CS4624
Multimedia, Hypertext, and Information Access

May 10, 2022

COVID-19FakeNews

Virginia Tech, Blacksburg VA 24061

Instructor: Dr. Edward Fox
Client: Dr. Mohamed Farag
Team: Eric Wiley, Tung Nguyen, Fareeza Zameer, Ferrin Kirby, Kyle Toroc, Campbell Dalen

Table of Contents

Table of Tables

Table of Figures

1.0 Abstract

COVID is a virus that rampages through every country, from rural to urban areas. Since the beginning of the virus, facts and science have been politicized to align with party agendas which have unfortunately resulted in constituents being misinformed about the dangerous virus. From early April 2020 to early May 2020, Dr. Mohamed Farag collected a large set of tweets from users on Twitter. In these tweets, Twitter users expressed their thoughts, opinions, and facts on the virus. We aimed to filter these tweets, sort them into classes, and utilize machine learning to determine if these tweets, and future tweets that are to come, are a reliable source of accurate information or not.

Our goal in this project was to find rumors and false information that is spread about COVID as well as the perpetrators that spread this information. As more people around the world gain access to the internet, more people will continue spreading information and this results in an information “overload” where facts and myth are intertwined, and the public is unaware of the real truth. The COVID19FakeNews team focused on contributing to providing clarity to the public about which tweets spread dangerous lies.

We received a one terabyte file, filled with tweets, that Dr. Farag had collected. We converted these tweets into a unified format and stored them into a readable JSON format. We did this by making a Python script that utilizes different libraries associated with Python. We extracted the tweet IDs from the stored tweets collected, and, using the Twarc2 library, we were able to hydrate still existing – i.e., not deleted – tweets using the tweet ID that we extracted from the collection. This was crucial for finding currently visible tweets, so we can sort into future categories (buckets).

Once hydrated, a small sample of tweets was labeled into seven different categories by our team. These labels were then leveraged to train and test a machine learning model using SVM through the sklearn Python library. The model was trained with sufficient data so that the group would be satisfied with its accuracy. Then, the model was run on the remaining hydrated tweets, and we were able to classify those tweets. We created a front-end display to show the timeline of when different classes of tweets were published. The front-end also shows statistics on the raw and clean datasets, as well as users that have tweeted misinformation regularly. Overall, this project should be useful for researchers who are doing similar studies. It should also be useful to members of the public who are concerned about COVID.

2.0 Introduction

The data for this project was collected from a one terabyte file from our client Dr. Farag. We converted these tweets into a consistent format and stored them into a readable JSON file.

2.1 Objective

COVID misinformation has spread through social media. The overall objective of the project was to take in a dataset of tweets posted during the timeframe of April-May 2020 and then to filter, clean, and analyze them. This was to help people understand the misinformation that is taking place through social media as well as the direct sources of misinformation. After the direct sources of the misinformation were identified, a web page was built for users to sift through different statistics and view a timeline of when these tweets were tweeted out.

2.2 Deliverables

The purpose of this project was to distinguish misinformation regarding COVID-19 on social media. The following deliverables were produced:

1. A fully functioning Python script that can run the preprocessing and analysis steps
2. A web application that can show the analysis results

2.3 Client

The client for the project was Dr. Mohamad Farag who is a postdoctoral researcher at the Virginia Tech Center of Sustainable Mobility. He also has a background in being a researcher at the Digital Library Research Laboratory at Virginia Tech. Apart from his background in research he also has an interest in information retrieval, digital libraries, semantic web, and parallel computing.

2.4 Team

Our team consisted of the members Eric Wiley, Tung Nguyen, Ferrin Kirby, Campbell Dalen, Kyle Toroc, and Fareeza Zameer. All members are studying computer science and are all seniors except for Kyle who is a junior.

3.0 Requirements

In this section, we introduce and describe what was needed for this project to be considered complete.

3.1 Clean Given Data

The first important requirement for this project that we set amongst ourselves was to clean the data and store it in a way that is readable. We chose to extract and store the data that we received in a JSON structure. The reason behind this is that JSON is easily able to store the tweets in a readable format. The JSON files that we produced, after running our script, contain the collection type (whether it's a tweet, a retweet, or a quoted tweet), the tweet text, and the tweet ID [1]. These three fields were stored for every tweet. This was a crucial first step in this project because it allowed us to keep track of which tweet description belongs to the tweet ID, and whether that certain tweet was a retweet or not. Then from the given tweet ID, we created another similar JSON object file based on the hydrated tweets from the IDs that we initially collected.

3.2 Teach the Machine Learning Model

To be successful in having the public trust an information classifier like the one we were trying to build, the model needed to be very accurate. We believe that an accuracy score of 90% is more than enough for the public to have confidence in our model. Of course, to achieve an accuracy score this high, we needed to provide sufficient data to train on and test on. We provided the model with 6,000 tweets. The tweets that we provided to the model were tweets that we had gathered after hydrating. We then classified every single tweet into different buckets/categories that we had come up with before and confirmed with our client.

3.3 User Interface

The user interface contains two main sections, the statistics, and the timeline. The statistics section includes general information about the raw data as well as the cleaned data, and the hydrated tweets. The second section includes several timelines that shed light on when fake news tweets were published and filters based on the classifier. We also include an area on the statistics where we showcase users that spread more misinformation than others. Additionally, the front-end is accessible for all users, ranging from researchers to students.

4.0 Design

4.1 Twitter V1 JSON Format

Our raw data was originally inside a .gz file. Once the file was uncompressed our team read through it with Python. Each line of our file contains one JSON tweet object with the attributes listed in Table 1. The raw data attributes that we were particularly interested in are the lang, id, is_quote_status, quote_status, and retweeted_status.

Since our project was centered around some form of Natural Language Processing (NLP) we needed to make sure that the tweets we labeled were in English. The identifier of the tweet needed to be examined so we could determine if the tweet was still up on Twitter or if it had been removed. The is_quote_status determines if the tweet quotes another tweet, and if it does this attribute is set to “True. If is_quote_status is set to “True’ then the quote_status will contain another JSON tweet object. The retweeted attribute is deprecated and cannot be used to determine if the tweet retweeted another tweet because it is always false [2]. However, we can check if retweeted_status exists within the tweet [1]. The reason behind examining the quote_status and retweeted_status objects is because we want to examine those tweets as well and extract the ID from the raw data.

JSON Attribute	Type	Description
created_at	String	UTC time when this tweet was written
id	int	Unique identifier for this tweet, used in hydration process
id_str	String	The string representation of the ID attribute
text	String	The text (in UTF-8) of tweet
in_reply_to_status_id	String	If the tweet object is a reply, then this field is filled with the ID of the tweet it's replying to.
quoted_status_id	int	This field is filled with the tweet ID of the tweet the current tweet is quoting
is_quote_status	Boolean	Indicates if the tweet is a quoted tweet
quote_status	Tweet	This contains the tweet object of the original tweet.
retweeted	Boolean	Indicates if the tweet is retweeted; always false because the attribute is deprecated.
retweeted_status	Tweet	This contains the tweet object of the original tweet.
user	User	This user object contains an id, name,

	Object	screen_name, location, descriptions, and more.
entities	Entities	Any hashtags, URLs, or user mentions will be listed here.
possibly_sensitive	Boolean	This is only true when the tweet contains a URL in the tweet.
lang	String	Indicates the language the tweet was written in.

Table 1: Individual tweet JSON structure for Twitter v1

4.2 Hydration

Once the tweet IDs were stored, we hydrated them using an API called Twarc. Twarc uses Twitter API credentials to provide access to different functionalities like searching, filtering, hydrating, dehydrating, etc. Once our team received academic research access to the Twitter API, we were able to hydrate tweets using Twarc. Twarc is fed a list of tweet IDs and in return gives back a JSON object of tweets [3].

One issue with this whole process is that between the time the tweets were gathered (April 2020) and the time the tweets were hydrated (March 2022) Twitter changed the formatting of the API JSON object. So, we needed to educate ourselves on the differences between the Twitter v1 and Twitter v2 APIs. Table 2 shows the attributes of the Twitter v2 API, and a description of each attribute.

Attribute	Type	Description
id	String	Essentially identical to id_str in v1
text	String	The UTF-8 encoded tweet
attachments	Object	Specifies the types of attachments in the tweet
author_id	String	Unique identifier of the user that posted this tweet
created_at	Date (ISO 8601)	Date and time the tweet was made
entities	object	Entities that have been excluded from the text attribute that are included in the original tweet.
lang	String	Language the tweet was written in
possibly_sensitive	Boolean	This field is true when a link is contained
data	Object	The main object of Twitter v2, it encapsulates all

		above attributes.
includes	Object	This object is inserted if any users, polls, media, or place fields need to be included in the response.
users	Object	Array of user objects that contain attributes such as created_at, id, username, verified, name, etc.
media	Object	Array of media objects that contain attributes such as height, duration_ms, media_key, type, public_metrics, etc.
Tweets	Object	Array of tweets can be in both the includes object and the data object, and contains typical tweet attributes such as id, created_at, text, author_id, etc.

