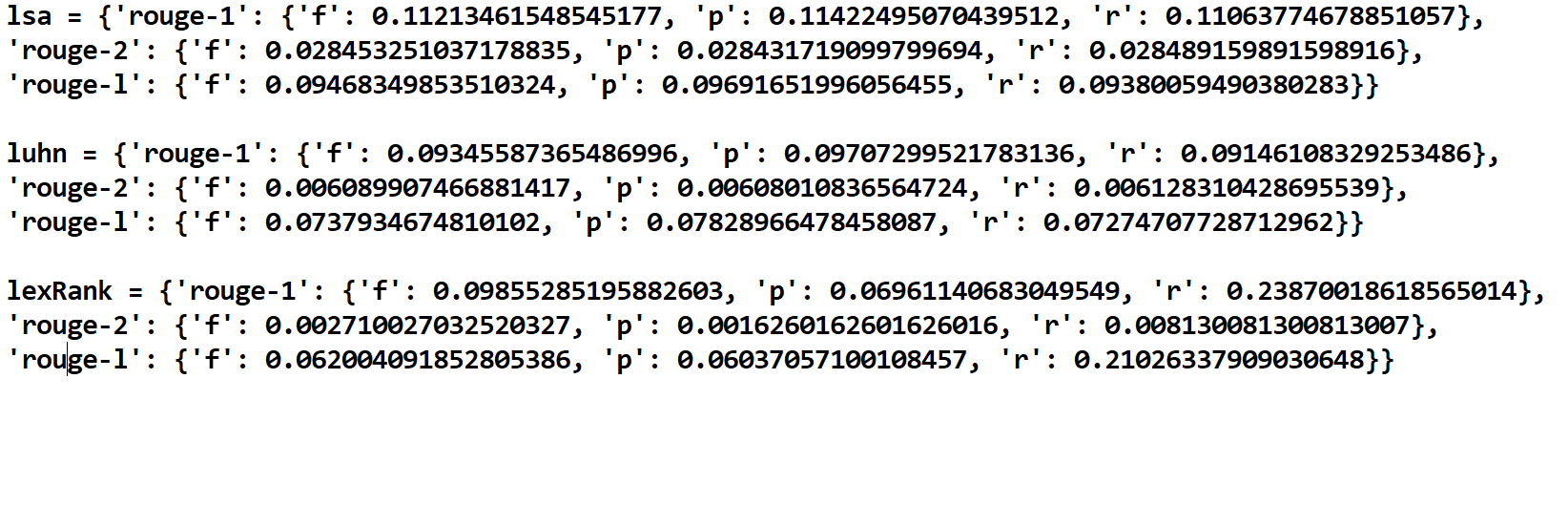




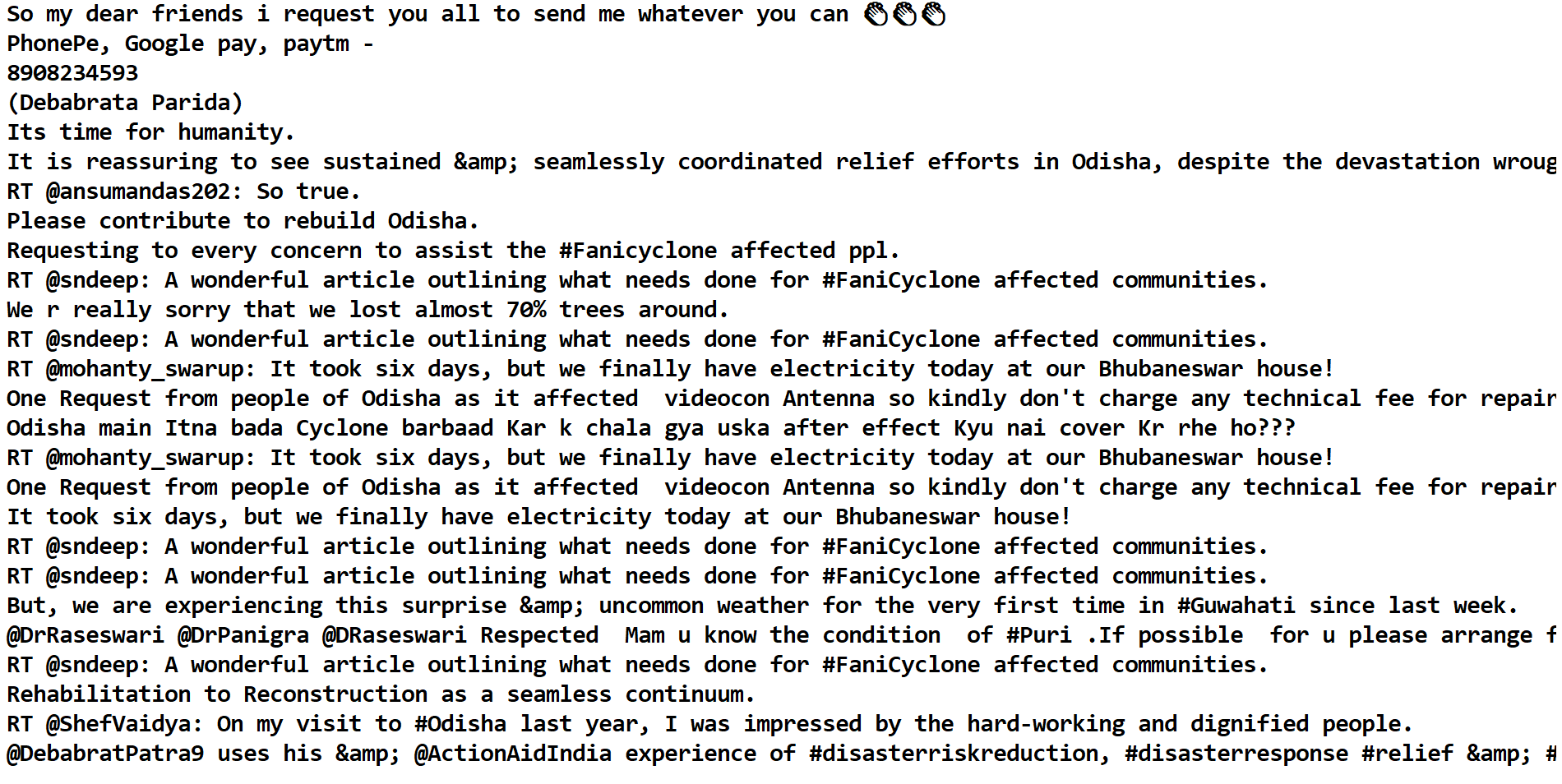
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Output:

Rouge Scores:



Summarized Tweets:



--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

CONCLUSION:

Summarization of microblogs posted during emergency situations is an important and practical problem. Hence, summarization of microblogs posted during emergency events plays a crucial role for Emergency responses during natural or man-made disasters use information available on social media platforms such as Twitter.

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TEXT SUMMARIZATION (using crisisNLP and voice Recognizer and other Summarization Algorithm)

OUR WORK:

STEPS:

A. SPEECH RECOGNITION:

We have included speech recognition with our summarization for abled-friendly and more - quick & interactive information extraction.

Here, we have tried our speech elements with gtts, pyglet, google-search (web- browser) and using pyttsx .

B. SENTIMENTAL ANALYSIS:

We made sentimental analysis as part of our work- for to be extra sure of distress calls from the tweets generated.

So, as to respond quickly to those affected areas based on polarity of the summarized data.

C.TWEEPY AND DATASETS:

We used tweepy for on real-time Tweets generation using twitter api.

Thus, we have developed our on datasets with the tweepy api.

Although, our project can work on analysis of previous or other related datasets.

D. NLTK

We made use of nltk and other python libraries for better tweets cleaning, lemmatization, removing stop words, tokenization and other works, etc. to make our summary free of unwanted or irrelevant information

This ensures that our data is precise on -point and can be evaluated on the go, thus avoiding any delays on our part.

E. TESTING

We evaluated ourselves on different datasets and measured our efficiency.

Also, WE have done our prototyping on the FANI CYCLONE, to get better results.

--------------------------------------------------------------------------------------------------

. SPEECH RECOGNITION-

First, we begin with speech recognition- that we have added as a feature.

--------------------------------------Python libraries to be installed:---------------------------------------------------------------

- speech\_recognition

-webbrowser

- I have mentioned others as comment.

-------------------------------------------------------------------------------------------------------------------------------------------------

First- introduction program to speech recognition.

