Covid-19 Fake News Detection

April 27th, 2023

Virginia Tech, Blacksburg, VA 24061

Client/Instructor: Dr. Mohamed Farag
By: Vincent DiPerna, Sungjeon Choi, Michael Blair, Safa Kamran
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>Abstract</td>
<td>4</td>
</tr>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Problem</td>
<td>5</td>
</tr>
<tr>
<td>Motivation</td>
<td>6</td>
</tr>
<tr>
<td>Approach</td>
<td>6</td>
</tr>
<tr>
<td>Requirements</td>
<td>9</td>
</tr>
<tr>
<td>Python Preprocessing</td>
<td>9</td>
</tr>
<tr>
<td>Website/Analysis</td>
<td>9</td>
</tr>
<tr>
<td>AI Model</td>
<td>9</td>
</tr>
<tr>
<td>Deliverables</td>
<td>10</td>
</tr>
<tr>
<td>Design</td>
<td>11</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>11</td>
</tr>
<tr>
<td>Website Design</td>
<td>12</td>
</tr>
<tr>
<td>AI Classifier Design</td>
<td>13</td>
</tr>
<tr>
<td>Implementation</td>
<td>14</td>
</tr>
<tr>
<td>Full Stack Website Implementation</td>
<td>14</td>
</tr>
<tr>
<td>AI Classifier Implementation</td>
<td>18</td>
</tr>
<tr>
<td>User Manual</td>
<td>24</td>
</tr>
<tr>
<td>Web Application</td>
<td>24</td>
</tr>
<tr>
<td>Developer Manual</td>
<td>28</td>
</tr>
<tr>
<td>Preprocessing Script</td>
<td>28</td>
</tr>
<tr>
<td>Full Stack Website</td>
<td>32</td>
</tr>
<tr>
<td>AI Model</td>
<td>35</td>
</tr>
<tr>
<td>Figure 31: Jupyter Notebook Import Statements</td>
<td>36</td>
</tr>
<tr>
<td>Lessons Learned</td>
<td>37</td>
</tr>
<tr>
<td>Timeline</td>
<td>37</td>
</tr>
<tr>
<td>Problems/Solutions</td>
<td>38</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>38</td>
</tr>
<tr>
<td>AI Related</td>
<td>38</td>
</tr>
<tr>
<td>Future Work</td>
<td>40</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>40</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Social Media Statistics................................................................. 6
Figure 2: Top 100 Keywords in Fake News.................................................. 8
Figure 3: System Context Diagram..............................................................12
Figure 4: Full Stack Development of Website..............................................15
Figure 5: Home Page of Website.................................................................16
Figure 6: Link Download Page of Website...................................................16
Figure 7: User Input Tool Page of Website..................................................17
Figure 8: Tweet Database.............................................................................18
Figure 9: Tweet List Page of Website..........................................................19
Figure 10: Load the dataset, filter tweets, and write the filtered tweets to a CSV. 21
Figure 11: Label each category in our dataset and combine data frames together........... 22
Figure 12: This python code splits, converts, and trains the model....................23
Figure 13: This python code uses scikit-learn library to test our AI model.............23
Figure 14: The output of evaluating our AI model........................................24
Figure 15: The python code used to generate joblib files................................24
Figure 16: View One..................................................................................25
Figure 17: Menu Functionality (Top Right Corner).......................................26
Figure 18: View Two..................................................................................26
Figure 19: Download Text File Buttons.......................................................26
Figure 20: View Three.................................................................................27
Figure 21: User Input Tweet Tool.................................................................27
Figure 22: Button to Detect Tweet Category................................................27
Figure 23: Results of Detected Category from Tweet Displayed.......................27
Figure 24: View Four...................................................................................28
Figure 25: Page Skipper, Item Number Viewer............................................28
Figure 26: One of Four Query Sections to Filter Results...............................28
Figure 27: An example of a tweettexts.txt file..........................................31
Figure 28: An example of a tweetlinks.txt file..........................................32
Figure 29: An example of a sqlcommands.txt file......................................33
Figure 30: Persistence Unit to Connect MySQL Database.............................36
Figure 31: Jupyter Notebook Import Statements..........................................37
Abstract

The Covid-19 virus is a respiratory illness that causes the isolation and retreat of people globally. People wanted updates and information in real time related to the virus such as what regions/areas are affected, to what degree are the regions affected (heavily infected, none infected, etc), how to prevent catching the virus, and cures for the virus. Social media became a popular platform for people to share information, news, and opinions about the virus. As much positive information that may be spread among social media, just as much, if not more, misinformation can be spread on social media platforms.

Misinformation is harmful because it can directly affect the health of individuals who fall victim to the misinformation. For example, say a twitter user tweets medical advice about Covid-19, and people who see the tweet choose to follow the advice. Now consider the scenario where they were intentionally spreading false information, which is indeed the opposite of what you should do. The individuals who followed the twitter trolls medical advice may have their own health at risk, and anyone in their sphere of influence.

