<u>CoVis: COVID-19 Twitter Sentiment Analysis & Extraction Visualizer</u>

1. Introduction

1.1 Overview

In this era of flourishing technology, Social Media has become a powerful platform for the public to voice their concerns and beliefs. Among them one such platform is Twitter. Twitter has been a popular platform for microblogging in the past few years. In this context, Sentiment Analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. Across the past few years, as the organizations and governments across the world start to adopt the ability to extract insights from social data, the applications of sentiment analysis are broad and powerful. There has been a clear implication that shifts in sentiment on social media correlate with shifts in the economics of a country and also the common notion among the public.

Due to the recent COVID-19 pandemic, there has been a wide change in sentiments of various sectors of the Indian public towards the government policies/actions. Studying the sentiment of the people on the epidemic and government decisions is very important as it acts as a sanity check for the effectiveness of the adopted government policies. This study also provides insight into the business models required to be adopted to suit this new age of post-COVID-19 where people's sentiments have widely changed. In this context 'Sentiment Analysis of COVID-19 Tweets' is a very important problem statement.

1.2 Purpose

The outbreak of COVID-19 caused heavy disruption to the everyday lives of people across the globe. In a country like with a large, diverse population like India, there are bound to be instances of mass hysteria and panic which are further fuelled by unreliable and sometimes inaccurate data. Gauging the feelings/emotions of the citizens would provide insights into the public mindset and would pave the way for the government and many organizations to address these situations by providing them with the right data and information, eradicating fake news, thereby helping in suppressing unnecessary panic among the people. Social media acts as the bridge between the people, the government, and such organizations. The scope of this project lies in the application of sentiment analysis to the views expressed by people on social media, twitter, in this case, to analyze the trends in the dynamic mood of the population. Usually, the terms "fight" and "positive" are used in a negative and positive context respectively, but we observe a role reversal in this situation. The identification of such terms and their usage according to the context would be an essential part of the project. Also, the scope of the project can be found in stopping the spread of fake news related to the pandemic, ***creating an interactive dashboard that delivers information about***

the current situation, real-time sentiment analysis of tweets, trend analysis of various COVID-19 related hashtags, engagement on Twitter, overall sector-wise polarity score of the tweets and the public emotion charts.

2. LITERATURE SURVEY

2.1 Existing Problem

The whole world is dealing with the coronavirus pandemic right now. But another crisis has also manifested itself along with the virus, which is as detrimental as the virus itself. That is the large flow of information regarding the virus in the form of tweets, blogs, news, and too little analysis of this tremendous amount of information flowing in and out of the system every moment. This information could sometimes be fake, mixed news or just some opinion about the pandemic. The spreading of such incomplete, inaccurate news would be the cause of mass hysteria and fear among the public and hence the need of the hour is to address and better understand this information crisis regarding the pandemic. The government, companies, and many organizations form their decisions to cater to the needs of the people, and hence understanding the right sentiment and opinions of the people towards the pandemic plays a crucial role in policymaking and creating appropriate business models that are of necessity to the people.

2.2 Proposed Solution

The exponential rise in the social media usage by the people in the last decade to express their opinions, perspectives and also as a source of news rather than the traditional news has laid emphasis on the usage of Deep Learning and Artificial Intelligence methods in gauging information from these sites to analyze and extract sentiments that act as valuable sources of insights for corporate companies and the government alike in making policies and appropriate business models to match the trend of the public. Hence, the main purpose of this project lies in the creation of a Public Sentiment Analysis Dashboard that depicts the sentiment trend among the Indian public towards the pandemic. Twitter is a microblogging site that has gained popularity in recent years, for the governments, organizations, and people alike, as a platform to make official announcements, expressing moods, opinions towards current, ongoing issues. Hence the tweets, that are the short posts that are made on this platform, act as our data set for creating this dashboard. The application of Deep Learning methods to analyze these tweets, gauge sentiments from them, and then representing these results on a live interactive dashboard is the main aim of this project.

A sentiment extraction model that analyses this data on a large scale and provides valuable insights into the sentiment trends of the public Recent language models such as BERT, XLNET, T5, Roberta, Electra have shown a great contextual understanding of language. So a transfer learning approach based on these models was used. A caveat that appeared in the context of COVID-19 is that few words portray a negative sentiment whereas originally in a normal context they convey a positive sentiment, for instance, the word ‘positive’. Instances like these were looked upon and it was found out after a certain amount of analysis that these words were not frequently used out of context as they were initially assumed to be. Also, another interesting challenge that was taken up was the process of extraction of common notions of public regarding COVID-19 actions taken by the Indian government and business corporations. To facilitate features to overcome these challenges during sentiment analysis, language models capable of simultaneous sentiment classification and extraction were developed. Also, the problem statement requires a good graphical user interface which provides keen insights into the public opinion on government and business decisions during COVID. For this, a user-friendly interactive dashboard that delivers information about the current situation, real-time sentiment analysis of tweets, trend analysis of various COVID-19 related hashtags, engagement on Twitter, overall sector-wise polarity score of the tweets and the public emotion charts was created. Thus, in this context, the sentiment extraction approach and the post prediction analysis have facilitated the easy extraction of the climate of opinion surrounding the COVID-19 pandemic.

3. Theoretical Analysis

3.1 Block Diagram

3.2 Hardware/ Software Design

Initially, the IEEE Coronavirus (COVID-19) Tweets Data set was downloaded from their website. Upon inspection, it was found that many tweets did not have geo-location tags, and also many were in different languages apart from English. Due to this challenge in obtaining proper data, a new data set named Geo-Tagged Coronavirus (COVID-19) Tweets Data set was obtained from the same website. These tweets were then hydrated using the “Hydrator” software and also a few python commands. Then, tweets in “English” and tweets from “India” were randomly chosen and a new dataset was created. Further, other data set containing COVID-19 related tweets from India were obtained from Kaggle. This data set was then cleaned and normalized to make it useful for further analysis.

This cleaned data were then subjected to further analysis by extracting bigrams, trigrams, and plotting frequency bar graphs, Word Clouds, Relationship Nexus, Boxplots, etc. This was done using both Python and R. Some Interactive Plots were also plotted. This complete process is termed as Exploratory Data Analysis.