--------------------------------------------------------------------------------------------------------------------------------------------------

#Just to recognize our voice.

import speech\_recognition as sr

r=sr.Recognizer()

with sr.Microphone() as source:

print('say something!')

audio=r.listen(source)

print('Listening...')

try:

print('\ngoogle thinks you said:\n'+ r.recognize\_google(audio))

except:

pass

print("\nCouldn't recognize your voice")

program for google-speech search

------------------------------------------------------------------------------------------------------------------------------------------------------

#pip install gtts(google-text-to-speech)

#pip install pyglet(playing audio files within system)

import speech\_recognition as sr

import webbrowser as wb

#import speak

# %s token for better string concatenation

chrome\_path="C:/Program Files (x86)/Google/Chrome/Application/chrome.exe %s"

r=sr.Recognizer()

with sr.Microphone() as source:

print('say something!')

audio=r.listen(source)

print('Listening...')

try:

text= r.recognize\_google(audio)

print('\ngoogle thinks you said:\n'+ text)

#lang='en'

# speak.tts(text, lang)

#query text

q\_text= '<https://www.google.com/search?q=>'+ text

#for opening the browser

wb.get(chrome\_path).open(q\_text)

#except:

except Exception as e:

print(e)

print("\nCouldn't recognize your voice")

-----------------------------------------------------------------------------------------------------------------------------------------------

The above mentioned programs - have been done using gtts and pyglet.

SPEECH RECGONITION USING PYTTSX-

Speech recognition with pyttsx.

----------------------------------------Python libraries to be installed:---------------------------------------------------------------

- pyttsx3 (Text-to-speech x-platform )

--------------------------------------------------------------------------------------------------------------------------------------------------

First- introduction program to speech recognition using pyttsx3.

--------------------------------------------------------------------------------------------------------------------------------------------------

'''

#an application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3.Engine instance.

#during construction, the engine initializes a pyttsx3.driver.

#DriverProxy object responsible for loading a speech engine driver implementation from the pyttsx3.drivers module

""

import pyttsx3

engine = pyttsx3.init()

# Speed percent (can go over 100)

engine.setProperty('rate', 150)

#volume(0-1)

engine.setProperty('volume', 0.9)

engine.say("A B C D E F ")

engine.runAndWait()

For change voice and language

#This will give the available options you can try for voices on your computer.

#I have attached an image for reference- related to my system.

------------------------------------------------------------------------------------------------------------------------------------------------------

import pyttsx3

engine = pyttsx3.init()

#voices = engine.getProperty('voices')

#for voice in voices:

print("Voice:")

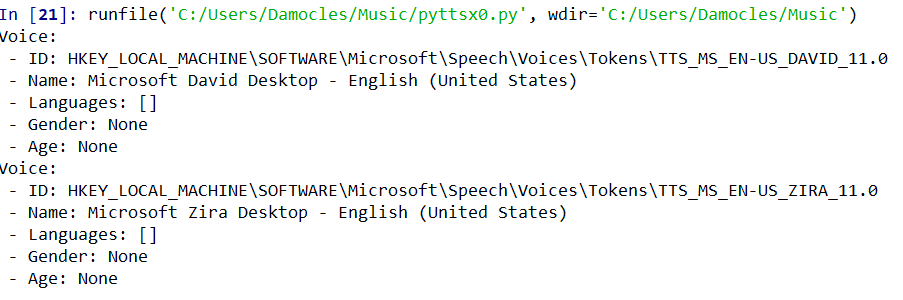
print(" - ID: %s" % voice.id)

print(" - Name: %s" % voice.name)

print(" - Languages: %s" % voice.languages)

print(" - Gender: %s" % voice.gender)

print(" - Age: %s" % voice.age)



Using different voices..

----------------------------------------------------------------------------------------------------------------------------------------------------------

import pyttsx3

engine = pyttsx3.init()

# Voice IDs pulled from engine.getProperty('voices')

# These will be system specific

en\_voice\_id = "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS\_MS\_EN-US\_DAVID\_11.0"

ru\_voice\_id = "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS\_MS\_EN-US\_ZIRA\_11.0"

# Use male English voice

engine.setProperty('voice', en\_voice\_id)

engine.say('Hello with my new voice')

# Use female English voice

engine.setProperty('voice', ru\_voice\_id)

engine.say('hello with my very new voice')

engine.runAndWait()

engine.runAndWait()

----------------------------------------------------------------------------------------------------------------------------------------------------

We have used pyttsx3 for our speech recognition purpose.