Our aim is to understand the types of misinformation spread in social media, and help people identify misinformation spread on Twitter related to the subject of Covid-19. We’re going to do this by extracting relevant information such as the content of a tweet (the tweet itself) or the author of the tweet. Then, we will identify whether the tweets include true information or fabricated information. Once we do this, we are going to test and train an AI model to identify whether a tweet is spreading misinformation, or real information. After we train an AI model to identify the type of information, we will categorize the tweet into the category it was trying to spread information about. Our end goal is to integrate the preprocessing script and the AI model with a website that shows the analysis of the tweets. We want users to be able to insert a tweet into our website related to Covid-19, and the user should be returned with the relevant classification of the tweet. Also, users will be able to download a Web Archive file (WARC) of the archived tweet. Overall, we think the combination of these tasks will help aid users in identifying misinformation related to Covid-19.
Introduction

Problem

On February 11, 2020, the World Health Organization (WHO) officially designated the infectious disease caused by the SARS-CoV-2 virus as Coronavirus disease (Covid-19). The gravity of the situation was further recognized by the WHO on March 11, 2020, when Covid-19 was declared a pandemic, resulting in widespread lockdowns across the globe.[1] As accurate and timely information is essential during a medical crisis, it is concerning to note that a recent research paper revealed that almost half of the population gathers news from social media, primarily from Twitter.[2] It is therefore imperative that the information disseminated through social media during such a critical time is precise and reliable.

Figure 1: Social Media Statistics
Motivation

Misinformation about Covid-19 has been spread the most on the social media platform Twitter. Our goal is to accurately identify and categorize tweets related to the subject of Covid-19. A user can navigate to our webpage and input a tweet into our AI classifier. Our classifier will identify whether the tweet contains real information, or if the tweet is spreading false information in regards to Covid-19. If identified as false information, the classifier will tell you the type of misinformation the tweet is. This will help users become knowledgeable about the types of misinformation they are subject to, and help them avoid misinformation that may negatively affect their health.

Approach

- Defining Fake News And It's Categories
  - Determining what constitutes fake news and what does not is a complex process that requires consideration of various factors. To identify fake news from authentic news, it is important to consider the numerous forms of misinformation that exist, as outlined in a glossary of such terms.[5] In conducting our research, we referred to the report of a prior project that tackled a similar issue, examining their methodology as a starting point for our own process. Specifically, this prior study developed labeled buckets of tweets, grouping them according to the types of tweets they represented. The crucial aspect of this approach was the creation of at least one group of tweets considered to be truthful, allowing the model to distinguish between real and fake news.
  - Further research led us to two studies that analyzed data from Italy, China, and the US, identifying common keywords associated with fake news.[3,6] From these studies, we extracted the top 100 words used in fake news and identified key indicators of misinformation, including superficial attitudes, word choices that generate confusion, misinformative attitudes, racial attitudes, and definitive attitudes. This information guided the development of the classifiers we want to
use for identifying fake news, which included general misinformation, sources, vaccines, cases and testing, and truth.

<table>
<thead>
<tr>
<th>No.</th>
<th>Features</th>
<th>No.</th>
<th>Features</th>
<th>No.</th>
<th>Features</th>
<th>No.</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trump</td>
<td>26</td>
<td>Country</td>
<td>51</td>
<td>Money</td>
<td>76</td>
<td>democratic</td>
</tr>
<tr>
<td>2</td>
<td>President</td>
<td>27</td>
<td>Image</td>
<td>52</td>
<td>Federal</td>
<td>77</td>
<td>Racist</td>
</tr>
<tr>
<td>3</td>
<td>People</td>
<td>28</td>
<td>Russia</td>
<td>53</td>
<td>Sanders</td>
<td>78</td>
<td>Information</td>
</tr>
<tr>
<td>4</td>
<td>Obama</td>
<td>29</td>
<td>government</td>
<td>54</td>
<td>Woman</td>
<td>79</td>
<td>students</td>
</tr>
<tr>
<td>5</td>
<td>Clinton</td>
<td>30</td>
<td>don</td>
<td>55</td>
<td>administration</td>
<td>80</td>
<td>Military</td>
</tr>
<tr>
<td>6</td>
<td>Hillary</td>
<td>31</td>
<td>bill</td>
<td>56</td>
<td>City</td>
<td>81</td>
<td>immigration</td>
</tr>
<tr>
<td>7</td>
<td>White</td>
<td>32</td>
<td>man</td>
<td>57</td>
<td>Speech</td>
<td>82</td>
<td>Day</td>
</tr>
<tr>
<td>8</td>
<td>State</td>
<td>33</td>
<td>vote</td>
<td>58</td>
<td>Public</td>
<td>83</td>
<td>supporters</td>
</tr>
<tr>
<td>9</td>
<td>news</td>
<td>34</td>
<td>FBI</td>
<td>59</td>
<td>School</td>
<td>84</td>
<td>intelligence</td>
</tr>
<tr>
<td>10</td>
<td>campaign</td>
<td>35</td>
<td>law</td>
<td>60</td>
<td>Security</td>
<td>85</td>
<td>Senate</td>
</tr>
<tr>
<td>11</td>
<td>media</td>
<td>36</td>
<td>Americans</td>
<td>61</td>
<td>Presidential</td>
<td>86</td>
<td>War</td>
</tr>
<tr>
<td>12</td>
<td>video</td>
<td>37</td>
<td>World</td>
<td>62</td>
<td>Cruz</td>
<td>87</td>
<td>Department</td>
</tr>
<tr>
<td>13</td>
<td>police</td>
<td>38</td>
<td>Show</td>
<td>63</td>
<td>Family</td>
<td>88</td>
<td>Washington</td>
</tr>
<tr>
<td>14</td>
<td>Republican</td>
<td>39</td>
<td>Russian</td>
<td>64</td>
<td>National</td>
<td>89</td>
<td>United</td>
</tr>
<tr>
<td>15</td>
<td>America</td>
<td>40</td>
<td>GoP</td>
<td>65</td>
<td>Left</td>
<td>90</td>
<td>Voters</td>
</tr>
<tr>
<td>16</td>
<td>American</td>
<td>41</td>
<td>CNN</td>
<td>66</td>
<td>Group</td>
<td>91</td>
<td>Congress</td>
</tr>
<tr>
<td>17</td>
<td>Donald</td>
<td>42</td>
<td>Gun</td>
<td>67</td>
<td>Children</td>
<td>92</td>
<td>Comey</td>
</tr>
<tr>
<td>18</td>
<td>Black</td>
<td>43</td>
<td>Fox</td>
<td>68</td>
<td>Press</td>
<td>93</td>
<td>Office</td>
</tr>
<tr>
<td>19</td>
<td>women</td>
<td>44</td>
<td>told</td>
<td>69</td>
<td>Candidate</td>
<td>94</td>
<td>VE</td>
</tr>
<tr>
<td>20</td>
<td>election</td>
<td>45</td>
<td>Twitter</td>
<td>70</td>
<td>Investigation</td>
<td>95</td>
<td>Post</td>
</tr>
<tr>
<td>21</td>
<td>time</td>
<td>46</td>
<td>Court</td>
<td>71</td>
<td>Support</td>
<td>96</td>
<td>Does</td>
</tr>
<tr>
<td>22</td>
<td>Republicans</td>
<td>47</td>
<td>year</td>
<td>72</td>
<td>Wife</td>
<td>97</td>
<td>Protesters</td>
</tr>
<tr>
<td>23</td>
<td>house</td>
<td>48</td>
<td>Muslim</td>
<td>73</td>
<td>Interview</td>
<td>98</td>
<td>Great</td>
</tr>
<tr>
<td>24</td>
<td>party</td>
<td>49</td>
<td>states</td>
<td>74</td>
<td>Story</td>
<td>99</td>
<td>Did</td>
</tr>
<tr>
<td>25</td>
<td>political</td>
<td>50</td>
<td>Democrats</td>
<td>75</td>
<td>Attack</td>
<td>100</td>
<td>Violence</td>
</tr>
</tbody>
</table>