Table 2: Individual tweet JSON structure for Twitter v2 API

From the Twitter v2 JSON object, we needed to only extract a few attributes to label the data. We extracted the text from the data JSON object of the first 6,000 tweets and stored them in files for each team member to label (1,000 tweets each). Additionally, we needed the user_name attribute when we were examining which tweets contained misinformation, so we could find common sources of bad facts [4]. We could use this information to share with the user to steer them clear of users that commonly don't share truth.

4.3 Label Structure

Tweet	**Label**
It's irresponsible to write "Coronavirus is killing black people" without explaining why. And we know why: Poverty, medical redlining, doctor bias, profiteering... It's like reporting: "For some reason, Black people keep dying when the cops show up" YOU. HAVE. TO.	General misinformation

SAY. WHY.	

Table 3: Individual tweet bucket labeling

Table 3 demonstrates the format that we used to label the hydrated tweets. The First column indicates what the tweet is and the second and final column is which category we placed the tweet. The first column was filled using the Panda library in Python and stored into a CSV file, one for each member. Then each team member labeled each tweet using the second column in their respective CSV file. 1,000 tweets were stored per .csv file.

Figure 1: Label Buckets

As seen in Figure 1, these are the seven designated labels that we came up with before, to sort the tweets into. Each team member made a best effort labeling each tweet. However, uncertainty and inconsistency existed in some cases, which could lead to problems with our subsequent work on classification.

4.4 JSON Layout

In this project, we devised several scripts to obtain the data required. The first script was used to extract the tweets' IDs from the collected tweets for tweet hydration.

```
{
  "quote_count": 0,
  "quoted_status_permalink": {
    "url": "https://t.co/Z75GkGKpyS",
    "expanded": "https://twitter.com/commonstreasury/status/1247895764304449538",
    "display": "twitter.com/commonstreasur…"
  },
  "contributors": null,
  "truncated": false,
  "text": "@bbclaurak @Peston",
  "is_quote_status": true,
  "in_reply_to_status_id": null,
  "reply_count": 0,
  "id": 1247912781648625700,
  "favorite_count": 0,
  "entities": {…
  },
  "quoted_status_id": 1247895764304449500,
  "retweeted": false,
  "coordinates": null,
  "timestamp_ms": "1586360566900",
  "quoted_status": {…
  },
  "source": "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for iPhone</a>",
  "in_reply_to_screen_name": "bbclaurak",
  "id_str": "1247912781648625665",
  "retweet_count": 0,
  "in_reply_to_user_id": 61183568,
  "favorited": false,
  "user": {…
  },
  "geo": null,
  "in_reply_to_user_id_str": "61183568",
  "lang": "und",
  "created_at": "Wed Apr 08 15:42:46 +0000 2020",
  "quoted_status_id_str": "1247895764304449538",
  "filter_level": "low",
  "in_reply_to_status_id_str": null,
  "place": null
}
```


Figure 2: Raw Tweet JSON Layout (Twitter v1)

Figure 2 is an illustration of a JSON file of raw tweets. From there, we extracted the tweet ID and stored it in a file, to be hydrated using Twarc. The hydrated tweets were returned in Twitter v2 format (see Figure 3) which differs from Twitter v1 format (recall Figure 2).

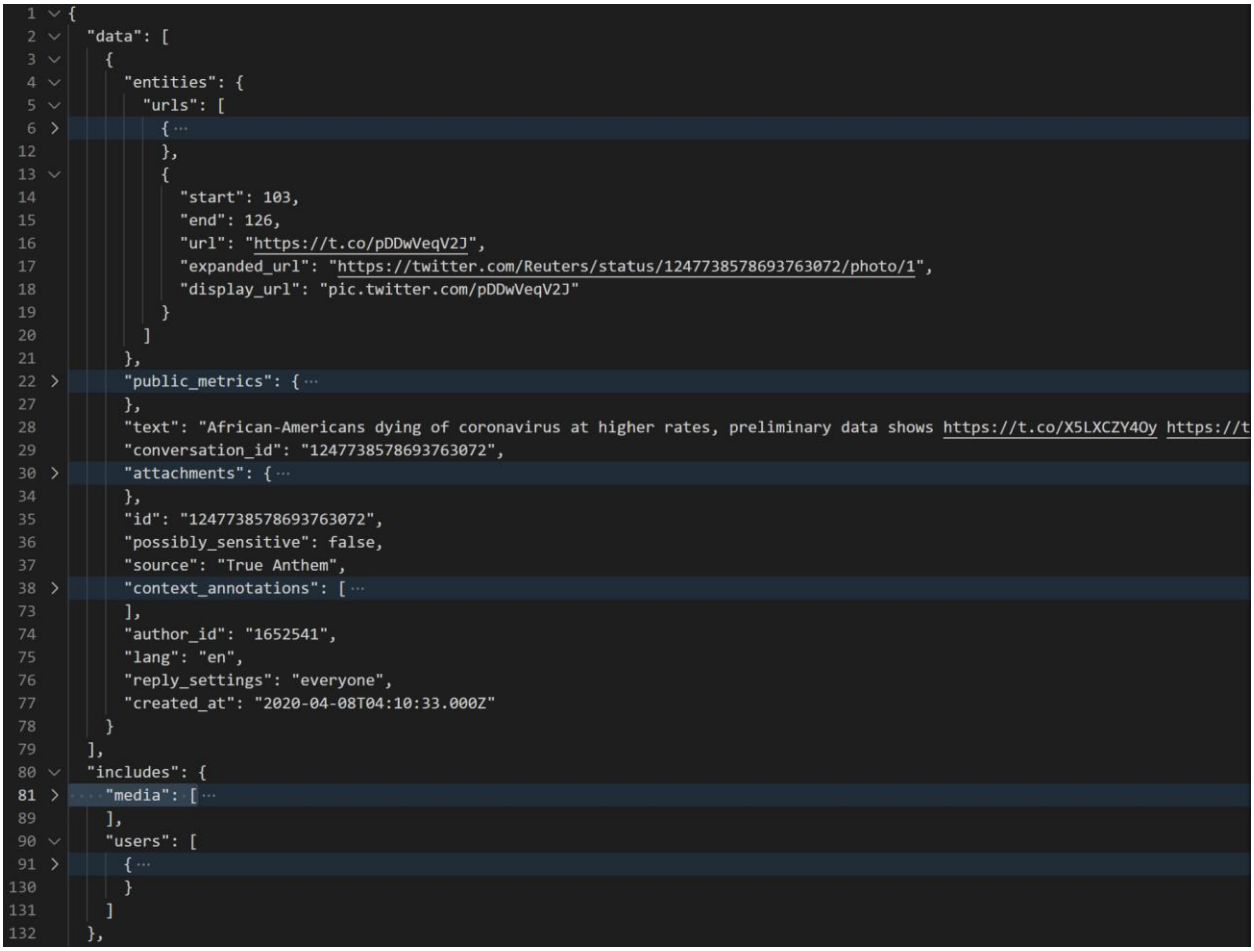

Figure 3: Twarc Tweet JSON Layout (Twitter v2)

4.5 Script Design

The chosen language for this project was Python. We could have chosen any language to make these scripts, but Python was chosen because of its flexibility in script design and since all our group members were comfortable with using it.

We chose to make several Python scripts. The first script was to run over the large initial file given to us by our client and remove duplicate tweets; a screenshot of the script is shown in Figure 7. The second script extracts the tweet IDs and stores them into a separate file [5]. The third script hydrates these tweets. The fourth script stores these hydrated tweets into different CSV files for labeling. The fifth script, which was given to us by our client, was the machine learning model that we needed to input our labeled tweets into. As shown in Figure 4, our plan was to clean, train and then evaluate our machine learning model and then continue to iterate through this pattern until we were satisfied with the accuracy of our algorithm.

Figure 4: Training Model

5.0 Implementation

Our implementation for this project included the use of several software components. There is one converter for individual tweets. The converter takes the form of a Python script that accepts an input data file and outputs a set of converted JSON tweets.