After Exploratory Data Analysis is completed, the tweets are then Tokenized and are made in a format suitable for the Language Model. In this Step, two models were used:

1) Roberta Model: Transfer learning methods were implemented to carry out sentiment analysis. Sentiment Analysis of Tweets was carried out by integrating and using both the Huggingface Transformer Library and FastAI. Further **Slanted Triangular Learning Rates**, **Discriminate Learning Rate** and even **Gradual Unfreezing** were used, as a result of which, state-of-the-art results were obtained rapidly without even tuning the parameters. The tokenized data was then passed through the RoBERTa model to perform Sentiment Analysis. This yielded a model with an accuracy of 97% over the data set. The Tweepy API was used to scrape tweets in real-time which were then passed through the model to obtain the sentiments.
2) RoBERTa-CNN Sentiment Extractor: After the completion of the sentiment analysis the data was further explored for the sentiment triggers in the tweets. HuggingFace transformers don't have a TFRobertaForQuestionAnswering, for this purpose, a TFRobertaModel was created to convert trained data into arrays that the Roberta model can interpret. While training the Sentiment Extractor model, 5 stratified KFolds were used in such a way that, in each fold, the best model weights were saved and these weights were reloaded before carrying out testing and predictions. Roberta with CNN head was used for Twitter Sentiment Extraction. Thus after passing the data through this model we obtained a new column of the extracted text for the sentiments which was also used to plot certain graphs.

Now the entire process pertaining to the data and Model building is completed. Now, the Flask APP is built for the purpose of Deployment. First, the application is deployed on the localhost and debugged and then we move on to deploying on the WebServers.

A flask app was used for setting up website routing. It is used to integrate the back end machine learning models with the dashboard. Then Socketio (web sockets) were used for dynamic implementations on the website, namely the Real-Time Plot Generators and Twitter live feed. The basic functionality of the Flask Socketio lies in running background threads when the client is not connected to the website thereby enabling dynamic plotting. The above built Dashboard was deployed on the Local Machine and debugged for any possible errors. The scraping rate and other parameters were monitored and corrected accordingly.

Project Flask/Socketio APP with Sentiment Analyzer & Extractor Code:

07/15/20 09:22:37 C:\Users\Nikhil Keetha\Downloads\app\main.py

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pathlib import Path
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.model_selection import StratifiedKFold
import os

# pytorch
import torch
import torch.optim as optim

import random

# fastai
from fastai import *
from fastai.text import *
from fastai.callbacks import *

# transformers
from transformers import *
from transformers import PreTrainedModel, PreTrainedTokenizer, PretrainedConfig
from transformers import RobertaForSequenceClassification, RobertaTokenizer, RobertaConfig

# tensorflow
import tensorflow as tf
import tensorflow.keras.backend as K

import tokenizers
import pickle
import math
import re
import string
import seaborn as sns
color = sns.color_palette()
import matplotlib.pyplot as plt
from nltk import bigrams
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
eng_stopwords = stopwords.words('english')
import collections
from wordcloud import WordCloud
from textwrap import wrap
import plotly.graph_objects as go
import plotly.express as px
import networkx as nx

from tweepy import OAuthHandler
#from tweepy.streaming import StreamListener
import tweepy
import csv
import time
import datetime

from flask import Flask, request, jsonify, make_response
from flask import render_template, url_for, flash, redirect, copy_current_request_context
from flask_socketio import SocketIO, emit
from threading import Thread, Event

from flask_restful import reqparse, abort, Api, Resource
import json
from flask import jsonify

app = Flask(__name__)
app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba012'
app.config['DEBUG'] = True
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

socketio = SocketIO(app, async_mode=None, logger=True, engineio_logger=True)

# Twitter credentials
consumer_key = 'rm2bLDjA2BzljoA0GomL5o6W7'
consumer_secret = 'xiFBG4VKWPuQts1v3uqAesllpDp36y44YkFnzBtezSbSYW9dBV'
access_key = '935519854064418816-sOBxmFMaDygAx3FQXRBjH0drpZ2OXpB'
access_secret = 'GbOTefzapdet9vpmR3H9OBRuJNJNs1cI4Adh5HrkIYPJz'

# Pass your twitter credentials to tweepy via its OAuthHandler
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
t_api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)

class TransformersBaseTokenizer(BaseTokenizer):
    """Wrapper around PreTrainedTokenizer to be compatible with fast.ai"""
    def __init__(self, pretrained_tokenizer: PreTrainedTokenizer, model_type = 'bert', **kwargs):
        self._pretrained_tokenizer = pretrained_tokenizer
```

```python
        self.max_seq_len = pretrained_tokenizer.max_len
        self.model_type = model_type

    def __call__(self, *args, **kwargs):
        return self

    def tokenizer(self, t:str) -> List[str]:
        """Limits the maximum sequence length and add the spesial tokens"""
        CLS = self._pretrained_tokenizer.cls_token
        SEP = self._pretrained_tokenizer.sep_token
        if self.model_type in ['roberta']:
            tokens = self._pretrained_tokenizer.tokenize(t, add_prefix_space=True)[:self.max_seq_len - 2]
            tokens = [CLS] + tokens + [SEP]
        else:
            tokens = self._pretrained_tokenizer.tokenize(t)[:self.max_seq_len - 2]
            if self.model_type in ['xlnet']:
                tokens = tokens + [SEP] +  [CLS]
            else:
                tokens = [CLS] + tokens + [SEP]
        return tokens

class TransformersVocab(Vocab):
    def __init__(self, tokenizer: PreTrainedTokenizer):
        super(TransformersVocab, self).__init__(itos = [])
        self.tokenizer = tokenizer

    def numericalize(self, t:Collection[str]) -> List[int]:
        "Convert a list of tokens `t` to their ids."
        return self.tokenizer.convert_tokens_to_ids(t)
        #return self.tokenizer.encode(t)

    def textify(self, nums:Collection[int], sep=' ') -> List[str]:
        "Convert a list of `nums` to their tokens."
        nums = np.array(nums).tolist()
        return sep.join(self.tokenizer.convert_ids_to_tokens(nums)) if sep is not None else self.tokenizer.convert_ids_to_tokens(nums)

    def __getstate__(self):
        return {'itos':self.itos, 'tokenizer':self.tokenizer}

    def __setstate__(self, state:dict):
        self.itos = state['itos']
        self.tokenizer = state['tokenizer']
        self.stoi = collections.defaultdict(int,{v:k for k,v in enumerate(self.itos)})