Figure 2: Top 100 Keywords in Fake News

- **Dataset: Tweets And Webpage**
  - After determining the appropriate approach for classifying fake news, we received a 40 GB JSON file comprising the gathered tweets. Subsequently, we undertook the task of transforming the data into a comprehensible format and establishing a way for communication between the data and the webpage using SQL/MySQL. Further elaboration on this matter will be provided on page *insert page number*.

- **Classifier**
  - Upon completion of data preprocessing, binary text classification will be employed to train and test the AI model, utilizing the sklearn library. Additionally,
Support Vector Machines (SVM) and term frequency-inverse document frequency (TFIDF) will be integrated into a pipeline to optimize the classification process, while irrelevant words will be removed. Further information regarding these techniques will be expounded upon in the upcoming page *insert page number*. 
Requirements

We have met with our client to establish requirements for completing our project. The requirements and deliverables we established are in the following sections.

Python Preprocessing

One of the requirements of our project was to use a Python script to extract data from a JSON file. The JSON file contained about 40 GB worth of tweets. The script is able to extract information such as the author of the tweet, the content of the tweet, and the category of the tweet when given to the AI model.

Website/Analysis

The client wanted a website that shows the results of our project. The client specifically requested that our website display a list of tweets from the dataset we were given. Each item in the list should include the Twitter user who tweeted, the actual tweet itself, and the category our AI/ML model put the tweet in. We will use a MySQL database as the backend of our website to help display these tweets. In addition, any user that visits the website should be able to input a tweet into the website, and it will notify the user what category our AI/ML model decides to put it in. (That being, either ‘vaccine misinformation’, ‘treatment misinformation’, or ‘other’.) Finally, users will have the ability to download .txt files of all the links used in our research. These come from the 40 GB JSON file we used for our data and is a list of tweet ID’s and the links found among each ID. One added functionality is that our website will have a PDF of this final report for users who want to know more about our research and how we accomplished everything.

AI Model

We are also required to test and train an AI Model to classify tweets into categories. The AI model will detect tweets it deems to contain misinformation. Anything not identified as misinformation of a specific category will be categorized into a category labeled as “other.” We are going to evaluate our model using a weighted percentage comprised of:
- The accuracy of detecting a tweet containing fake news
- The accuracy of detecting a tweet containing “other” news (mix of true news, unrelated news, and misinformation we haven’t trained the AI model to detect)

**Deliverables**

The deliverables of our project include:

- Python scripts to run preprocessing steps
- A web application to show the results of our analysis
- An AI classifier to classify tweets into categories
- Text file for the links used in research from the tweets
Design

![System Context Diagram](image)