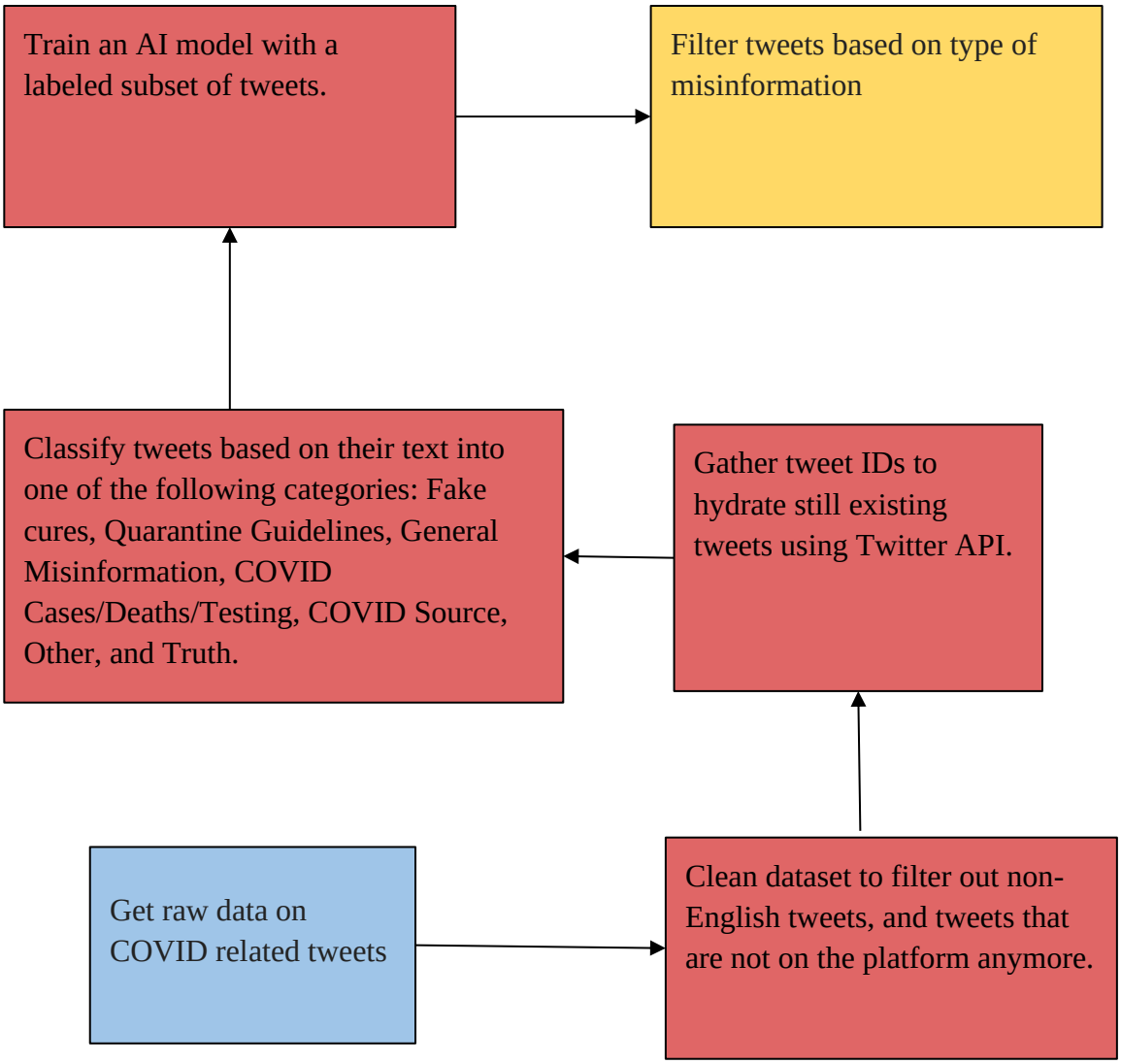

Figure 5: Methodology Pipeline

As seen in Figure 5, the first crucial step in our project was that we convert the raw data into a JSON file. Additionally, we wrote the converter scripts to consistently output their JSON data with a single JSON object per line. There are several advantages to this approach:

- Data can easily be obtained and cleaned from the desired object.
- Data is formatted in a professional fashion to present to our client.

5.1 Implementation Environment

This project was completed on a server provided by Dr. Fox and Dr. Mohamed Farag. The server operating system is Centos 7 (Linux) and the server has 2 TB of disk space with 8 cores, and 32 gigabytes of memory. The server also contains a GPU which we leveraged when training and testing the AI model. Before running any of the scripts we installed Python 3.6.8 and installed the following libraries: scikit-learn, pandas, JSON, warnings, Gensim, and io. These libraries were used one or more times to prepare the data or train the AI script.

5.2 Script Implementation

5.2.1 Cleaning Tweets

The cleaning tweets process consisted of processing the raw, collected data, and breaking it into JSON objects. Our first script took care of this by removing duplicated tweets and tweets that were not in English. Figure 6 gives the Python script that we used to achieve this desired goal of obtaining clean, readable tweets from the raw collected data. Since the raw data file was so large, we decided to read it by line using Python. We also wanted to collect all tweets – including retweets and quoted tweets – in English, this is shown in Figure 6. After running the script, the result was output into a JSON file [5].

```
import json
import io
import ijson
from pandas.io.json import json_normalize
import warnings
file_name = 'covid.jsonl'
warnings.filterwarnings("ignore")
# clean_tweets = open("clean_tweets.txt", "w")
num_lines = 0
with io.open(file_name, 'r', encoding='utf-8') as f:
    for line in f:
        tweet = json.loads(line)
        if tweet['lang'] != 'en':
            continue
        #Grab the quoted tweet id, and original id
        # if tweet['is_quote_status'] == 'true':
        #     continue
        #grab original tweet id
        # if tweet['retweeted'] == 'true':
        #     continue
        # id = str(tweet['id'])
        # clean_tweets.writelines(id + "\n")
        num_lines = num_lines + 1
print(num_lines)
# clean_tweets.close()
```

Figure 6: Cleaning Script

5.2.2 Removing Duplicates

We did not want to teach the AI the same thing repeatedly, so removing duplicated tweets was a crucial step. For this script, we utilized a dictionary in Python and added the tweet IDs to it. Each tweet has their own unique tweet ID, so for every single ID that was extracted and cleaned from the original raw file, we would check to see if the tweet ID was already in the dictionary. If not, it was added to the dictionary, and if it was, then it was skipped.

```
filename = "all_tweets.txt"
new_file = "all_tweets_no_duplicates.txt"
id_dictionary = {}
with open(filename, "r") as old_file:
    for line in old_file:
        if line not in id_dictionary:
            id_dictionary[line] = 1
with open(new_file, "w") as new_f:
    for key in id_dictionary:
        new_f.write(key)
```

Figure 7: Removing Duplicates Script

5.2.3 Hydrating Tweets

Tweet hydrating is the process of filling an object with data – in this case, JSON objects with tweet data. Tweet hydrating generally involves four steps. The first step involved finding a data set to work on. That led to the tweet IDs extracted from what we cleaned and collected from the client's raw COVID tweet data. The second step was to make a Twitter account and register it under their developer program. This was important because Twitter is very protective about who has access to their data. To get a developer account, one must visit Twitter's developer website and apply. With the consumer (API) key, consumer (API) secret, access token, access token secret, and bearer token (optional), we were able to hydrate the tweet IDs, using the Twarc2 library [3]. The general layout for setting up Twarc is:

from twarc2 import Twarc
t_inst = Twarc(consumer_key, consumer_secret, access_token, access_token_secret)

BEARER_TOKEN=BEARER_TOKEN
CONSUMER_KEY=CONSUMER_KEY
CONSUMER_SECRET=CONSUMER_SECRET
ACCESS_TOKEN=ACCESS_TOKEN
ACCESS_TOKEN_SECRET=ACCESS_TOKEN_SECRET

5.2.4 Word Frequency Analysis

After all the hydrated tweets were gathered in a file, we decided it would be beneficial to get an idea of frequently used words used in the tweets so that we could produce a classification scheme that would reflect what's inside the data. To do this, we needed to be able to store and look up words quickly since this database of hydrated tweets has tens of thousands of unique

words. We decided to use a dictionary in Python because it offers O(1) lookup and insertion time. The idea was simple: parse each tweet word by word and if the word was already in the dictionary, then use the word as the key and add one to the value. If the word didn't exist in the dictionary, then add the key value pair (word, 1) to the dictionary. Once this was done for all tweets, we sorted the dictionary by value and printed out the most common words. We omitted common words such as "the" and "and" because these words hold no relevant value. This was possible by using a stop word list.

When we ran this script, we unfortunately couldn't find value behind the words that were commonly used. Figure 8 shows the most frequently used words we found from running the word_frequency.py script. As you can see, there's not a lot of value that we can extract from this list of words. So, we decided to produce a list of classes, which were agreed upon by our client, independent of this script. The seven tweet classes that, we thought, cover the quarantine COVID timeline are: COVID Source, Fake Cure, Quarantine Guidelines, General Misinformation, COVID Cases/Deaths/Testing, Truth, and Other.

Figure 8: Most common words in hydrated data

5.2.5 Making Training Sets

Our goal as a team for training the machine learning script was to label 6,000 tweets. To make that fair, everyone on our 6-person team was assigned 1,000 tweets to examine and label. To do this, we used a simple modulo operator to divide up the first 6,000 tweets in the dataset to train. Since the tweets were in no order, selecting the first 6,000 tweets was random. We created dataframes for each person in the group and added a frame with just the tweet and an empty 'label' column that they were to later fill in. Once there were 1000 frames in each dataframe we exported the dataframe into a CSV file then downloaded and shared on Google Drive so everyone could label their tweets. Each CSV file is 1000x2 in dimension; see Figure 9.