# defining our model architecture
class CustomTransformerModel(nn.Module):
    def __init__(self, transformer_model: PreTrainedModel):
        super(CustomTransformerModel,self).__init__()
        self.transformer = transformer_model

    def forward(self, input_ids, attention_mask=None):

        # attention_mask
        # Mask to avoid performing attention on padding token indices.
        # Mask values selected in ``[0, 1]``:
        # ``1`` for tokens that are NOT MASKED, ``0`` for MASKED tokens.
        attention_mask = (input_ids!=pad_idx).type(input_ids.type())

        logits = self.transformer(input_ids,
                                  attention_mask = attention_mask)[0]
        return logits

def predict_sentiment(learner, text):
  sentiment = learner.predict(text)[1].item()
  return sentiment

def sentiment_label (Sentiment):
   if Sentiment == 2:
       return "positive"
   elif Sentiment == 0:
       return "negative"
   else :
       return "neutral"

def replace_url(string): # cleaning of URL
    text = re.sub(r'http\S+', 'LINK', string)
    return text


def replace_email(text):#Cleaning of Email related text
    line = re.sub(r'[\w\.-]+@[\w\.-]+','MAIL',str(text))
    return "".join(line)

def rep(text):#cleaning of non standard words
    grp = text.group(0)
    if len(grp) > 3:
        return grp[0:2]
```

```python
    else:
        return grp# can change the value here on repetition
def unique_char(rep,sentence):
    convert = re.sub(r'(\w)\1+', rep, sentence)
    return convert

def find_dollar(text):#Finding the dollar sign in the text
    line=re.sub(r'\$\d+(?:\.\d+)?','PRICE',text)
    return "".join(line)

def replace_emoji(text):
    emoji_pattern = re.compile("["
    u"\U0001F600-\U0001F64F" # emoticons
    u"\U0001F300-\U0001F5FF" # symbols & pictographs
    u"\U0001F680-\U0001F6FF" # transport & map symbols
    u"\U0001F1E0-\U0001F1FF" # flags (iOS)
    u"\U00002702-\U000027B0"
    u"\U000024C2-\U0001F251"
    "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'EMOJI', text)

puncts = [',', '.', '"', ':', ')', '(', '-', '!', '?', '|', ';', "'", '$', '&', '/', '[', ']',
          '>', '%', '=', '#', '*', '+', '\\', '•', '~', '@', '£', '·', '_', '{', '}', '©', '^',
          '®', '`', '<', '→', '°', '€', '™', '›', '♥', '←', '×', '§', '″', '′', 'Â', '█',
          '½', 'à', '…', '“', '★', '”', '–', '●', 'â', '►', '−', '¢', '²', '¬', '░', '¶',
          '↑', '±', '¿', '▾', '═', '¦', '║', '―', '¥', '▓', '—', '‹', '─', '▒', '：', '¼',
          '⊕', '▼', '▪', '†', '■', '’', '▀', '¨', '▄', '♫', '☆', 'é', '¯', '♦', '¤', '▲',
          'è', '¸', '¾', 'Ã', '⋅', '‘', '∞', '∙', '）', '↓', '、', '│', '（', '»', '，', '♪',
          '╩', '╚', '³', '・', '╦', '╣', '╔', '╗', '▬', '❤', 'ï', 'Ø', '¹', '≤', '‡', '√']

def clean_text(text: str) -> str:
    text = str(text)
    for punct in puncts + list(string.punctuation):
        if punct in text:
            text = text.replace(punct, f'')
    return text

def replace_asterisk(text):
    text = re.sub("\*", 'ABUSE ', text)
    return text

def remove_duplicates(text):
    text = re.sub(r'\b(\w+\s*)\1{1,}', '\\1', text)
    return text

def change(text):
    if(text == ''):
        return text
  #calling the subfunctions in the cleaning function
    text = replace_email(text)
    text = replace_url(text)
    text = unique_char(rep,text)
    text = replace_asterisk(text)
    text = remove_duplicates(text)
    text = clean_text(text)
    return text

def extract_tweets(search_words,date_since, date_until, numTweets):
  return(tweepy.Cursor(t_api.search, q=search_words, lang="en", since=date_since, until=date_until,
tweet_mode='extended').items(numTweets))

def scraptweets(search_words, date_since, date_until, numTweets, numRuns):
    # Define a pandas dataframe to store the date:
    db_tweets = pd.DataFrame(columns = ['username', 'acctdesc', 'location', 'following', 'followers', 'totaltweets',
'usercreatedts', 'tweetcreatedts', 'retweetcount', 'text', 'hashtags'])
    db_tweets['hashtags'] = db_tweets['hashtags'].astype('object')
    #db_tweets = pd.DataFrame()

    for i in range(numRuns):

        tweets = extract_tweets(search_words,date_since,date_until,numTweets)
        # Store these tweets into a python list
        tweet_list = [tweet for tweet in tweets]
        print(len(tweet_list))
        noTweets = 0

        for tweet in tweet_list:
            username = tweet.user.screen_name
            acctdesc = tweet.user.description
            location = tweet.user.location
            following = tweet.user.friends_count
            followers = tweet.user.followers_count
            totaltweets = tweet.user.statuses_count
            usercreatedts = tweet.user.created_at
            tweetcreatedts = tweet.created_at
            retweetcount = tweet.retweet_count
            hashtags = tweet.entities['hashtags']
```