**Figure 3: System Context Diagram**

Preprocessing

The preprocessing script is designed to read a JSON file filled with tweets, and extracts all the necessary data from it. This involves opening the file, iterating through it, and for each tweet, getting the important fields and storing that tweet with the important fields as an entry in a SQL database. This is important because the front end needs to be able to display the information in an easily readable and interactive way. This is done using JSON parsing methods, and removal of unnecessary characters using the regular expression library. For each tweet, the fields relevant are stored as columns in the SQL database, and the associated information for each field in each tweet is stored in the proper column in the database. This database will initially contain all tweets in the provided JSON file with text fields. The fields stored will be at the minimum the username, text field, and the category it is classified in by the AI model. The script is written in
Python, and is easy to run from a shell. The program also uses the argparse library from Python to make interaction with it easier. When running the program, if you provide the script a `-h` or `--help` flag, it will list all possible flags you can give the script to make it perform different functions. For example, you can provide the script with a `--v` flag, and it will show additional information about how the script is running including real time data on which tweet is being analyzed. The script creates 3 files. The first is called sqlcommands.txt and is a newline separated list of SQL INSERT commands for each tweet. This can later be used to build a SQL database for easier viewing in a front end implementation, or searching. The tweettexts.txt is a newline separated list of parsed text fields from each tweet and is the raw output that is given to the AI model for training. The tweetlinks.txt file contains links associated with each tweet and does not include other twitter links that were associated with a tweet.

**Website Design**

The front end is currently using a combination of tools. All the code for our website is written in the IntelliJ development tool. We used Java as our main language, and Gradle as our building system within a Jakarta EE generator. Further, we used Java EE 8 as some other versions of Java came with errors and technical difficulties. Our Module SDK chosen was the Oracle OpenJDK version 11.0.2 because it was most compatible with wildfly. We chose to host our website on a Wildfly, formerly known as JBoss, application server because we were most familiar with it. This supplies the framework so we can compile our project and view it on our localhost during the development process. The actual code takes advantage of the PrimeFaces library of UI Components for JSF, or Java Server Faces. JSF helps formatting the xhtml and makes the UI more compatible with different devices as well as gives our site a nicer layout. A primefaces-12.0.0-RC3.jar file was imported into our project so we could use JSF. Finally, we created a SQL file with four fields: an ‘id’ field to help identify tweets and order, an ‘username’ field with the tweet’s original tweeter’s name, a ‘text’ field holding the actual tweet, and a ‘category’ field to hold the detected category the tweet was placed in by our AI model. We parse the twitter data in the same python script as used before, clean it to format the text to fit SQL specifications, and create a TWEET table holding all the data. We used this file to configure the MySQL database on wildfly as a datasource, and connected it to a JBDC driver so we could easily import everything into our IntelliJ work environment. Finally, our AI script is hosted on a
Flask server. We call this Flask server any time the website needs to use the AI model to determine the category of a tweet.

**AI Classifier Design**

Besides this integration, we are going to train and test our AI model in Python. We will be working with the Scikit-Learn library in Python to test and train the model. The type of model we are using is a Support Vector Classifier (SVC). The reason we chose an SVC model is because it is a popular model to use when a program involves classification. Also, it works well with linear AND non-linear data. Our group tried using a variety of models like a Multinomial Naive Bayes Model, a Logistic Regression model, and an eXtreme Gradient Boosting model (XGBoost). The SVC model still yielded the best results out of the models we tried using. We will evaluate the accuracy of the model using a confusion matrix based on the number of categories we have.

In the context of our entire system, the AI classifier is integrated with our web application. We want to integrate the two because one of the main features of our website is to categorize tweets that the model detects as misinformation. As mentioned in multiple sections, the web application will allow for an input of Covid-19 related tweets, and the web application will return if the tweet contains vaccine misinformation about Covid-19, treatment misinformation about Covid-19, or it will return “other” if the tweets is not classified as either vaccine misinformation or treatment misinformation.
Implementation

Full Stack Website Implementation

The website uses a Java backend that consists of a link controller, tweet controller, tweet entity bean, and tweet facade bean to work with the MySQL database. The Tweet entity bean’s use is to define the class and fields so they can be accessed in other sections of the code. The Tweet facade bean’s use is to use a persistence unit and actually find all of the Tweet entity’s to create a usable list in the JSF code. The Tweet controller has two main jobs; one is to call on the facade bean to obtain the list of tweets when the xhtml page with data is loaded. The other is to connect to the Flask server running our AI model. Our full stack development process can be seen in the figure below. The user interacts with the monitor, goes through the JSP on the frontend (JSF in this webpage’s case), relays information back to the Java Beans, which goes back to the database to get the information requested. There is then a full traversal back and JSP (again, JSF in our case) sends the response back to the monitor.

Figure 4: Full Stack Development of Website

The website has four pages. The home page, or index, is simply an iFrame with this report so website users can learn more about our research process and what went into this project. The home page is shown in figure 5 below.
The second page is the ‘Link Download’ page. This page provides directions and buttons so the user can download a text file with all of the links used in this research from the JSON objects. The ‘Ajax’ option just enables the javascript function that creates a spinning circle to show estimated time of download, for which some browsers don’t support, hence the option. Once either of the command buttons are clicked, the linkController gets the streamed content file and returns it to the front end for download.
The third page is a user input tool. It allows a user to input the text of a tweet and use our AI model backend to see what category of news it should be placed in. When the user clicks the ‘detect tweet’ button, an action is sent to our backend Tweet Controller. This is where we have a method that uses the DataOutputStream to connect our website to the Flask server. We are able to send the user’s tweet input via a parameter in the URL we connect with, and then create a buffered reader to attain the output from our tool. The text below the ‘detect tweet’ button updates with the output category from our AI model after sending a POST request to our Flask server with the input tweet as the parameter.