5	In case the scientific 'experts' haven't realised it yet, the establishment, rightly or wrongly are about to throw them under the bus ! Special Report: Johnson listened to his scientists about coronavirus - but they were slow to sound the alarm https://t.co/iPZeUhMVbV	General misinformation
6	Media briefing on #COVID19 with @DrTedros. #coronavirus https://t.co/BOe2WSGfnK	Truth
7	55% of Americans are smart enough to see that Trump has done a poor job of preventing the spread of the coronavirus in the United States. The other 45% of Americans think the former host of Celebrity Apprentice is doing a great job because they are total fucking morons.	General misinformation
8	Should you invest during coronavirus pandemic? https://t.co/wJdewyVSFE @MorningsMaria @FoxBusiness	General misinformation
9	ICE is trying to hide the coronavirus outbreaks raging through detention centers. Another thing to watch for? ICE may try to hide deaths by "releasing" people in comas and on ventilators at a hospital. Once "released," the death is no longer "in custody." They've done it before. https://t.co/yjSJU5UtGp	General misinformation
10	Pope Francis has said the coronavirus pandemic is one of "nature's responses" to humans ignoring the current ecological crisis. https://t.co/mgrDyNhIEX	General misinformation
11	HUGE: Brilliant Dr. Shiva, Inventor of Email, Outlines Connections Between Bill Gates, Dr. Fauci, the WHO and the CDC – Relevant to Coronavirus Pandemic https://t.co/dhZEIm6spu https://t.co/A0pAutRrTT	General misinformation
12	UK rejects Trump offer to help Boris Johnson's coronavirus treatment - "We're confident the prime minister is receiving the best possible care from the National Health Service," "Any treatment he receives is a matter for his doctors." https://t.co/Lj7CGgImbR	truth

Figure 9: Screenshot of Google Sheets with labeled tweets

5.2.6 Making Training/Testing Folders

Once someone in the group finished the labels, the make_ai_data_folders.py script was used to convert the CSV file into something compatible with the machine learning program our client gave to us. To make the CSV file compatible, each tweet needed to be put into its own separate file and placed into a folder with tweets that share that same label. So, 'general misinformation' tweets are inserted into their own folder, while 'truth' tweets are inserted into their respective folder, and so on. However, these folders could not be given their conventional names, they needed to be numbers. So, we assigned a number to each classifier as follows: (Quarantine Guidelines: 0), (Fake Cures: 1), (COVID Source: 2), (COVID Cases/Deaths/Testing: 3), (General Misinformation: 4), (Other: 5), (Truth: 6).

Each CSV file that was ready was converted into text files by using the make_ai_data_folders.py script and passing in the CSV file as an argument. The script can take an unlimited number of CSV files and convert each tweet in them into a text file. Each text file is given a simple name; the naming convention for each tweet file is: "integer".txt. For example, the first tweet's file name would be "0.txt" and the second would be "1.txt", and so on. If the program doesn't recognize a label because of a misspelling it will print out the label and the row number of the tweet occurrence in the CSV file. This helped find and correct spelling mistakes that team members made while labeling.

5.2.7 Training and Testing the AI Model

Our machine learning script is called ‘text_classification.py’ and uses the sklearn library from Python [6]. The first objective of our client’s script is to read in the folders of tweets, which can be done by passing the name of the folder that holds the classifier folders in as an argument. Once the argument is read, the files are loaded into the program and split into training and testing data. We allocated 80% of the files to training, and 20% of them to testing. Next, some pipelining is done using SVM and TFIDF (term frequency-inverse document frequency). What this pipelining does is create a matrix, including every word used in the whole training set, which is used in the training aspect of the script.

SVM (Support Vector Machines) aimed to find a separating line between 2 or more classes of data, and since we had 7 classes, this type of machine learning model was perfect. SVMs work by finding points closest to a line (the entity that distinguishes 2 classes); these are called support vectors. The distance between the line and support vectors is called a margin and the hyperplane that maximizes the margin is said to be the optimal hyperplane. This is expected because if the points closest to the line are far away relative to other hyperplanes, then these points are easier to distinguish between classes and there is higher confidence in classification [7].

We excluded some of the most common words by using a stop word list using Gensim. Such words have little value toward classification [8]. We also used stemming to remove prefixes and suffixes from words [9]. Once the matrix is made, a classifier model is built using the training data, and an F1 score is calculated.

Once the model was trained, it was saved as a .joblib file which can then be used to label the rest of the hydrated tweets. After, we tested the model and checked for accuracy and recall, which are two important criteria in evaluating the performance of a machine learning model. Evaluation results are shown in Figure 13.

Originally, we had used all labeled tweets to train the machine learning model. However, because of an uneven distribution of tweet labels this made our model less accurate for certain classifiers. The uneven distribution of tweets can be seen in Figure 10. The performance metrics for the uneven tweet labels is highlighted in Figure 11. To combat this problem, we revised the make_ai_data_folders.py script to find the fake news category with the smallest number of tweets. From there, we limited the number of tweets in the other categories as well so that it would be more evenly distributed. In our case, the ‘Covid Source’ classifier had the lowest number of labels at 250. Figure 12 shows how the tweet categories were distributed. Comparing the F1 scores from Figure 11 and 13, we see that distributing the tweets more evenly positively affected the performance of the classification model.

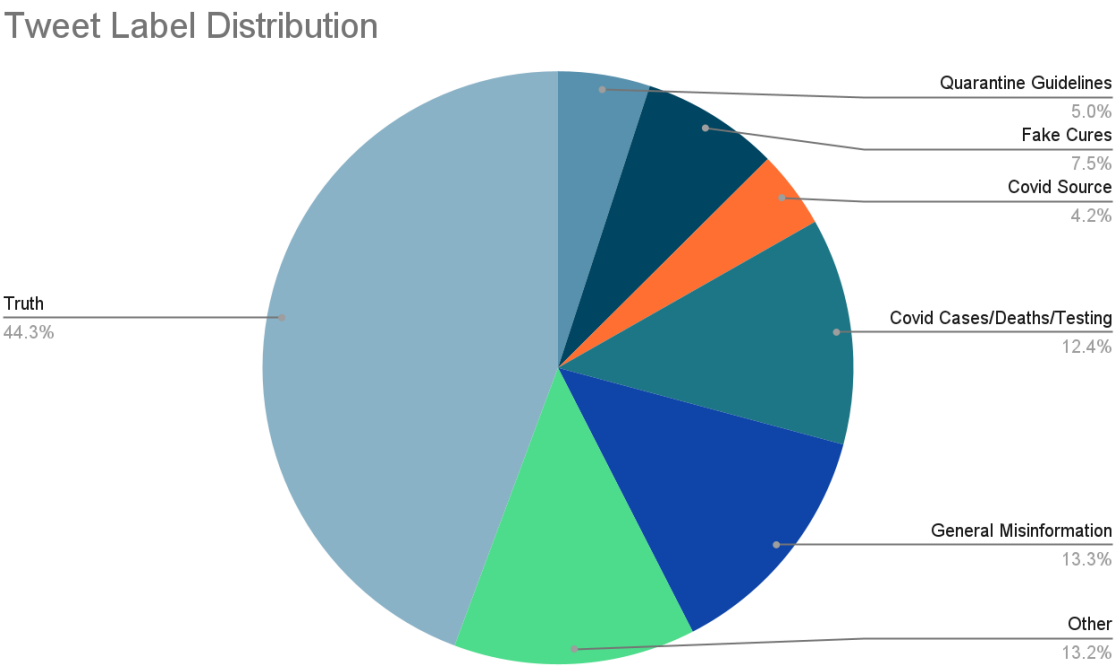

Figure 10: Uneven Distribution of Tweet Labels

```
Model f1=0.429
Accuracy=0.420

              precision    recall  f1-score   support

           0       0.40      0.60      0.48        57
           1       0.58      0.80      0.67        83
           2       0.11      0.25      0.16        51
           3       0.59      0.64      0.61       150
           4       0.26      0.43      0.33       137
           5       0.24      0.29      0.27       153
           6       0.62      0.33      0.43       550

    accuracy                           0.42      1181
   macro avg       0.40      0.48      0.42      1181
weighted avg       0.49      0.42      0.43      1181

confusion matrix
[[ 34   0   7   2   4   4   6]
 [  0  66   3   0   6   1   7]
 [  2   0  13   2  13   4  17]
 [  7   1  14  96  14   6  12]
 [  4   5   8   8  59  25  28]
 [  3   2   9   4  49  45  41]
 [ 36  40  60  51  80 100 183]]
```