You can refer more info: <https://pyttsx3.readthedocs.io/en/latest/engine.html#the-engine-factory>

B. SENTIMENTAL ANALYSIS-

Sentimental Analysis with TextBlob.

--------------------------------------Python libraries to be installed:---------------------------------------------------------------

- TextBlob

-Tweepy (twitter api -> for tweets extraction)

--------------------------------------------------------------------------------------------------------------------------------------------------

Program for sentimental analysis.

I Have attached a copy of output for the same.

Also sentimental analysis by including a file, or on writing a few sentences.!!

----------------------------------------------------------------------------------------------------------------------------------------------------

#pip install TextBlob

#pip install tweepy

#ML

#Sentimental Analysis :)

import tweepy

from textblob import TextBlob

'''

These keys- have to be regenerated to work.

'''

#my access keys from tweepy api

consumer\_key='OqDiXa4UipJfnoYbaGXmK'

consumer\_secret='FwfnuI4X8E6V6LCeF5zKDztZ3tQUBMrJblOhNbA0vk'

access\_token='1074542975726698498-IFICXr7XsTcBSR3ChfFbDFvV'

access\_token\_secret='BTd5YClRnKyPoHHf0H4JyNHlUVnQ0htbfdSs'

#authorization

auth=tweepy.OAuthHandler(consumer\_key,consumer\_secret)

auth.set\_access\_token(access\_token,access\_token\_secret)

api=tweepy.API(auth)

#search for keyword

public\_tweets=api.search('Hillary Clinton')

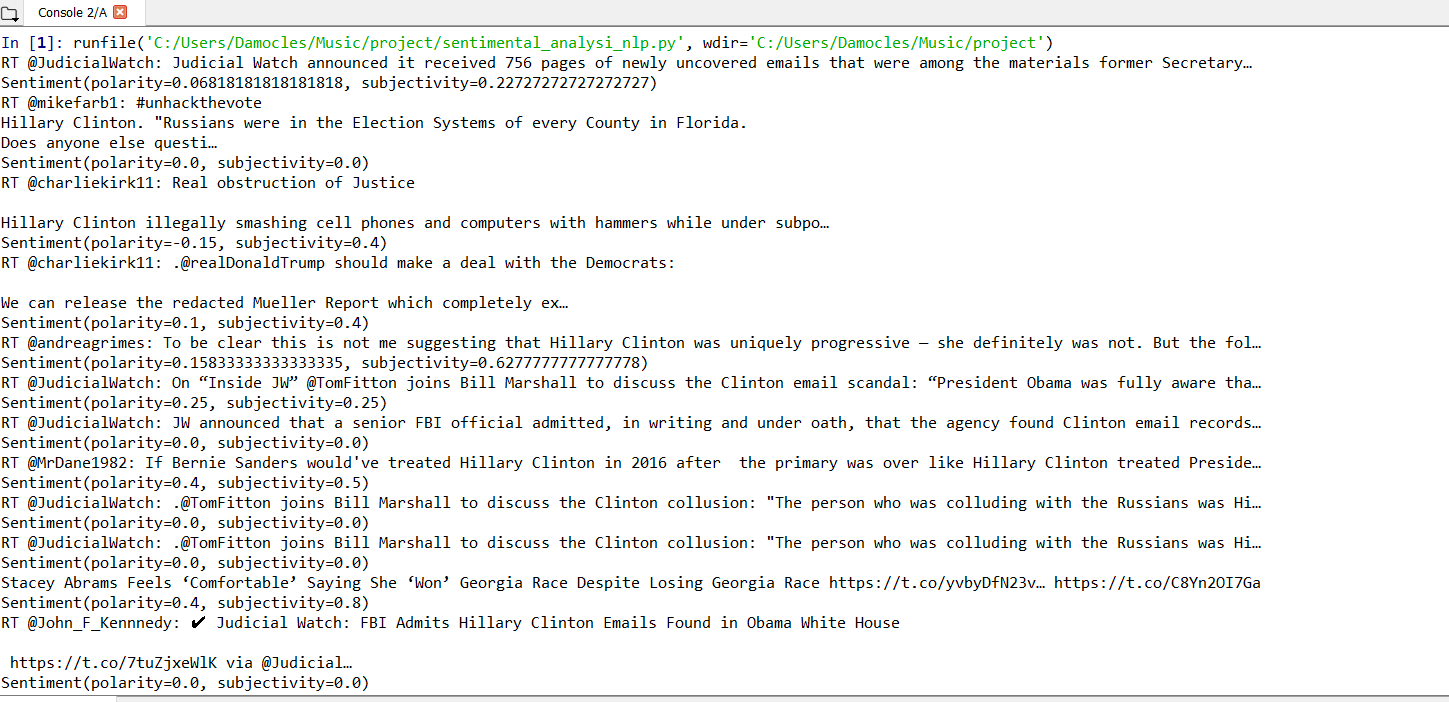
#sentimental analysis- polarity

for tweet in public\_tweets:

print(tweet.text)

analysis=TextBlob(tweet.text)

print(analysis.sentiment)



--------------------------------------------------------------------------------------------------------------------------

We have used TextBlob for our sentimental analysis purpose.

You may also try with spaCy.

D.TWEEPY (for microblogging)

Twitter Api --> Tweepy

--------------------------------------Python libraries to be installed:---------------------------------------------------------------

-Tweepy

--------------------------------------------------------------------------------------------------------------------------------------------------

First introduction Program for tweepy

Let's see what's trending today...

I Have attached the output below.

---------------------------------------------------------------------------------------------------------------------------------------------------

import tweepy

consumer\_key='OmDiXa4UipJfnoYbaGXmK'

consumer\_secret='FwfnuI4X6ss60tQG6LCeF5zKDztZ3tQUBMrJblOhNbA0vk'

access\_token='1074542975726698498HRpDTcBSR3ChfFbDFvV'

access\_token\_secret='BTd5YClRnKyPoHqsf0H4JyNHlUVnQ0htbfdSs'

auth=tweepy.OAuthHandler(consumer\_key,consumer\_secret)

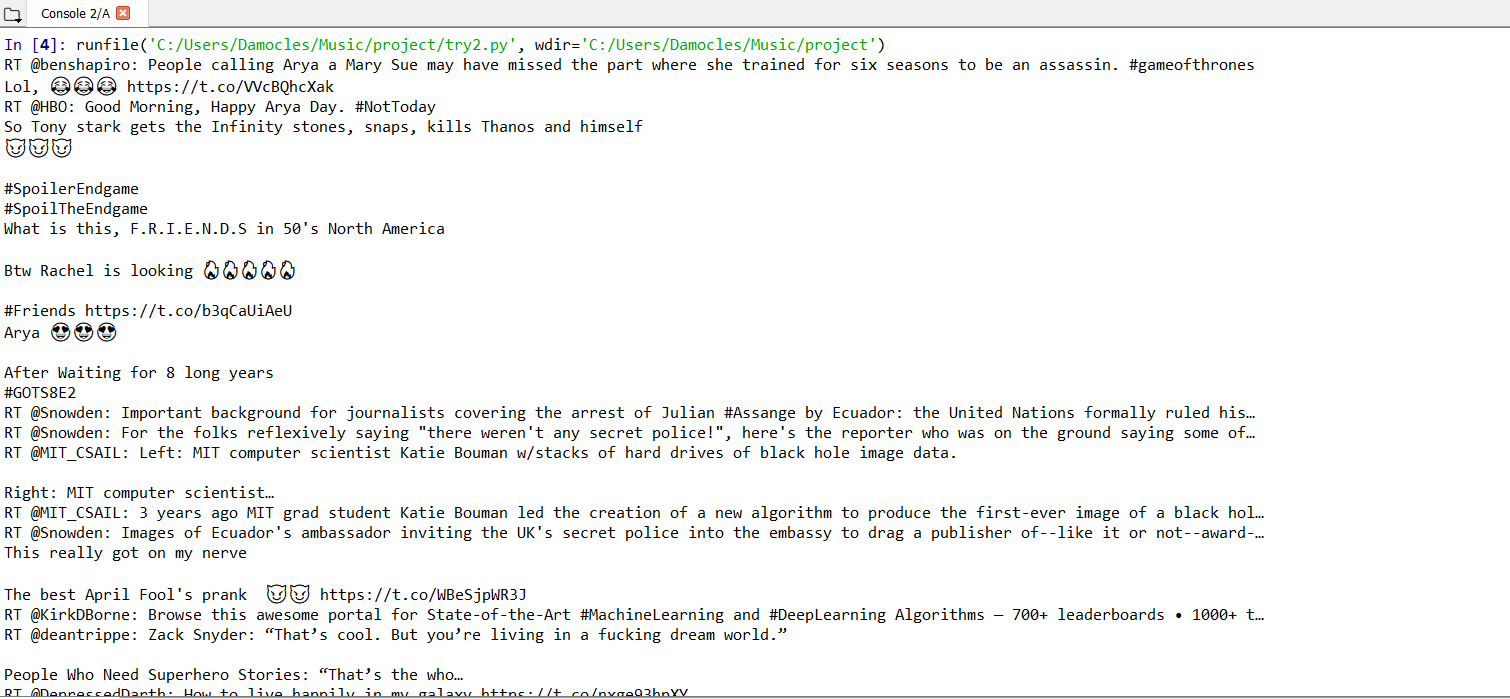
auth.set\_access\_token(access\_token,access\_token\_secret)