The fourth and final page is the Tweet List view. This is a list view of our MySQL database imported into our project, containing 500 tweets. The SQL file used to create the database was filled in with the ID, the JSON data for username and tweet, and the category detected from our AI tool before being put into MySQL. In the case there was no username available, we filled in that field with the string “(Username Not Available)”. We tried to put in the whole database, but as configured through Wildfly’s server, the MySQL datasource cannot have more than 500 items in it. This could have been solved by using a different application.
server, but was discovered too late in the development process. The database can be seen in its IntelliJ view below in figure 8.

![Image of database view](image_url)

**Figure 8: Tweet Database**

We use the TweetFacade bean on the backend to collect all of the items in the database and create a `List<Tweet>` object to hold all the entries to show on this page. This UI includes different viewing options such as showing 5, 10, 20, or 50 items at once. The user can then page through all the tweets. There are also search queries for each category as well as a global search query if the user would like to sort or search for specific entries. The whole view can be seen in figure 9.
AI Classifier Implementation

We decided to classify tweets into 3 categories: vaccine misinformation, treatment misinformation, and other, which replaced the initial categories of vaccine, origin, and truth. The Covid-19 vaccine misinformation category will classify tweets that contain misinformation about the vaccine. The treatment misinformation category will classify tweets that contain misinformation about the treatment of Covid-19. Finally, the other category will contain tweets that are not deemed to be vaccine misinformation, or treatment misinformation. This may include tweets that contain true information, tweets that contain misinformation related to Covid-19 that we have not trained the model to categorize, and tweets unrelated to Covid-19.

The move to change the categories we wanted to classify tweets as was prompted by challenges with the origin and truth categories, as there were insufficient origin-related tweets for a robust dataset, and the truth category was too vague for effective filtering. By replacing truth with the "other" category, a broader range of unrelated topics was included, aligning with the primary goal of detecting false information rather than verifying truthfulness. Consequently, the model flags tweets with misinformation on treatment or vaccines and categorizes other content...
as "other," leading to a more focused and effective analysis of Covid-19 information shared on social media platforms.

As mentioned in the design section, we created a program in Python to test and train our AI Model. Before we could start testing and training our AI model, we had to preprocess the data we were given. We were given a 40 GB file that contained around 17.5 million tweets. With the models/methods we are familiar with, it’s a very complex task to have our AI model classify tweets as misinformation AND classify tweets as true information. The big problem here is how will a model be able to determine if a tweet is true or not? We took this issue into consideration, and we decided to manually categorize the tweets ourselves which reinforces our “Supervised Machine Learning” approach. However, it is nearly impossible for a group of 4 people to manually label tweets in a reasonable amount of time. Our group decided to create a Python program that uses a keyword match to find relevant tweets we want to use as our dataset for the testing and training of our model.

After researching keywords that are relevant to each of our categories, we filtered the 17.5 million tweets (which we loaded into a dataframe) and we wrote the filtered tweets related to its respective category into a comma separated values file (CSV). These files were named "vaccine_related_tweets", "treatment_related_tweets", and “other_tweets”. However, as mentioned above and due to a variety of challenges, it was necessary to manually categorize the tweets located in these files into CSV files named: "fake_vaccine," "fake_treatment," and "other." Any tweets that we classified as vaccine misinformation were placed in the fake_treatment csv, any tweets that we classified as treatment misinformation were placed in the fake_treatment csv, and any tweets that we classified as other were placed in the other_csv. The challenges described included the presence of duplicates even after filtering, the frequent use of acronyms, abbreviations, or misspellings, and truncated tweets due to character limitations. In some cases, it was essential to retrieve the complete tweets using their URLs for accurate categorization.
Furthermore, we encountered tweets containing URLs leading to external websites. After careful consideration, we decided to omit these tweets from our dataset to avoid the added complexity of verifying the authenticity of linked content through our program. This decision was made to streamline the process and maintain a focus on our primary objectives. Once the manual sorting process was completed, we were able to obtain a well-structured training dataset for our model. This dataset facilitated the development of a more accurate and reliable model by mitigating the issues encountered during the initial filtering and organization stages.

After splitting the dataset into 3 separate CSV’s, we loaded the CSV’s into their own dataframe, and labeled it either 0, 1, or 2. 0 represents the vaccine misinformation category, 1 represents the treatment misinformation category, and 2 represents our other category. Then, we combined the data frames and started the testing and training of our AI model.
# Read in the data

df0 = pd.read_csv('C:/Users/safak/Multimedia/ai/fake_vaccine.csv')
df1 = pd.read_csv('C:/Users/safak/Multimedia/ai/fake_treatment.csv')
df2 = pd.read_csv('C:/Users/safak/Multimedia/ai/other.csv')

# Add a "label" column to each dataframe and set the values accordingly
df0['label'] = 0
df1['label'] = 1
df2['label'] = 2

# Concatenate the dataframes into one dataframe
combined_df = pd.concat([df0, df1, df2], ignore_index=True)

# Impute missing values with an empty string
combined_df['tweet'] = combined_df['tweet']

Figure 11: Label each category in our dataset and combine all of the data frames together

We split the dataset into an 80/20 split. After splitting the data, we vectorized our data using a Term Frequency - Inverse Document Frequency vectorizer (TF-IDF). It’s necessary to convert our text data into a numerical representation that the AI model can understand. The classification model we used was a Support Vector Classifier (SVC) model. Next, we used the “fit” method from the scikit-learn library to train the model using our training data.
Figure 12: This python code splits the dataset, converts the data from text to a numerical representation, and trains the model using the training data.