Figure 11: Performance Results for Unevenly Distributed Tweets

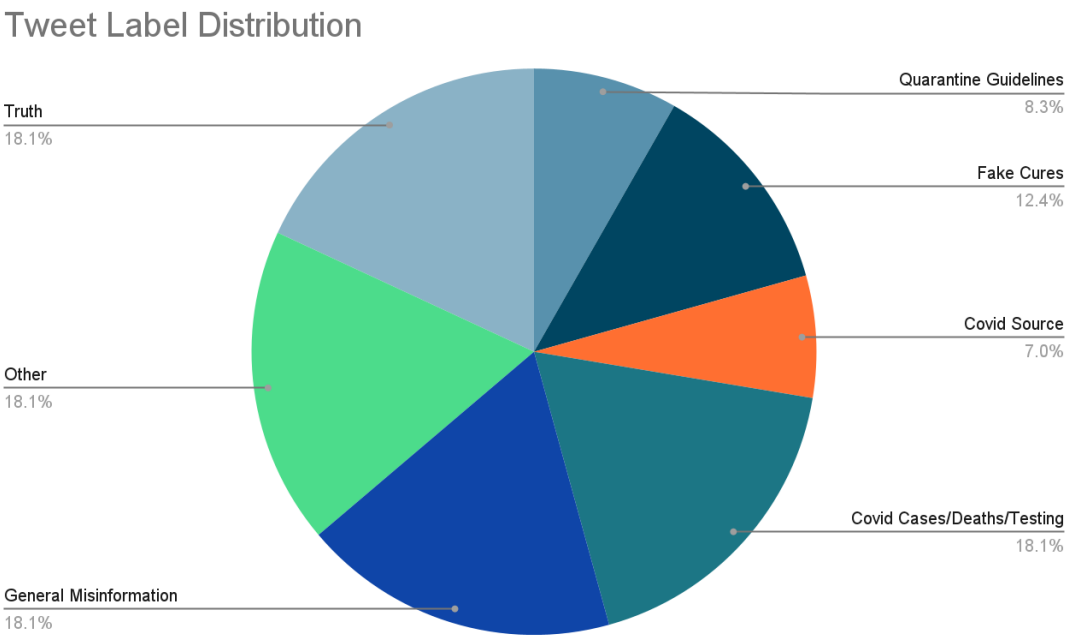

Figure 12: Even Distribution of Tweet Labels

```
Model f1=0.587
Accuracy=0.580


              precision    recall  f1-score   support

           0       0.42      0.60      0.50        48
           1       0.83      0.84      0.83       104
           2       0.30      0.43      0.36        49
           3       0.78      0.70      0.74       126
           4       0.44      0.41      0.43       123
           5       0.45      0.53      0.49       129
           6       0.76      0.51      0.61       140

    accuracy                           0.58       719
   macro avg       0.57      0.58      0.56       719
weighted avg       0.61      0.58      0.59       719

confusion matrix
[[29  0  6  0  4  8  1]
 [ 1 87  1  1  6  2  6]
 [ 5  0 21  5  9  8  1]
 [ 6  1  7 88  7 14  3]
 [ 6  8 11  9 51 30  8]
 [13  0 15  4 24 69  4]
 [ 9  9  8  6 15 21 72]]
```

Figure 13: Performance Results for Evenly Distributed Tweets

Our main criteria to validate the accuracy of our fake news detector was the F1 score and the accuracy score, as instructed by our client Dr. Farag. As our team didn't have much background knowledge on machine learning or artificial intelligence, we followed Dr. Farag's advice. An F1 score is calculated using the precision and recall of the model and can be found using: $f1 = 2 * (\frac{precision * recall}{precision + recall})$ where $precision = \frac{\#\ of\ True\ Positives}{\#\ of\ True\ Positives + \#\ of\ False\ Positives}$ and $recall = \frac{\#\ of\ True\ Positives}{\#\ of\ True\ Positives + \#\ of\ False\ negatives}$ [10].

As shown in Figure 13, our F1 score is lower than our target mentioned in Section 3.2. However, we still received some good tweet predictions. Of course, there were some poor predictions as well; examples of predictions can be seen in Table 4.

Tweet	Classifier
African-Americans dying of coronavirus at higher rates, preliminary data shows	COVID Source
RT @FloydShivambu: The socioeconomic conditions, access to healthcare, and the inter generational poverty of black people in America places…	Truth
It's irresponsible to write "Coronavirus is killing black people" without explaining why. And we know why: Pover…	COVID Cases/Deaths/Testing
Clear and forensic account from Reuters of how #COVID19 policy in the UK evolved from January to March, and how the…	Quarantine Guidelines
RT @kieran_walshe: Clear and forensic account from Reuters of how #COVID19 policy in the UK evolved from January to March, and how the scie…	General Misinformation
Mayor of Greater Manchester Andy Burnham says 4 in 10 cars on Gtr Mcr's roads are now exceeding speed limits. Befor…	Quarantine Guidelines
Bernie Sanders' announcement comes after weeks of clinging to an all-but-impossible	General Misinformation

path to victory over his modera…	
Hydroxychloroquine is a drug needed to treat hundreds of thousands of Americans with Lupus and Rheumatoid Arthritis…	Fake Cures
Thinking of seeing friends and family this Easter? Don't. If you leave home, you could catch or spread…	Quarantine Guidelines
RT @GOVUK: Thinking of seeing friends and family this Easter? Don't. If you leave home, you could catch or spread #coronavirus. #StayHom…	Quarantine Guidelines

Table 4: Tweet Classifier Examples

Also shown in Figure 13, we calculated a confusion matrix, which is another way to gauge the performance of our classification algorithm. To calculate a confusion matrix, we needed to know the total number of correct predictions, and the total number of incorrect predictions, organized by the class that was predicted [11]. For example, the number 0 refers to the class 'Quarantine Guidelines' so row zero corresponds to 'Quarantine Guidelines' in the confusion matrix. Each entry in row zero represents the number of times a tweet was classified in the corresponding column. So, entry [0,0] in the confusion matrix represents the total number of times a tweet classified as 'Quarantine Guidelines' was predicted to be a tweet regarding 'Quarantine Guidelines'. To continue, entry [0, 1] represents the number of times a tweet classified as 'Quarantine Guidelines' was predicted incorrectly as 'Fake Cure'.

Upon closer inspection of the confusion matrix in Figure 13, it seems that entries at columns 4 through 6 and rows 4 through 6 are high compared to other entries in the matrix. This indicates that there are tweets that are mis-predicted between the 'General Misinformation', 'Other', and 'Truth' classifiers. The causes for this could be any number of reasons. One such reason could be that these classes are too general, relative to the others. There also could be problems related to selection and weights of keywords. For instance, the 'General Misinformation' classifier could be looking for keywords such as 'mask' and 'social distancing' that also were weighted highly by the classifiers for 'Other' and 'Truth', so it's more difficult to distinguish between them.

5.2.8 Front-end

The main goal of our front-end was to display both a timeline of tweets that the classification model had labeled, and statistics on tweets/users. To achieve this, we created an Angular project that can read from the JSON files that the backend produces. At first, when the

team was trying to access the information in the JSON files, we did not realize that we could not simply iterate through the file in a for loop. For example, if we wanted to acquire the most used word in a specific category, the JSON had the word stored as the key, and the number of times it showed up as the value. This meant that if we were to access the JSON file, we would need to know the word specifically to access these numbers. To work around this, we decided to order the JSON values with ranks, going from 1 to x, and the values being both that common word as well as the number of times it showed up separated by a comma. This allowed us to access the JSON with a simple for loop, grabbing the most important keys at the front, and using string manipulation to separate both the word and the number to be displayed.

This project used two components to display our different pages: the timeline component and the statistics component. The timeline component shows a vertical scrollable display of how many tweets were flagged each calendar day, with different buttons to switch between each of our different classifiers. Figure 14 in Section 7.4 shows the timeline, the buttons to navigate between each classifier, and the navigation buttons at the top to switch between the timeline and statistics pages.

The statistics tab shows general statistics about each tweet category. The statistics shown are separated by each category, and all have the most common word, average tweets flagged per day, most common Twitter account, and an example tweet from the respective bucket. All this information is pulled from separate JSON files and updates the HTML document as each button is pressed. Finally, the buttons for each classifer can change what information is displayed, like the timeline page. Figures 15 and 16 in Section 7.5 are two screenshots of the All Tweets classifier and the Quarantine Guidelines classifier, respectively.

6.0 Testing

We implemented several scripts to ensure we generate the desired output for each specific test. The first test was to make sure that no tweet IDs were repeated. The second test ensured that the Twarc library hydrate function worked as expected. The third test that we deployed was to test user's satisfaction on our interface. Finally, the fourth test was used to learn about the Twitter v2 API format, since there was very little information available online.

6.1 Tweet ID Test Implementation

The first script was quite simple. Since we had outputted all our tweet IDs to another file, after cleaning, all we had to do was make a Python program where we open the input file and check the number of times that tweet ID appears. If that number was greater than one, we ignore that specific tweet ID from the program. If it is one, then we output that tweet ID to a final JSON file – storing each tweet ID per line.

6.2 Hydration Test

Since no one on our team had experience with the Twarc library, we thought it would be beneficial to test it out. Specifically, we wanted to make sure that the hydration command worked correctly. To test this, we had a small text file of 10 tweet IDs we published ourselves. Five of the IDs were of tweets we knew were online and five were tweets that we deleted. We ran the hydration command and found that all the online tweets were returned while the deleted tweets failed to hydrate. From this test, we knew that there weren't any issues with the Twarc library.

6.3 Users' Satisfaction Test

For testing the UI, we asked several people to use it and give us feedback after a certain period. The purpose of this was so we could receive the general sentiment from users while using our interface. Furthermore, we were able to focus on observing bugs that we missed while coding. This could range from buttons that would not work when clicked, to a timeline that fails to display, and more.