api=tweepy.API(auth)

public\_tweets=api.home\_timeline()

for tweet in public\_tweets:

print(tweet.text)



--------------------------------------------------------------------------------------------------------------------------

Streaming with tweepy.

Requires keyboard interrupt to stop streaming.

------------------------------------------------------------------------------------------------------------------------

import tweepy

consumer\_key='OmD72qDiXaXmK'

consumer\_secret='FwfnuI4X8E6V6ss6QUBMrJblOhNbA0vk'

access\_token='107454297XsV4HRpDTcBSR3ChfFbDFvV'

access\_token\_secret='BTd5YClRnKymRqsf0H4JyNHlUVnQ0htbfdSs'

auth=tweepy.OAuthHandler(consumer\_key,consumer\_secret)

auth.set\_access\_token(access\_token,access\_token\_secret)

api=tweepy.API(auth)

'''

#Therefore using the streaming api has three steps.

1.Create a class inheriting from StreamListener

2.Using that class create a Stream object

3.Connect to the Twitter API using the Stream.

'''

#override tweepy.StreamListener to add logic to on\_status

#creating a stream listener

class MyStreamListener(tweepy.StreamListener):

def on\_status(self, status):

print(status.text)

#creating a stream

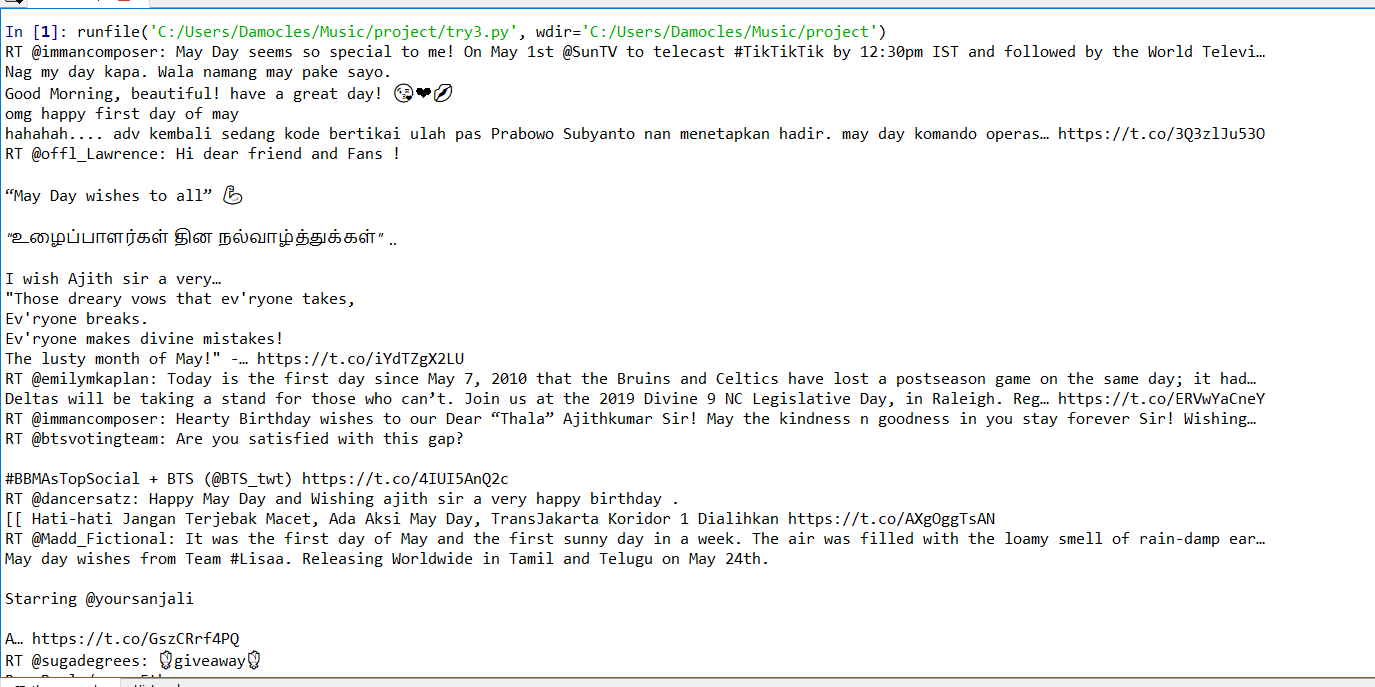
myStreamListener = MyStreamListener()