We use the predict method from the scikit-learn library to make predictions on the testing data. In addition, we use built-in metrics from the scikit-learn library to evaluate our AI model. We evaluate the model using a parameter known as “average=’macro’” to display the results across all 3 of our categories. Another metric we used to evaluate our model is a confusion matrix.

Figure 13: This python code uses built-in metrics from the scikit-learn library to evaluate and test our AI model.
Finally, we generated joblib files for our model and vectorizer. The end goal of our project was to have a user navigate to our website and input a tweet related to Covid-19 to have it classified as misinformation or not. To prevent our webpage from training the model every time the page is loaded, we saved the already trained model and vectorizer into joblib files.

```
#Save the trained model and the vectorizer as files
joblib.dump(model, 'svc_model.joblib')
joblib.dump(tfidf, 'tfidf_vectorizer.joblib')
```

Figure 14: The output of evaluating our AI model

Figure 15: The python code used to generate joblib files.
User Manual

Web Application

A user that wants to use our web application just needs to navigate to the web page. Our original plan was to deploy the .war file of the project on an AWS EC2 server so anybody could view it at any time. This was deemed not necessary by our client. On our website, there are tabs displaying different areas of the application that the user can navigate too via the top right ‘MENU’ button. The home page has no user functionality, it is just used to view our final report.

Figure 16: View One
The second page, the link download, has instructions built into the page. The user simply presses either of the download buttons to get the .txt file of the links used in our research. The instructions on the web page notify the user that the file sizes may be too large to view in a regular ‘notepad’ (or any other basic text editor) application. The user may need to use something like Sublime Text or another application to actually view the large text file.
The third page is the Input Tweet page. The user is given 280 characters (current max limit of any tweet) to type into the text box and then must press the ‘Detect Tweet’ button. This will automatically display our AI’s results for the given text below the button.

![Image of the Input Tweet page with a text box and a 'Detect Tweet' button.]

**Figure 20: View Three**

**Bleach is a cure for Covid #WUHAN!**

246 characters remaining...

**Figure 21: User Input Tweet Tool**

![Image of the 'Detect Tweet' button with detected category results.]

**Figure 22: Button to Detect Tweet Category**

Our tool detected that your tweet was -- COVID-19 Treatment Misinformation

**Figure 23: Results of Detected Category from Tweet Displayed**
The final page shows the list of tweets and is again self-explanatory. Users can type in one of four query sections to search the database for any given strings. There is an item-number specifier up top and down on the button so users can view either 5, 10, 20, or 50 tweets at one time. Based on the total pages needed to view all the tweets, there is a page skipper so the user can just to any spot in the list.

![Image]

**Figure 24:** View Four

**Figure 25:** Page Skipper, Item Number Viewer

**Figure 26:** One of Four Query Sections to Filter Results
Developer Manual

Preprocessing Script

The preprocessing script is a Python script and is designed to be run with one argument provided, which is the JSON file containing the tweets. The repository can be cloned from GitLab from the following link over SSH

    git@git.cs.vt.edu:dipernavz/cs-4624-covid-19-fake-news-detection-spring-2023.git

In order to run the script there are several requirements. The script must be run using Python 3.11.2 since the AI files were created using that version of Python. The packages required to run the script are

- joblib version 1.2
- numpy 1.24.2
- scikit-learn 1.2.2
- scipy 1.10.1
- sklearn 0.0.post4
- threadpoolctl 3.1.0

It is possible that some slightly newer versions of these packages will work, but we only guarantee that the script will run with the exact versions of Python and packages listed above.

The invocation is as follows with tweets.json being the tweets file located in the current directory:

    python3.11 analyze.py tweets.json
The python script is designed to be run using Python 3.11.2. The script is called analyze.py. The tweets.json argument is required, and is a file of tweets in JSON format. There are numerous files generated including

- sqlcommands.txt → The list of SQL commands that can be given to MySQL to generate a database of the tweets and relevant information
- tweettexts.txt → The text field of each tweet separated by a newline, and containing the exact text given to the AI model for training purposes
- tweetlinks.txt → The links associated with each tweet, separated by a newline, and does not include links to Twitter URLs

These are some examples of the files generated as a result of the script. They can be several gigabytes in size depending on the number of tweets in the provided file.
Figure 27: An example of a tweettexts.txt file
The development of the website starts with setting up your environment. The first step in this process is setting up and installing OpenJDK. You need to find OpenJDK version 11.0.2 for this application. This is based on Wildfly’s recommendation to run Wildfly on the most recent long-term support release, so either OpenJDK version 8 or 11 will work. Next, you need to set up the jdk-11.0.2 as part of your path. I created the variable name JAVA_HOME corresponding to
the path of the jdk file, then added this variable to my path. Although technically unnecessary, in the long run, this step will make development easier.