This testing was done by letting about 15 different people in different majors test out our front-end. These majors spanned ECE to BIT to Art majors, and all had varying knowledge of computers and technology. The questionnaire had different questions to be rated by a score of 1-5, with the questions being things such as ease of accessibility, visual appeal, how easy it was to understand, and so on. At the end, the form asked for additional comments, both positive and negative, and then a final question about any bugs that may have occurred; fortunately, none were found.

Many of these users liked how easy it was to understand what the program was about and how everything was laid out well for easy access. However, most said that our visual appeal department was our weakest, as our front-end looked very bare, with minimal colors. We learned that we needed to change things on both the statistics as well as the timeline pages. Many people suggested that the timeline was too long, and that people had to scroll too much for the number of tweets we were displaying. We fixed this by changing the timeline to display the number of

tweets per day rather than each individual tweet, and had the statistics page show individual example tweets for each classifier. Though the timeline was still relatively long, our client wanted the information we were displaying at the minimum, and we could not cut the timeline page to be any shorter. Table 5 highlights the questions and average responses from the survey.

Questions	Average Response
How easy is the application to maneuver?	4.6
How visually appealing is the application?	2.2
How easy is it to understand the application?	4.8
How easy is it to find the specific information you are looking for?	3.7

Table 5: User Satisfaction Question Results

6.4 Manual Inspection Test

There was another method we used to test our expected output data. Unfortunately, we were unable to find much information on the Twitter v2 API JSON format. To help with comprehension, we needed to read one tweet object at a time. By hydrating individual tweets, we could determine easily if we extracted the correct attributes from the tweet object. We completed this test with different types of tweets to observe how the structure of the JSON object changed. We tested 5 original tweets, 5 retweeted tweets, and 5 quoted tweets to compare the structure in each.

7.0 User's Manual

7.1 Use Environment Discussion

The command lines that are specified in the requirements use the Linux/GitBash versions of commands. If the user is on a different operating system, alter the command line commands to that operating system's equivalent of the commands.

7.2 Use Cases/Tasks Supported

Our codebase can support a variety of different tasks, but these can mostly be put into three categories: data conversion, data validation, and data utilization.

7.3 Web Application

When the web app is first opened, the user will be directed to the tweet timeline showing the number of tweets for each day. There is a toolbar at the top with the timeline and statistics buttons which will take the user to each respective section.

7.4 Timeline

On the timeline page, the user is presented with a title, a list of buttons, and a vertical timeline below all the buttons. The buttons change what is displayed on the timeline; the timeline itself shows a count of the total tweets flagged each day for possible misinformation. Figure 14 demonstrates the user interface for the timeline.

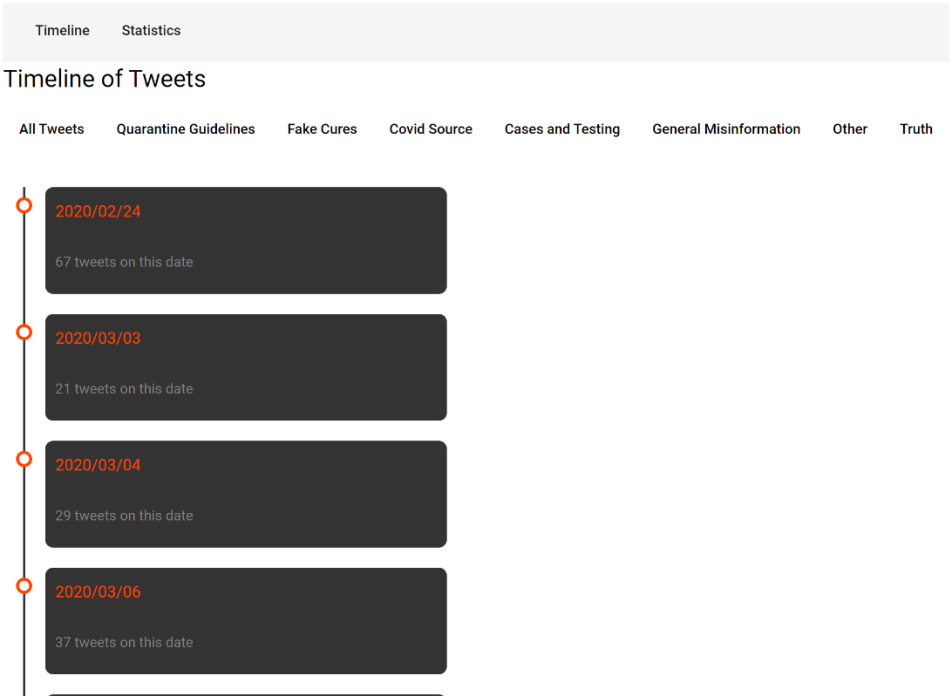

Figure 14: Timeline UI

7.5 Statistics

The statistics page is like the timeline page in that there is a title and then the buttons of all the buckets are listed. However, there is an extra button at the front that shows the statistics of all the tweets, the number of tweets that were true, number of tweets that are in each bucket, most popular false information tweeters, and the number of offending users posting to each bucket. The other category buttons will show the specific statistics of that category alone. Figure 15 shows off the statistics we found in the entire dataset. Users are also able to see the same statistics for a specific classifier, an example is shown in Figure 16.

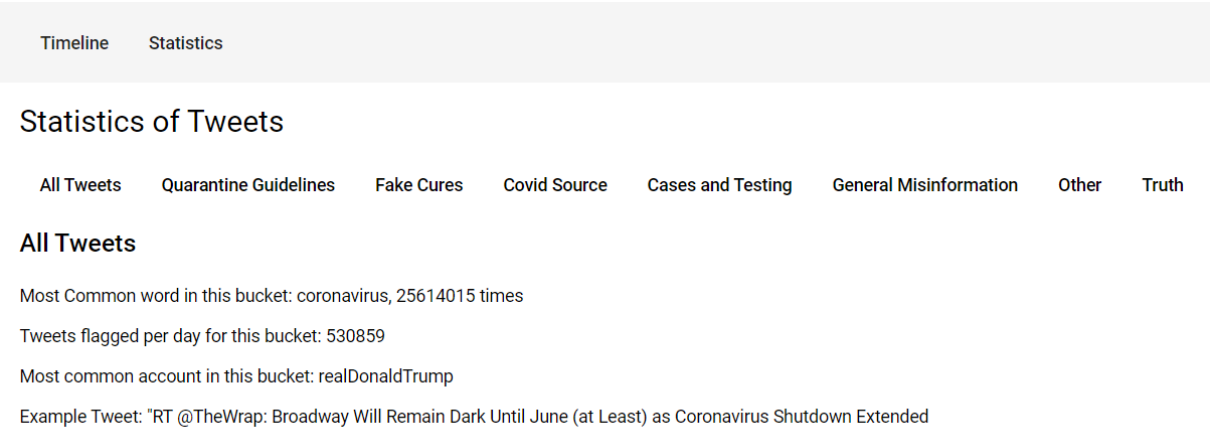

Figure 15: All Tweets Statistics UI

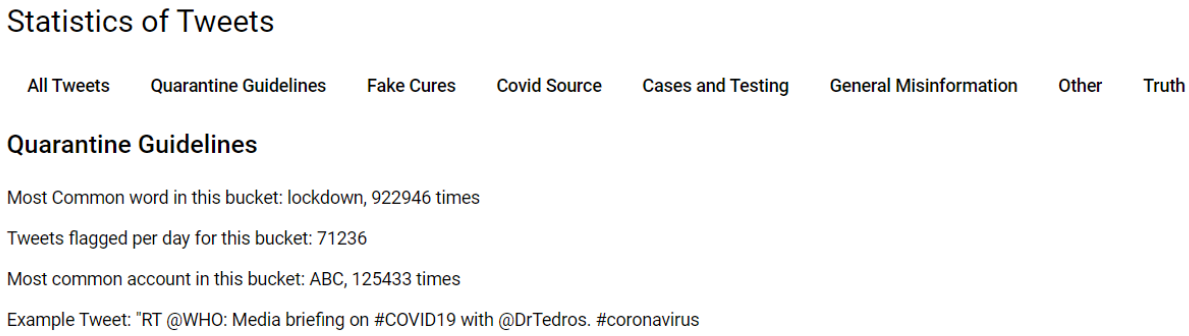

Figure 16: Quarantine Guidelines Statistics UI

8.0 Developer's Manual

8.1 Program Files

Program Name	Command	Description
clean_tweets1_1.py	python3 clean_tweets1_1.py <JSONl file> <output.txt>	Taking in two arguments, the JSONl file (in Twitter v1 format) is the input file clean_tweets1_1.py reads from and outputs the IDs in the output.txt file.
make_ai_data_folders.py	python3 make_ai_data_folders.py <file_1.csv> <file_2.csv> … <file_n.csv>	This program takes in n CSV files and reads them line by line. Each tweet in a line is placed in its own text file, in a folder corresponding to its label. Each label corresponds to a number 0-7.
make_csv_files_for_training.py	python3 make_csv_files_for_training.py	This program will look at the first 6,000 tweets and divide them up into 6 dataframes using the Pandas library. Then, we convert the dataframe into a CSV file with the PID[1] of each group member to specify.
text_classification.py	python3 text_classification.py <data_folder>	This program is the machine learning script that mainly leverages the scikit-learn Python library to do the machine learning. The program takes in a parameter which is the path to the directory that stores all the directory classifiers for the labeled tweets. From there, the tweets are divided into test and train, 80% and 20%, respectively. Then we use

[1] Refers to the unique alphanumeric username Virginia Tech designates for each student.