```
            lst=[]
            for h in hashtags:
                lst.append(h['text'])
            try:
                text = tweet.retweeted_status.full_text
            except AttributeError:  # Not a Retweet
                text = tweet.full_text

            itweet =
[username,acctdesc,location,following,followers,totaltweets,usercreatedts,tweetcreatedts,retweetcount,text,lst]
            db_tweets.loc[len(db_tweets)] = itweet

            noTweets += 1

            #filename = "tweets.csv"
            #with open(filename, "a", newline='') as fp:
             #   wr = csv.writer(fp, dialect='excel')
              #  wr.writerow(itweet)

        if i+1 != numRuns:
            time.sleep(920)

        filename = "static/analysis.csv"
        db_tweets['text'] = db_tweets['text'].apply(change)
        db_tweets = db_tweets[['retweetcount', 'text']]
        # Store dataframe in csv with creation date timestamp
        db_tweets.drop_duplicates(subset ="text", keep = 'first', inplace = True)
        db_tweets.to_csv(filename, index = False) #
        print('Scrapping Done')

# Functions for Sentiment Extractor
def save_weights(model, dst_fn):
    weights = model.get_weights()
    with open(dst_fn, 'wb') as f:
        pickle.dump(weights, f)


def load_weights(model, weight_fn):
    with open(weight_fn, 'rb') as f:
        weights = pickle.load(f)
    model.set_weights(weights)
    return model

def loss_fn(y_true, y_pred):
    # adjust the targets for sequence bucketing
    ll = tf.shape(y_pred)[1]
    y_true = y_true[:, :ll]
    loss = tf.keras.losses.categorical_crossentropy(y_true, y_pred,
        from_logits=False, label_smoothing=LABEL_SMOOTHING)
    loss = tf.reduce_mean(loss)
    return loss
 #Global Constants for TF sentiment extractor
MAX_LEN = 310
PAD_ID = 1
num_splits = 1
SEED = 88888

PATH = 'static/Tf-Roberta/'
tokenizer = tokenizers.ByteLevelBPETokenizer(
    vocab_file=PATH+'vocab-roberta-base.json',
    merges_file=PATH+'merges-roberta-base.txt',
    lowercase=True,
    add_prefix_space=True
)

def build_model():
    ids = tf.keras.layers.Input((MAX_LEN,), dtype=tf.int32)
    att = tf.keras.layers.Input((MAX_LEN,), dtype=tf.int32)
    tok = tf.keras.layers.Input((MAX_LEN,), dtype=tf.int32)
    padding = tf.cast(tf.equal(ids, PAD_ID), tf.int32)

    lens = MAX_LEN - tf.reduce_sum(padding, -1)
    max_len = tf.reduce_max(lens)
    ids_ = ids[:, :max_len]
    att_ = att[:, :max_len]
    tok_ = tok[:, :max_len]

    config = RobertaConfig.from_pretrained(PATH+'config-roberta-base.json')
    bert_model = TFRobertaModel.from_pretrained(PATH+'pretrained-roberta-base.h5',config=config)
    x = bert_model(ids_,attention_mask=att_,token_type_ids=tok_)

    x1 = tf.keras.layers.Dropout(0.1)(x[0])
    x1 = tf.keras.layers.Conv1D(768, 2,padding='same')(x1)
    x1 = tf.keras.layers.LeakyReLU()(x1)
    x1 = tf.keras.layers.Dense(1)(x1)
    x1 = tf.keras.layers.Flatten()(x1)
    x1 = tf.keras.layers.Activation('softmax')(x1)

```

```
    x2 = tf.keras.layers.Dropout(0.1)(x[0])
    x2 = tf.keras.layers.Conv1D(768, 2,padding='same')(x2)
    x2 = tf.keras.layers.LeakyReLU()(x2)
    x2 = tf.keras.layers.Dense(1)(x2)
    x2 = tf.keras.layers.Flatten()(x2)
    x2 = tf.keras.layers.Activation('softmax')(x2)

    model = tf.keras.models.Model(inputs=[ids, att, tok], outputs=[x1,x2])
    optimizer = tf.keras.optimizers.Adam(learning_rate=3e-5)
    model.compile(loss=loss_fn, optimizer=optimizer)

    # this is required as `model.predict` needs a fixed size!
    x1_padded = tf.pad(x1, [[0, 0], [0, MAX_LEN - max_len]], constant_values=0.)
    x2_padded = tf.pad(x2, [[0, 0], [0, MAX_LEN - max_len]], constant_values=0.)

    padded_model = tf.keras.models.Model(inputs=[ids, att, tok], outputs=[x1_padded,x2_padded])
    return model, padded_model

def generate_wordcloud(data,title):
  wc = WordCloud(width=400, height=330, max_words=150,colormap="Dark2",background_color='white', collocations=False).generate_from_frequencies(data)
  plt.figure(figsize=(10,8))
  plt.imshow(wc, interpolation='bilinear')
  plt.tight_layout(pad=0)

@app.route("/",methods=['GET', 'POST'])
@app.route("/dashboard", methods=['GET', 'POST'])
def home():
    return render_template('dashboard.html')

@app.route("/live_case_count", methods=['GET', 'POST'])
def live_count():
    return render_template('live_case_count.html')

@app.route("/about_us", methods=['GET', 'POST'])
def about():
    return render_template('about_us.html')

HOUR = 3600;

thread = Thread()
thread_stop_event = Event()

model_type = 'roberta'
pretrained_model_name = 'roberta-base'

model_class, tokenizer_class, config_class = RobertaForSequenceClassification, RobertaTokenizer, RobertaConfig

transformer_tokenizer = tokenizer_class.from_pretrained(pretrained_model_name)
transformer_base_tokenizer = TransformersBaseTokenizer(pretrained_tokenizer = transformer_tokenizer, model_type = model_type)
fastai_tokenizer = Tokenizer(tok_func = transformer_base_tokenizer, pre_rules=[], post_rules=[])

pad_idx = transformer_tokenizer.pad_token_id

p = 'static/Roberta_Model'
learner = load_learner(p, 'transformer.pkl')

def plotGenerator():
    """
    Generates real time plots every 1 day.
    """
    while not thread_stop_event.isSet():
        # Initialise these variables:
        print('Code is Running!!!!!')