The second step is to download MySQL on your computer. I chose to download the mysql-installer-community-8.0.32.0.msi for this project. I configured MySQL as ‘Server-Only’ when setting it up. Create an account for MySQL and continue with your download. Again, adding your MySQL download to your path will speed up development in some steps.

The next step is to download Wildfly. I downloaded Jakarta EE 8 Full & Web Distribution version 26.1.2 Final on the wildfly website. Once the .zip folder has been downloaded and extracted, use your terminal to go into the wildfly bin and run the command

```
\add-user.bat
```

Add a ‘Management User’ and create a username and password for your wildfly account. When prompted, the user should belong to all groups it asks about. Next, run

```
\standalone.bat
```

Leave this running, as it creates a standalone wildfly server that you can access on your localhost. Open another terminal and in the same directory run

```
\jboss-cli.bat
```

If this shows disconnected, type the ‘connect’ command. You can then access your wildfly console under [http://localhost:9990/console](http://localhost:9990/console). Before you set up MySQL on your wildfly account, you need to add the files to wildfly in your directory. Add two directories mysql/main/ to the path /…wildfly…/modules/system/layers/base/com, then copy the mysql-connector-j-8.0.32.jar file from your MySQL download into this directory. This file allows you to create the MySQL JDBC driver. Next, create a module.xml file with the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.5" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-j-8.0.32.jar" />
  </resources>
  <dependencies>
```

```xml"
<module name="javax.api"/>
<module name="javax.transaction.api"/>
</dependencies>
</module>

This will define the JDBC driver as a resource for your Wildfly account. Next, make sure your personal account has full control permissions as an administrator for Wildfly under its properties. Once all this is done, go back to [http://localhost:9990/console](http://localhost:9990/console) and set up your JDBC driver in configuration settings. The name should be mysql so it can connect to the folder you previously created in Wildfly. Wildfly and MySQL should now be completely configured together and ready for use.

Next, you need to download PrimeFaces so you can use JSF. I downloaded version 12.0.0 of PrimeFaces from the Maven Repository. This should be all you need to do in this step.

Finally, setting up the IntelliJ environment. The Ultimate edition is required for Jakarta EE cloud development, which is what I accomplished in this project. You can just download the latest version of IntelliJ IDEA Ultimate. You will need to create an account and activate it during the installation to gain access to the Ultimate benefits. In order to start the project, create a new Jakarta EE Web Application. Select Wildfly as the application server, and Gradle as the build system. Make sure to select Java EE 8 as the version, and include the full platform of dependencies. Next, populate your MySQL table with the SQL file of the Tweet objects on a terminal. If it’s included in your path, you should be able to just type the command

```bash
mysql -u root -p;
```
To sign in. Create a table and populate it with the SQL file’s data. Then, you need to start back up the Wildfly server and configure a datasource for the newly created MySQL table. You can give it a connection URL to connect to the database in your IntelliJ. Adding this persistence unit will allow you to directly connect to the database from wildfly and manipulate it from there as needed:
If you now wanted to, you could create that same project and copy over all xml, xhtml, and java files from my project and the same website would be created. As long as the same build.gradle file is also used and changed to your local directory. Running the project in IntelliJ with wildfly configured will automatically open up the website on your localhost. The longer way of doing things requires you compile and run the project in order to create the .war file in the /build/libs folder. This war file can then be deployed by running the following commands on two separate terminals:

```
path\to\wildfly\bin\standalone.bat
path\to\wildfly\bin\jboss-cli.bat
  connect
  deploy path\to\NameOfProject-1.0.war
```

You should then be able to access the website under localhost:8080/NameOfProject.

**AI Model**

The AI Model was developed in Jupyter Notebook. Some dependencies you need to install to test and train the AI model are:

- Python 3.11.2
After installing all of these dependencies (installed mostly using pip on my command line and specifying what I wanted to install), it’s very simple to start testing and training the AI model. First, you should create a new Jupyter Notebook by finding the Jupyter Notebook module in the Anaconda Navigator. As long as you create a .ipynb file in the same directory where your testing/training data is located, you shouldn’t run into any problems. Also, as long as you have the correct import statements, your code should run in Jupyter Notebooks smoothly. The import statements I used for filtering tweets into comma separated value (csv) files, for testing and training the AI model, and for generating joblib files are below.