		SVM to machine learn and determine the accuracy of the model through the F1 and accuracy score seen in Figure 13.
word_frequency_analysis.py	python3 word_frequency_analysis .py	This program looked at the hydrated tweets and used a dictionary to count the total times a word had been used in all the hydrated tweets. Common English words were omitted using a stopword list from the Gensim Python library. From there, we sorted this dictionary by the number of occurrences a word made and printed out the top 100 most used words into a file called word_frequency_analysis.txt.

Table 6: Program Files

8.2 Data Files

Program Name	Command	Description
all_tweets.jsonl	N/A	File that stores all of the hydrated tweets.
all_tweets_no_duplicates.txt	N/A	New file after running our script that removes all of the duplicated tweet IDs.
all_tweets.txt	N/A	Original file that stores all of the tweet IDs of tweets that our client collected but stored as a text file.
pure_JSON_covid.jsonl	N/A	Raw data file given to us by our client
word_frequency_analysis.txt	N/A	Output after we run our frequency analysis script.

covid_zipped.gz	N/A	Original file that holds all the collected tweets data from our client.
training_folder	N/A	Folder that stores our 6000 labeled tweets in individual files.

Table 7: Data Files

8.3 Test Files

Program Name	Command	Description
remove_duplicate_ids.py	python3 clean_tweets1_1.py <JSONl file> <output.txt>	This program takes in a JSONL file, checks the IDs in the first line with all the other IDs. If there are any that match, ignore it, and output unique IDs to the output file.
hydrated_test.txt	twarc2 hydrate hydrated_test.txt > hydrated_test.jsonl	Text file containing tweet IDs we know are on Twitter and that aren't on Twitter. We expect hydrated_test.jsonl to have 5 tweets in the 'data' object and 5 in the 'errors' object showing that it couldn't locate 5 tweets.
manual_inspection_test.txt	twarc2 hydrate manual_inspection_test.txt > inspection.jsonl	Used to inspect the Twitter v2 API format, 5 retweets, original tweets, and quoted tweet IDs in this file.

Table 8: Test Files

8.4 Miscellaneous Files

Program Name	Command	Description
raw_data_fooling_around.py	N/A	Used for testing codes.
clean_tweets.py	python3 clean_tweets.py	Used for cleaning our raw data file

find_retweet.py	python3 find_retweet.py	Used this script to see if any tweet objects have the `retweeted" attribute set to 'True'. Found out here that the attribute is deprecated.
find_range_of_dates.py		Find_range_of_dates.py is used to find the range of dates of tweets that are collected. Originally our client, Dr. Farag said he collected tweets from April 2020 until March of 2021 however this program shows that tweets are only collected from April 2020 until the beginning of May 2020. Discussed in detail in Section 10.2.
model.joblib	N/A	This is the machine learning model that is trained using labeled tweets. It is then used later to label the rest of the raw dataset.
ericwiley10__training_data.csv	N/A	Eric Wiley's tweets that he labeled.
ferrinkirby__training_data.csv	N/A	Ferrin Kirby's tweets that he labeled.
tungngvyen__training_data.csv	N/A	Tung Nguyen's tweets that he labeled.
supplement_tweets.csv	N/A	Campbell Dalen's tweets that he labeled.
ktoroc__training_data.csv	N/A	Kyle Toroc's tweets that he labeled.
fareezaz__training_data.csv	N/A	Fareeza Zameer's tweets that she labeled.
word_frequency_analysis.txt	N/A	The 100 top most common words in the hydrated tweets. Used when considering different classifiers.

Table 9: Miscellaneous Files

The above tables are descriptions of all files relevant to this project. Table 6 describes all scripts used to clean, label, and train/test a machine learning model. Table 7 describes the files that contain the data we used. Table 8 is used to mention any testing we completed, and Table 9 mentions miscellaneous files.

8.5 Tutorials

8.5.1 Installing Python Libraries

The best and easiest way to install any of the Python libraries mentioned in Section 8.6 is to use the ‘pip’ command. ‘Pip’ stands for preferred installer program and is used widely today for adding Python libraries to a machine. To run it, simply type ‘pip install <library name>’ if you’re running on Python 1 or 2. If Python3 is running on your machine then use the command ‘pip3 install <library name>’ instead.

8.5.2 Working with Twarc

Like Pandas and JSON, Twarc is another Python library. However, it is also a command that can run on the command line. To do this, we first registered an API account on Twitter [12]. Once you’ve been approved for an account, you can run the command ‘twarc2 configure’. Once that is typed in, you will be asked to complete a bearer token; however, if you decline you can easily enter in your consumer_key, consumer_secret, API_key, and API_secret. Make sure you do not share these credentials with anyone [3].

8.5.3 Installing jq Command

Unlike other dependencies this is not mandatory. However, it is very helpful, and we wish we had discovered it earlier. The ‘jq’ command is useful for making JSON files much easier to read. In the raw data file, and the hydrated JSON files, each JSON tweet object could contain other JSON tweet objects, making readability very low. To add to the reading issue, the raw data file was very large, too big to open on any text editor. However, two commands, ‘head’ and ‘tail’ allowed us to see the beginning and end (respectively) of the raw file. Once we had a smaller chunk of raw data, we used the ‘jq’ command to increase readability. To install this command requires several steps. First, we need to install the epel repository with the command ‘yum install epel-release -y’. Since the jq command is part of the epel repository, installing it first is necessary. Next, we simply install the command calling: ‘yum install jq -y’ [13]. After that we have installed the jq command and we can call it by typing: ‘jq . <unformatted JSON file> > <formatted_output>. In Figure 17 and Figure 18, you can see the difference in readability between the unformatted JSON and the formatted JSON example.

```
{"quote_count": 0, "quoted_status_permalink": {"url": "https://t.co/Z75GkGKpyS"
, "expanded": "https://twitter.com/commonstreasury/status
/1247895764304449538", "display": "twitter.com/commonstreasur\u2026"},
"contributors": null, "truncated": false, "text": "@bbclaurak @Peston",
"is_quote_status": true, "in_reply_to_status_id": null, "reply_count": 0,
"id": 1247912781648625665, "favorite_count": 0, "entities": {"user_mentions":
[{"id": 61183568, "indices": [0, 10], "id_str": "61183568", "screen_name":
"bbclaurak", "name": "Laura Kuenssberg"}, {"id": 14157134, "indices": [11,
18], "id_str": "14157134", "screen_name": "Peston", "name": "Robert Peston"}]
, "symbols": [], "hashtags": [], "urls": []}, "quoted_status_id":
1247895764304449538, "retweeted": false, "coordinates": null, "timestamp_ms":
"1586360566900", "quoted_status": {"quote_count": 101, "contributors": null,
"truncated": true, "text": "1/3 JUST PUBLISHED: @MeljStride has written to
@RishiSunak urging him to set out what action he will take to help t\u2026
https://t.co/mk6IDqKQOt", "is_quote_status": false, "in_reply_to_status_id":
null, "reply_count": 86, "id": 1247895764304449538, "favorite_count": 174,
"entities": {"user_mentions": [{"id": 3183943337, "indices": [20, 31],
"id_str": "3183943337", "screen_name": "MeljStride", "name": "Mel Stride"},
{"id": 1168968080690749441, "indices": [47, 58], "id_str":
"1168968080690749441", "screen_name": "RishiSunak", "name": "rishisunak"}],
"symbols": [], "hashtags": [], "urls": [{"url": "https://t.co/mk6IDqKQOt",
"indices": [117, 140], "expanded_url": "https://twitter.com/i/web/status
/1247895764304449538", "display_url": "twitter.com/i/web/status/1\u2026"}]},
"retweeted": false, "coordinates": null, "source": "<a href=\"https://mobile
.twitter.com\" rel=\"nofollow\">Twitter Web App</a>",
"in_reply_to_screen_name": null, "id_str": "1247895764304449538",
"display_text_range": [0, 140], "retweet_count": 139, "in_reply_to_user_id":
null, "favorited": false, "user": {"follow_request_sent": null,
"profile_use_background_image": true, "default_profile_image": false, "id":
976055334, "default_profile": true, "verified": true,
```


Figure 17: Unformatted JSON Example

```
{
  "quote_count": 0,
  "quoted_status_permalink": {
    "url": "https://t.co/Z75GkGKpyS",
    "expanded": "https://twitter.com/commonstreasury/status
      /1247895764304449538",
    "display": "twitter.com/commonstreasur…"
  },
  "contributors": null,
  "truncated": false,
  "text": "@bbclaurak @Peston",
  "is_quote_status": true,
  "in_reply_to_status_id": null,
  "reply_count": 0,
  "id": 1247912781648625700,
  "favorite_count": 0,
  "entities": {
    "user_mentions": [
      {
        "id": 61183568,
        "indices": [
          0,
          10
        ],
        "id_str": "61183568",
        "screen_name": "bbclaurak",
        "name": "Laura Kuenssberg"
      },
      {
        "id": 14157134,
```


Figure 18: Formatted JSON Example

8.6 Dependencies

All scripts in this project were written in Python 3.6.8 because all members of this group were familiar with the language and there are libraries available that can work with big data. Further, the language is highly readable, writable, and is one of the most popular languages today so it made it very easy to search for any problems we had with a library or our own code. The following libraries were used in the making of this project:

1. twarc
2. JSON
3. pandas
4. scikit-learn
5. joblib
6. os
7. sys

Further, we used a command called jq that helps make JSONl files more readable. This command can be downloaded following the instructions in Section 8.5.3.