        search_words = "(#India AND #COVID-19) OR #COVID19India"
        yesterday = datetime.datetime.now() - datetime.timedelta(days = 1)
        date_since = yesterday.strftime("%Y-%m-%d")
        date_until = datetime.datetime.today().strftime('%Y-%m-%d')
        numTweets = 2000
        numRuns = 1
        # Call the function scraptweets
        program_start = time.time()
        scraptweets(search_words, date_since, date_until, numTweets, numRuns)
        program_end = time.time()

        path = 'static/analysis.csv'
        predictions = pd.read_csv(path)

        print('Start Prediction')

        predictions['Prediction'] = predictions['text'].apply(lambda x: predict_sentiment(learner, x))
        predictions['Prediction'] = predictions['Prediction'].apply(sentiment_label)
        class_names = ['negative','positive','neutral']

        predictions.rename(columns={'Prediction':'sentiment'},inplace=True)

```

```
        print('Predictions Done')

        path = 'static/analysis.csv'
        predictions.to_csv(path,index=False)
        test = pd.read_csv(path)

        MAX_LEN = 310
        PAD_ID = 1
        num_splits = 1
        SEED = 88888

        PATH = 'static/Tf-Roberta/'
        tokenizer = tokenizers.ByteLevelBPETokenizer(
            vocab_file=PATH+'vocab-roberta-base.json',
            merges_file=PATH+'merges-roberta-base.txt',
            lowercase=True,
            add_prefix_space=True
        )

        test['len'] = test['text'].str.len()
        test = test[test['len']<=310]
        test.drop("len",axis=1,inplace=True)
        test.reset_index(drop=True, inplace=True)

        ct = test.shape[0]
        input_ids_t = np.ones((ct,MAX_LEN),dtype='int32')
        attention_mask_t = np.zeros((ct,MAX_LEN),dtype='int32')
        token_type_ids_t = np.zeros((ct,MAX_LEN),dtype='int32')
        sentiment_id = {'positive': 1313, 'negative': 2430, 'neutral': 7974}

        for k in range(test.shape[0]):

            # INPUT_IDS
            text1 = " "+" ".join(test.loc[k,'text'].split())
            enc = tokenizer.encode(text1)
            s_tok = sentiment_id[test.loc[k,'sentiment']]
            input_ids_t[k,:len(enc.ids)+3] = [0, s_tok] + enc.ids + [2]
            attention_mask_t[k,:len(enc.ids)+3] = 1

        DISPLAY=1 # USE display=1 FOR INTERACTIVE
        preds_start = np.zeros((input_ids_t.shape[0],MAX_LEN))
        preds_end = np.zeros((input_ids_t.shape[0],MAX_LEN))

        # for fold in range(0,5):
        K.clear_session()
        model, padded_model = build_model()
        path = 'static/R_CNN_weights/'
        weight_fn = path+'v0-roberta-0.h5'

        print('Loading model...')
        # model.load_weights('%s-roberta-%i.h5'%(VER,fold))
        load_weights(model, weight_fn)

        print('Predicting Test...')

        preds = padded_model.predict([input_ids_t,attention_mask_t,token_type_ids_t],verbose=DISPLAY)
        preds_start += preds[0]/num_splits
        preds_end += preds[1]/num_splits

        all = []
        for k in range(input_ids_t.shape[0]):
            a = np.argmax(preds_start[k,])
            b = np.argmax(preds_end[k,])
            if a>b:
                st = test.loc[k,'text']
            else:
                text1 = " "+" ".join(test.loc[k,'text'].split())
                enc = tokenizer.encode(text1)
                st = tokenizer.decode(enc.ids[a-2:b-1])
            all.append(st)

        test['selected_text'] = all
        test.to_csv('static/analysis.csv',index=False)

        print('Extraction Done')

        #Plots start
        data=pd.read_csv("static/analysis.csv")
        data['selected_text'] = data['selected_text'].astype(str)
        df = data.sentiment.value_counts()
        size = list(df.values)
        names = list(df.index)
        fig = plt.figure(figsize=(10,10))
        plt.xlabel("Sentiment",fontsize = 16)
        plt.ylabel("Frequency",fontsize = 16)
        sns.barplot(names,size,alpha = 0.8)
        fig.savefig("static/images/realtime/bar.png")
```

```
        df_new = pd.DataFrame(dict(
            r=list(df.values),
            theta=list(df.index)))
        plt.figure(figsize=(10,10))
        fig = px.line_polar(df_new, r='r', theta='theta', line_close=True)
        fig.update_traces(fill='toself')
        fig.write_image("static/images/realtime/radar_plot.png")

        # Pie chart
        labels = list(df.index)
        sizes = list(df.values)
        # only "explode" the 2nd slice
        explode = (0.1, 0.1, 0.1)
        #add colors
        colors = ['#ff9999','#66b3ff','#99ff99']
        fig1, ax1 = plt.subplots()
        ax1.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
                shadow=True, startangle=90)
        # Equal aspect ratio ensures that pie is drawn as a circle
        ax1.axis('equal')
        plt.tight_layout()
        plt.savefig('static/images/realtime/pie_chart.png')

        for i in range(3):
          Data= data[data["sentiment"]==df.index[i]]
          Word_frequency = pd.Series(' '.join(Data.selected_text).split()).value_counts()[:20]#Calculating the words frequency
          plt.figure(figsize=(25,10))
          plt.ylabel("Frequency",fontsize=16)
          plt.title("Sentiment Triggers")
          sns.barplot(Word_frequency.index,Word_frequency.values,alpha=0.8)
          plt.savefig("static/images/realtime/wordfrequency_"+df.index[i]+".png")

        for i in range(3):
            Analysis_Data = data
            data["selected_text"]=data["selected_text"].apply(lambda x: ' '.join([word for word in x.split() if word not in (eng_stopwords)]))
            Sentiment = Analysis_Data[Analysis_Data['sentiment'] == df.index[i]]#Creating the dataframe of having same sentiment
            Word_frequency = pd.Series(' '.join(Sentiment.selected_text).split()).value_counts()[:50]#Calculating the words frequency
            generate_wordcloud(Word_frequency.sort_values(ascending=False),data.index[i])
            plt.savefig("static/images/realtime/Wordcloud_" +df.index[i]+".png")