myStream = tweepy.Stream(auth = api.auth, listener=myStreamListener)

#starting a stream

#In this example we will use filter to stream all tweets containing the word May Day.

myStream.filter(track=['May Day'])



--------------------------------------------------------------------------------------------------------------------------------------------------------

As seen, the extracted tweets contain lots of unwanted words such as [#,@,.,The,etc......which](mailto:#,@,.,The,etc......which) keeps repeating.

Therefore, We need to clean our tweets, i.e. removing stopwords,to do lemmatization ,tokenization,etc...

We do cleaning of tweets with nltk library - for better summarization. :)

You can refer more info on tweepy API: <https://tweepy.readthedocs.io/en/latest/getting_started.html>

D. NLTK-

Let's get started with summarization now......

--------------------------------------Python libraries to be installed:---------------------------------------------------------------

- nltk (Natural Language Toolkit)

-------------------------------------------------------------------------------------------------------------------------------------------------

Introduction program for nlp tasks:

--------------------------------------------------------------------------------------------------------------------------------------------------

#just write in python- no need of any library.

text1="Ethics are built right into the ideals and objectives of the United Nations "

print(len(text1))

text2=text1.split(' ')

print(len(text2))

print(text2)

for w in text2:

if len(w)>3:

print(w)

for y in text2:

#word starting with captial letters

if y.istitle():

print(y)

for z in text2:

if z.endswith('s'):

print(z)

#just another simple code:

text5='ouagadougou'

text6=text5.split('ou')

print(text6)

text7=('ou'.join(text6))

print(text7)

print([list(text5)])

for c in text5:

print(c)

# program by using nltk library.

-------------------------------------------------------------------

import nltk

sentence = "Hey there, hope you are enjoying!!!"

tokens = nltk.word\_tokenize(sentence)

print(tokens)

---------------------------------------------------------------------------------------------------------------------------------------------

--------------------------------------Python libraries to be installed:---------------------------------------------------------------

- nltk (Natural Language Toolkit)

-sumy (for rouge score)

-------------------------------------------------------------------------------------------------------------------------------------------------

Output will be as summarized text docx. in your root directory folder...

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#pip install nltk

#pip install sumy

#we have done using parser...

import nltk

Max=0

#open a text document

with open("C:/Users/Damocles/Music/project/magic.txt",'r')as f:

for line in f:

Max+=1

print(Max)

from sumy.parsers.plaintext import PlaintextParser

from sumy.nlp.tokenizers import Tokenizer

file = "C:/Users/Damocles/Music/project/magic.txt"

parser = PlaintextParser.from\_file(file,Tokenizer("english"))

from sumy.summarizers.lex\_rank import LexRankSummarizer

summarizer = LexRankSummarizer()

summary = summarizer(parser.document, Max)

with open("LexRank.txt", "w") as f:

for sentence in summary:

f.write(str(sentence)+"\n")

from sumy.summarizers.luhn import LuhnSummarizer

summarizer\_1 = LuhnSummarizer()

summary\_1 = summarizer\_1(parser.document, Max)

with open("Luhn.txt", "w") as f:

for sentence in summary\_1:

f.write(str(sentence)+"\n")

from sumy.summarizers.lsa import LsaSummarizer

summarizer\_2 = LsaSummarizer()

summary\_2 = summarizer\_2(parser.document, Max)

with open("Lsa.txt", "w") as f:

for sentence in summary\_2:

f.write(str(sentence)+"\n")

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------