9.0 Lessons Learned

During this project, our group faced challenges and difficulties. Because of this we fell a little behind on our project timeline, outlined in Section 9.1. However, we were still able to complete our goals for the project and provide a front-end interface that achieves all that we wanted. Section 9.2 outlines any problems we encountered, and Section 9.3 describes how we overcame the problems we discussed in Section 9.2. Finally, Section 9.4 describes future work that could be done on this project.

9.1 Project Timeline/Schedule

- February 15: Have the data server updated and Git repository set up. Researched and examined other COVID misinformation datasets from Kaggle.
- February 28: Be able to grab unique tweet IDs and store them in file/data structure.
- March 15: Gather news articles for mis/dis-information and label 1000 tweets as to whether they're misinformation.
- March 22: Completed back-end Python script that uses labeled dataset training data to determine if tweet provides mis/dis-information. Back-end scripts use Python libraries like Pandas, json, twarc, and scikit-learn.
- March 31: Complete back-end testing. Be able to filter tweets based on sentiment analysis. Researched different visualization tools for data.
- April 8: Have data organized and preliminary front-end UI using Angular to create front-end, detailed in Section 5.2.8 [14].
- April 15: Completed front-end of the COVIDFakeNews detector.
- April 26: Completed front-end testing. Ready for presentation.

9.2 Problems

Some problems that we ran into while doing this project was learning how to convert different types of data into JSON objects. Due to our limited prior knowledge of tweet extraction, tweet manipulation, and tweet hydration, we found it difficult to get started in terms of finding and extracting what we needed. Additionally, tweet objects can store other tweet objects if the tweet was quoted or retweeted, so there were many duplicate tweets. This made it difficult to differentiate between a tweet, a retweet, and quoted tweet.

Another issue that we ran into was developing a user interface. Although we are all computer science students, none of us had much experience doing front-end coding – most VT courses and internships were focused on the back-end. Additionally, connecting the front-end of our code and the back-end was difficult. Further, we didn't know how to make the user interface available online, once we had it completed.

9.3 Solutions

In terms of extracting data, we were able to solve our problems by learning how to utilize JSON objects under the Twitter V2 data model. This was made possible by hydrating tweets with the Twarc2 library, which returns a JSON object. Inside that object, we were able to parse through every tweet since each new tweet has a different header. This was done by reading line by line of the JSON file.

As for building the UI issue that we had, a few of our group members ended up learning Angular – utilizing Jquery, HTML, and CSS. This allowed us to implement a front-end that satisfied our client. As for the issue regarding connecting the front-end to the back-end, we solved this by implementing a reader for our UI, then passing in JSON files for the program to display.

10.0 Future Work

10.1 Front-end Future Work

When planning out the front-end, we made a list of what the client wants shown on it. Our client wants the timeline to show # of tweets per day, as well as certain statistics for each classifier we have created. However, since it is already set up to read from a JSON file, the timeline could be changed to work with a collection of tweets where a few tweets are shown for each day. On the statistics page, there could be a count of the users tagged by each classifier. On top of this, each classifier could have a word bubble that shows the most used words in each respective classifier. Additions could be made to the home screen, such as to have an explanation of our project, or information about the website and how it works.

10.2 Back-end Future Work

While lots of progress has been made on the back-end side, there are some improvements that could be made. To start, the machine learning model should be exposed to more Twitter content and content from a wider range of dates. As said before, initially Dr. Farag had stated that he had collected Twitter data for about a year, however looking closer at the data it seems that tweets were only collected for about a month. This severely limited the events and keywords that the machine learning algorithm was exposed to, and there might be a decrease in accuracy if it were exposed to tweets published today.

Next, the connection from the front-end to the back-end could be more fluid. Right now, the back-end creates a JSON file that the front-end just reads from for the statistics and the timeline of tweets, and it's not so dynamic. On top of the statistics that we collected, some analysis should be completed on the common URLs that had been included in fake news tweets. This could shed some light on the websites that are spreading false information about the pandemic, and which ones are reliable.

There is also another issue that is currently unaddressed, which is how the tweets were collected. When Dr. Farag, our client, was introducing us to this project, he mentioned the tweets were collected from April 2020 until May 2021. For some reason, tweets were only collected from early April 2020 to May 4, 2020. The client said to not worry about it since we discovered this late in the project. However, in the future other datasets could be used that span a longer period. Increasing the time span of tweet collection will also introduce more variety of events related to COVID into the dataset. If a subset of these new tweets can be classified and used to train the classification algorithm, then it might become more accurate.

Lastly, the hydration process could be sped up. Something we didn't anticipate was the slowness of hydrating around 120 million tweets. This came as a surprise because every other script we had written also looked at each individual tweet and only took a few hours. However, hydrating this many tweets took about a week. Upon closer inspection we noticed that the API account we had only allowed fifty requests every 15 minutes. Each request contained around 100 tweet IDs, but that is still slow compared to everything we had done so far. In the future, if teams

want to iterate on this project rapidly then it would be wise to upgrade to a more premium account where the Twitter API does not down slow the developers.

11.0 Acknowledgements

There are many people we would like to thank for helping us be able to put this project together and assist us when we needed it. First, we would like to acknowledge Dr. Fox for his guidance and advice we could count on throughout the semester. We'd also like Dr. Mohamed Farag for being our client and answering any questions we had regarding his vision for the project and specific technical aspects of the project such as the Twitter API. Without him, the project wouldn't exist, and we wouldn't have been able to learn about machine learning, front-end development, or interacting with APIs. Finally, we'd like to thank the Computer Science Department at Virginia Tech for giving us access to the tml.cs.vt.edu server. This server gave us a place where our team could work together and granted us enough disk space to work with the huge raw data files.

Client: Mohamed Farag:
Email: mmagdy@vt.edu
Professor: Dr. Edward Fox
Email: fox@vt.edu

12.0 References

[1] Twitter Dev, "Data dictionary: Standard v1.1," Twitter, [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet. [Accessed 1 May 2022].

[2] D. Giovanni, "Twitter Community," December 2012. [Online]. Available: https://twittercommunity.com/t/retweeted-status-retweet-count-always-0-in-streaming-api-tweets/14523. [Accessed 1 May 2022].

[3] S. Hames, "twarc2," [Online]. Available: https://twarc-project.readthedocs.io/en/latest/twarc2_en_us/. [Accessed 1 May 2022].

[4] Twitter Dev, "Twitter API v2 data dictionary," Twitter, [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/tweet. [Accessed 1 May 2022].

[5] S. Mondal, "Twitter Data Cleaning and Preprocessing for Data Science," 1 August 2020. [Online]. Available: https://medium.com/swlh/twitter-data-cleaning-and-preprocessing-for-data-science-3ca0ea80e5cd. [Accessed 1 May 2022].

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825--2830, 2011.

[7] R. Pupale, "Support Vector Machines(SVM) — An Overview," 16 6 2018. [Online]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. [Accessed 1 May 2022].

[8] R. Řehůřek, "parsing.preprocessing – Functions to preprocess raw text," 1 May 2022. [Online]. Available: https://radimrehurek.com/gensim/parsing/preprocessing.html. [Accessed 1 May 2022].

[9] S. Singh, "NLP Essentials: Removing Stopwords and Performing Text Normalization using NLTK and spaCy in Python," 21 August 2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/. [Accessed 1 May 2022].

[10] J. Korstanje, "The F1 score," 31 August 2021. [Online]. Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6. [Accessed 1 May 2022].

[11] J. Brownlee, "What is a Confusion Matrix in Machine Learning," 18 November 2016. [Online]. Available: https://machinelearningmastery.com/confusion-matrix-machine-learning/#:~:text=A%20confusion%20matrix%20is%20a,two%20classes%20in%20your%20dataset.. [Accessed 2 May 2022].

[12] Twitter Dev, "Academic Research Access," Twitter, [Online]. Available: https://developer.twitter.com/en/products/twitter-api/academic-research. [Accessed 1 May 2022].

[13] "How to Install jq(JSON processor) on RHEL/CentOS 7/8," 12 November 2020. [Online]. Available: https://www.cyberithub.com/how-to-install-jq-json-processor-on-rhel-centos-7-8/. [Accessed 1 May 2022].

[14] Angular Dev Team, "Guide to AngularJS Documentation," Google, 14 September 2016. [Online]. Available: https://docs.angularjs.org/guide. [Accessed 9 May 2022].