        data["text"]=data["text"].apply(lambda x: ' '.join([word for word in x.split() if word not in (eng_stopwords)]))
        bigrams = [b for l in data.text for b in zip(l.split(" ")[:-1], l.split(" ")[1:])]
        bigram_counts = collections.Counter(bigrams)
        bigram_df = pd.DataFrame(bigram_counts.most_common(10),
                                 columns=['bigram', 'frequency'])
        x =bigram_df.bigram
        y = bigram_df.frequency

        fig, ax = plt.subplots(1, 1, figsize = (20, 15), dpi=300)
        sns.barplot(x,y,alpha=0.8)
        plt.ylabel("Frequency",fontsize=16)
        ax.set_xlabel('')
        plt.savefig('static/images/realtime/bigram_freq.png')

        ext_data_negative = data[data["sentiment"]=='negative']
        ext_data_positive = data[data["sentiment"]=='positive']
        bigrams = [b for l in ext_data_positive.selected_text for b in zip(l.split(" ")[:-1], l.split(" ")[1:])]
        bigram_counts = collections.Counter(bigrams)
        bigram_df_positive = pd.DataFrame(bigram_counts.most_common(60),
                                 columns=['bigram', 'frequency'])
        bigrams = [b for l in ext_data_negative.selected_text for b in zip(l.split(" ")[:-1], l.split(" ")[1:])]
        bigram_counts = collections.Counter(bigrams)
        bigram_df_negative = pd.DataFrame(bigram_counts.most_common(80),
                                 columns=['bigram', 'frequency'])

        # Create network plot
        G=nx.grid_2d_graph(2,2)

        pos = nx.fruchterman_reingold_layout(G,k=10,iterations=100)
        fig,ax = plt.subplots(figsize=(50,30))
        d = bigram_df_negative.set_index('bigram').T.to_dict('records')
        for k, v in d[0].items():
            G.add_edge(k[0], k[1], weight=(v * 10))
        pos = nx.fruchterman_reingold_layout(G,k=10,iterations=100)

        nx.draw_networkx(G, pos,
                         font_size=16,
                         width=4,
                         edge_color='#e25a4b',
                         node_size=500,
                         title = "Negative Sentiment",
                         with_labels = False,
                         ax=ax)
```

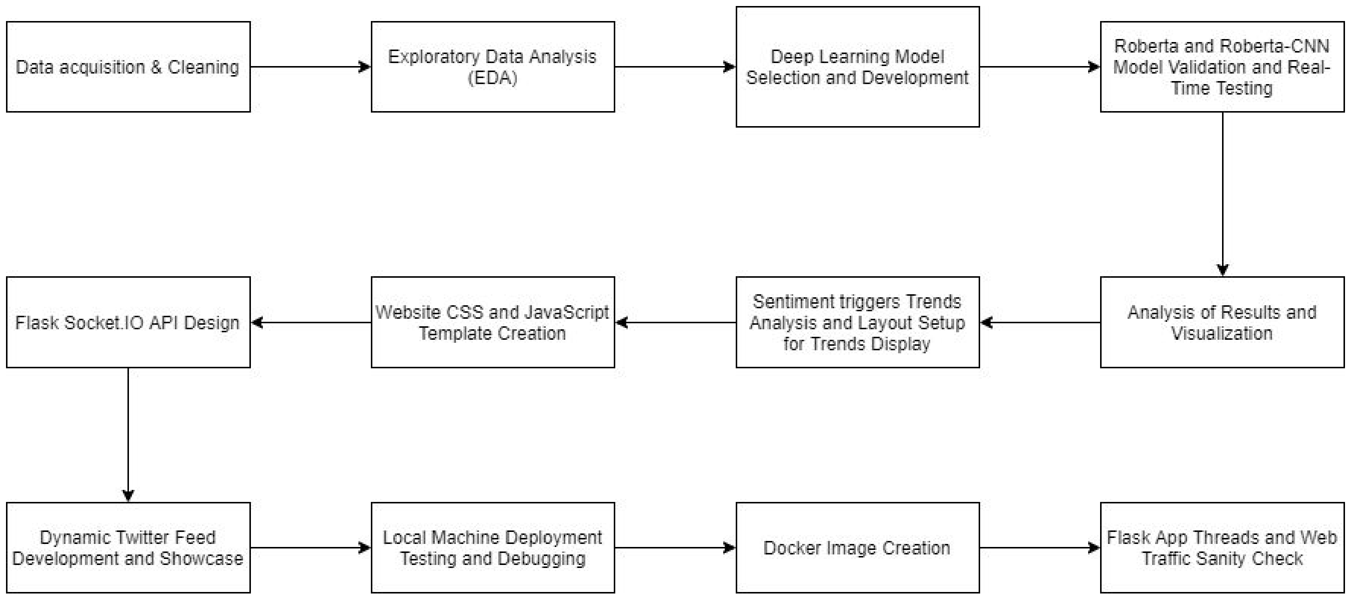
```
        x_values, y_values = zip(*pos.values())
        x_max = max(x_values)
        x_min = min(x_values)
        x_margin = (x_max - x_min) * 0.25
        plt.xlim(x_min - x_margin, x_max + x_margin)

        for key, value in pos.items():
            x, y = value[0]+.135, value[1]+.045
            ax.text(x, y,
                    s=key,bbox=dict(facecolor='#ffcd94', alpha=0.4),
                    horizontalalignment='center', fontsize=35)
        plt.savefig("static/images/realtime/ext_negative.png")
        fig,ax  = plt.subplots(figsize=(50,30))
        d = bigram_df_positive.set_index('bigram').T.to_dict('records')
        for k, v in d[0].items():
            G.add_edge(k[0], k[1], weight=(v * 10))
        pos = nx.fruchterman_reingold_layout(G,k=10,iterations=100)

        nx.draw_networkx(G, pos,
                         font_size=16,
                         width=4,
                         edge_color='#999894',
                         node_size=500,
                         with_labels = False,
                         title = "Positve Sentiment",
                         ax=ax)
        x_values, y_values = zip(*pos.values())
        x_max = max(x_values)
        x_min = min(x_values)
        x_margin = (x_max - x_min) * 0.25
        plt.xlim(x_min - x_margin, x_max + x_margin)

        # Create offset labels
        for key, value in pos.items():
            x, y = value[0]+.135, value[1]+.045
            ax.text(x, y,
                    s=key,bbox=dict(facecolor='#7c99d0', alpha=0.4),
                    horizontalalignment='center', fontsize=35)
        plt.savefig("static/images/realtime/ext_positive.png")

        data["text"]=data["text"].apply(lambda x: ' '.join([word for word in x.split() if word not in
(eng_stopwords)]))
        bigrams = [b for l in data.text for b in zip(l.split(" ")[:-1], l.split(" ")[1:])]
        bigram_counts = collections.Counter(bigrams)
        bigram_df = pd.DataFrame(bigram_counts.most_common(60),
                                 columns=['bigram', 'frequency'])

        d = bigram_df.set_index('bigram').T.to_dict('records')
        # Create network plot
        G = nx.Graph()
        for k, v in d[0].items():
            G.add_edge(k[0], k[1], weight=(v * 10))

        fig,ax = plt.subplots(figsize=(20,20))
        pos = nx.spring_layout(G,dim=2,k=5)

        # Plot networks
        nx.draw_networkx(G, pos,
                         font_size=12,
                         width=4,
                         edge_color='grey',
                         node_color='#4a4140',
                         node_size=500,
                         with_labels = False,
                         ax=ax)
        x_values, y_values = zip(*pos.values())
        x_max = max(x_values)
        x_min = min(x_values)
        x_margin = (x_max - x_min) * 0.25
        plt.xlim(x_min - x_margin, x_max + x_margin)


        # Create offset labels
        for key, value in pos.items():
            x, y = value[0]+.135, value[1]+.045
            ax.text(x, y,
                    s=key,
                    bbox=dict(facecolor='#ffcd94', alpha=0.4),
                    horizontalalignment='center', fontsize=25)

        fig.savefig('static/images/realtime/network.png')

        fig = px.box(data, y="retweetcount",points="all")
        fig.update_layout(
            yaxis_title="Retweet Count",
            font=dict(
                family="Courier New, monospace",
                size=18,
```

```
                color="#7f7f7f"
            )
        )
        fig.write_image('static/images/realtime/retweet_count_boxplot.png')

        fig = px.box(data, y="sentiment",points="all")
        fig.update_layout(
            yaxis_title="Sentiment",
            font=dict(
                family="Courier New, monospace",
                size=18,
                color="#7f7f7f"
            )
        )
        fig.write_image('static/images/realtime/sentiment_boxplot.png')

        print('Plotting Done.')

        socketio.sleep(24*HOUR)

@app.route('/real_time_analysis', methods=['GET', 'POST'])
def realtime():
    #only by sending this page first will the client be connected to the socketio instance
    return render_template('real_time_analysis.html')

@socketio.on('connect', namespace='/test')
def test_connect():
    # need visibility of the global thread object
    global thread
    print('Client connected')
    #Start the plot generator thread only if the thread has not been started before.
    if not thread.isAlive():
        print("Starting Thread")
        thread = socketio.start_background_task(plotGenerator)

@socketio.on('disconnect', namespace='/test')
def test_disconnect():
    print('Client disconnected')

twitter_thread = Thread()
twitter_thread_stop_event = Event()

def gettweets():

    while not twitter_thread_stop_event.isSet():
        print('Tweet feed starts')
        for tweet in tweepy.Cursor(t_api.search,q="#" + "COVID19India"  + " -filter:retweets",rpp=5,lang="en", tweet_mode='extended').items(20):
            temp = {}
            text = change(tweet.full_text)
            temp["text"] = text
            temp["username"] = tweet.user.screen_name
            text = change(tweet.full_text)
            prediction = predict_sentiment(learner,text)
            prediction = sentiment_label(prediction)
            temp["label"] = prediction
            socketio.emit('tweets', {'Text': temp['text'], 'Username': temp['username'], 'Sentiment': temp['label']}, namespace='/twitter')
        print('Twitter feed end')
        socketio.sleep(300)


@app.route("/twitter_live_feed", methods=['GET', 'POST'])
def twitter():
    return render_template('twitter_live_feed.html')

@socketio.on('connect', namespace='/twitter')
def twitter_connect():
    # need visibility of the global thread object
    global twitter_thread
    print(' Twitter Client connected')

    #Start the tweet feed thread only if the thread has not been started before.
    if not twitter_thread.isAlive():
        print("Starting Twitter Thread")
        thread = socketio.start_background_task(gettweets)

@socketio.on('disconnect', namespace='/twitter')
def twitter_disconnect():
    print('Twitter Client disconnected')

#Run APP

if __name__ == '__main__':
    socketio.run(app)
```

4. Experimental Investigations

During Exploratory Data Analysis (EDA), various types of graphs based on the sentiments and sentiment triggers were plotted to gain valuable insights from the data. The frequency distribution graphs give us a good idea about the data and also gives us an insight into predicting the model's generalization capability. From the frequency graphs and trends, it was clearly evident that the most frequently used terms by Twitterati in India and the rest of the world were "Corona" and "COVID-19" when compared to the other words. "rt", the most frequently used term in the world dataset, means "retweet". The high usage of the word in this pandemic could be attributed to the fact that people showed great enthusiasm in retweeting the tweets related to coronavirus. Also, the trigger “link” had many appearances in the graphs, this is because the trigger “link” was the code word used by the model in place of the URLs present in the tweets, this “link” has many appearances in the data set. From this, it can be understood that the model should not generalize and classify tweets based on the presence of the word "Corona", "COVID-19" and "rt". The trends from the graphs plotted further strengthen the belief that the Indian dataset is similar to the world dataset in terms of sentiments expressed.
The initial assumption about terms like positive and negative conveying the opposite meaning as opposed to the observed trends after the analysis and hence the assumption was dropped. Words like "fat" and "sick" have frequent occurrences in tweets, showing that people started paying attention to their health more than they previously did. "Love" was the most frequently used positive term and "nasty" was the most used negative term.By the nexus relationship graphs, we received better insights into the relationships between the sentiment triggers.

5. Flow Chart



6. Result

CoVis - The COVID-19 Twitter Sentiment Analysis Dashboard is the final product of the project. The dashboard contains many interactive tabs depicting various features like "Public Sentiment Dashboard", "Real-Time Analysis Dashboard", "Twitter Live Feed & Analysis" and "Live Case Counts" with live, robust, interactive graphs.

- *Public Sentiment Dashboard* - This tab contains an analysis of the tweets made by Twitterati in India in the past 90 days on dates when announcements like "Nation-wide Lockdown", "Unlock 1.0", etc. were made by the Government of India. These tweets were collecting on the basis of the hashtags used in the tweets. These tweets were then passed to the Roberta model created to extract sentiments. Finally, these sentiments were represented in the form of various graphs on the dashboard.
- *Real-Time Analysis Dashboard* - Tweets are scraped from Twitter based on hashtags given as input by using the Tweepy API. A total of 2,500 tweets are scraped (which is the maximum scraping limit per day for the Tweepy API), their sentiments are extracted and live, interactive graphs are plotted which are updated every 24 hours.
- *Twitter Live Feed & Analysis* - A few tweets are scraped using the Tweepy API every 5 minutes and their Sentiment Analysis is done. This page has a live scrolling of the tweets extracted, an interactive 3-D plot depicting the sentiment frequency of those tweets, and a block that displays the twitter user ID, the tweet body, and their extracted sentiment. This page gets updated on its own, every 5 minutes.
- *Live Case Counts* - This dashboard contains an overview of the live COVID-19 statistics of India in a state-wise manner. It has categories of "Active cases", "Recovered", "Deaths" and "Number of tests", which gets updated periodically.

7.ADVANTAGES & DISADVANTAGES

Advantages

The effect of COVID-19 pandemic is visible all over the world. National healthcare systems are facing the contagion with incredible strength, but concern regarding psychosocial and economic effects is critically growing. In a fast-moving crisis, as information swarms in from every direction, citizens look to their governments for information, guidance, and leadership. Sentimental Analysis is only the option in this current situation to understand the psychological condition/mental condition of the public. By Sentimental Analysis, the public opinion on COVID-19, regime policies, and actions can be understood. After Analysis, amendments can be made to the decisions taken by the regime policies, and the public can be fortified in such a way so as to enhance the sentiment towards a positive outlook. Not only this but also sentiment analysis will help NGOs and various organizations to come forward to help the people. Businesses can adapt their products and services to match the requirements of the people based on the real-time trending mood of the public, which will not only help businesses to grow but will also help the public meet their need of the hour. Also, this will enable the government to make business and people-friendly rules and laws to help in the betterment of the economy and the market in these untested times.

Disadvantages

Natural Language Processing models, in general, face a problem in recognizing human aspects of a language like irony, sarcasm, negotiations, exaggerations, and jokes - the sorts of things humans wouldn't face many problems in understanding. Machines sometimes fail in recognizing these aspects, which leads to skewed and incorrect results.

8. APPLICATIONS

The application of Sentiment Analysis lies in taking the sentiments of the people into consideration and creating appropriate business strategies and government policies so as to meet their needs. For example, many food delivery companies like Swiggy and online retail stores like Amazon are now opting for "Contactless Delivery" as a preventive step towards controlling the spread of the virus. This step taken by such companies would not only increase their profits but also would provide the essential services to the people while taking their sentiments into consideration in this pandemic. Not only this but also based on the sentiments analyzed form the tweets, rehabilitation or positive suggestions can be made to users who have consistently expressed negative sentiments. This way, public health can also be monitored using sentiment analysis.

9. CONCLUSION

Sentiment analysis or opinion mining is a hot topic in deep learning. There is still a long way to go before sentiments can be accurately detected from texts, because of the complexity involved in the English language, and even more when other languages like Hindi are considered. Though the Roberta model developed as a part of this project has predicted and classified the sentiments of the test data set into positive, negative and neutral categories with an accuracy of 97%, by making necessary modifications and additions to the model, sentiment analysis can be done with greater accuracy by taking the language complexities into consideration.

10. FUTURE SCOPE

Sentiment Analysis still has many aspects that can still be worked upon. This project can be further enhanced by training the model about the human aspects of a language and making it more accurate in cases where sarcasm, irony, and other aspects are used. Taking the actions of all the ministries into consideration while gauging the sentiments of the public can also make the analysis more detailed and sector-specific, which would help in the analysis of the area of development required in those sectors. The classifier can be further improved by trying to extract more features from the tweets, trying different kinds of features, tuning the Hyperparameters, and also by making it work on various Indian languages.

11. BIBLIOGRAPHY

1.https://towardsdatascience.com/twitter-data-collection-tutorial-using-python-3267d7cfa93e (Data collection)
2.https://ieee-dataport.org/open-access/coronavirus-covid-19-tweets-dataset (Datasets)
3.https://www.kaggle.com/ragnisah/text-data-cleaning-tweets-analysis (Data cleaning)
4.https://www.youtube.com/watch?v=ZhyLgPnOeh0 (Normalization)
5.https://cognitiveclass.ai/courses/data-visualization-with-python (Data Visualization)
6.https://github.com/DocNow/hydrator/blob/master/README.md (Hydrator)
7.https://dphi.tech/data-science-bootcamp-day-15-exploratory-data-analysis/ (Exploratory Data Analysis)
8.https://medium.com/swlh/understand-tweets-better-with-bert-sentiment-analysis-3b054c4b802a (BERT)
9.https://towardsdatascience.com/fasttext-sentiment-analysis-for-tweets-a-straightforward-guide-9a8c070449a2 (FastText)
10.https://medium.com/swlh/using-xlnet-for-sentiment-classification-cfa948e65e85 (XLNet(This given paper is only for Binary Classification.))
11.https://www.creative-tim.com/product/black-dashboard(Dashboards)
12.https://link.medium.com/ZZvca1Mf27 (Twitter live feed Sentimental Analysis)
13.https://flask-socketio.readthedocs.io/en/latest/(Flask Socket.IO)

14.https://github.com/covid19india/api (Live case count)

APPENDIX

A. Source code

https://github.com/SmartPracticeschool/SBSPS-Challenge-2700-Twitter-Sentiment-Analysis-Extraction-for-COVID-19