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import joblib
```

Figure 31: Jupyter Notebook Import Statements
Lessons Learned

Timeline

Our project was undertaken with a general timeline, which helped us to stay on track and ensure that we met our goals in a timely manner. We established monthly deadlines that we wanted to achieve, and these helped to ensure that we remained on course and made consistent progress towards our overall objective.

By February, we had reached our goals for the month, which included developing a prototype of Python scripts, wireframing the website, and designing the homepage. These were critical early steps that helped us to lay the foundation for the project and establish key elements of the design and functionality.

As we moved into March, we worked diligently to complete the website and integrate the Python script. By the end of the month, we had achieved these objectives and had begun the process of implementing the AI model. This was a key milestone for the project, as it allowed us to begin integrating the core learning portion of our project.

Looking ahead, we have several important goals that we are working towards. These include finishing the AI model and training it, downloading WARC files from our website, integrating all components of the project, finalizing our report and deliverables, and preparing for the final presentation. These are all critical steps that will help us to put the finishing touches on the project and ensure that we are able to present a comprehensive and polished final product. Overall, we are pleased with the progress we have made to date and look forward to the next phase of the project.
Problems/Solutions

Preprocessing

During the preprocessing phase, there were numerous problems that were run into. For example, there was an issue with parsing the tweets in JSON format, where we could consistently get fields that weren’t garbled. At first we tried to implement regular expressions in order to split the data into proper fields, and save them into the SQL database. This proved to be too complex, and eventually we decided to use JSON parsing methods provided by Python. This turned out to be quite successful, and we were able to analyze the data into meaningful information that could be inserted into the database. Another problem we had was inserting information into the database. We would run an INSERT command in SQL to insert a tweet and relevant fields into the database, and we would later check the SQL commands file to find that no information was in the database. After thinking about it for a bit, we realized that we had to manually commit after each INSERT statement in order to write the record to the database. This was frustrating initially because it was hard to narrow down the problem, but once we added the commit statement, all the data showed up properly. Another problem is implementing concurrency into the program.

AI Related

One of the first problems we encountered was that none of us had any idea on how to test and train an AI model. No one in our group had previous experience testing and training an AI model, or doing anything related to the sorts. There was a large learning curve in learning how to approach our implementation, but once we got our hands dirty, we were able to figure out our implementation. One thing that helped ease the learning curve is that the content taught in class was relevant to our project. We learned the major concepts that need to go into testing and training an AI model.

Another problem we encountered was that we were not able to directly load the dataset into a Python program. We were given a 40 GB json file that contained around 17.5 million tweets as mentioned earlier. However, Pandas/Python does not allow a dataframe to be created if the JSON file contains multiple JSON elements/objects, and the json file we were given had
thousands. Luckily, we were able to preprocess the data so that it only contained the text from tweets, and this was output to a text file.

The three categories selected for analysis were vaccine, treatment, and other, replacing the previous categories of vaccine, origin, and truth. The decision to make this change was driven by challenges faced with the origin and truth categories. In the case of the origin category, after filtering tweets, it became apparent that there were insufficient tweets related to the origin of Covid-19 to create a robust training dataset. Conversely, the truth category was too vague, making it difficult to identify relevant keywords and effectively filter tweets that could be considered truthful.

To resolve these issues, the truth category was replaced with a more general "other" category. This change allowed for a broader range of unrelated topics to be included, encompassing anything not specifically related to treatment or vaccine information. This approach aligns with the primary goal of the analysis, which is to detect false information in tweets rather than verifying the truthfulness of content. As a result, the model will only flag tweets that it deems to contain misinformation related to treatment or vaccines, while categorizing all other content as "other". This refined categorization strategy allows for a more focused and effective analysis of the information shared on social media platforms regarding Covid-19.

An additional challenge encountered during the process was the time-consuming and laborious nature of manually sorting the tweets. Regrettably, our team was unable to devise an efficient solution within the given timeframe. It is our hope that future teams working on this project will take this obstacle into consideration early on, allowing them ample opportunity to brainstorm and implement more effective strategies for managing the data categorization process.
Future Work

Preprocessing

The future work that needs to be done in terms of preprocessing is concurrency in order to speed up the implementation. We have tested our script on the 17.2 million tweets provided, but it is relatively slow. In order to analyze the entire 40GB file of provided tweets faster, concurrency will be necessary for the file to be analyzed in a reasonable amount of time. We did not have time to finish the concurrency aspect of the project due to time constraints. The script runs at about 1000 tweets per second at the moment, and since the AI model slows down the script significantly (about 10X slower), this will make the process of making the script multithreaded easier.

AI Model

The future work of the AI model includes improving all of the metrics we evaluated when testing the model such as accuracy, precision, and generating a confusion matrix. One way this could be done is by increasing the number of tweets used for testing and training the model. The more quality data we include in the training set, the more likely it is for the model to improve. Another way the AI model can be improved is by including more categories to detect misinformation from. The other category is classifying more tweets as other than it should, and so is the vaccine misinformation category. We believe that including more categories will help prevent the AI model from accidentally classifying tweets as vaccine misinformation or accidentally classifying tweets as other.

Another way the AI model can be improved is by using a more advanced model. Although the model works well for us right now, if the dataset increases, a different model or a more advanced model may yield better results. If you want to take the project a step further, you could try training the model to determine if a tweet contains true information or misinformation, and then have it classify the tweet as a certain type of misinformation. I imagine this could be done using TensorFlow where there are multiple inputs with multiple outputs or some sort of model that utilizes a black box.
Integration

Once the AI model has been trained and finished, we will need to integrate everything into the back end of our website. This is a little more complicated than normal because of our choice to use JSF and a wildfly server. We will need to use and access the python script as a REST service. We’re going to start work on this once the python script has been fully trained to identify false news on twitter regarding Covid-19. Finally, once everything is properly integrated, we will need to upload everything, (the database, war file, REST service), to the aws ec2 server on Amazon so the website will be publicly available. Because of terms of use from twitter, this website will just stay up during the grading period.
Acknowledgements

We would like to thank Dr. Mohamed Farag for his contributions to this project. He provided us with the dataset, and met with us bi-weekly to figure out if our approaches were plausible. We all agreed on requirements that would make all parties happy, and this led to the creation of an interesting and impactful project. We would also like to thank Eric Wiley, Tung Nguyen, Fareeza Zameer, Ferrin Kirby, Kyle Toroc, and Campbell Dalen. This is the previous Covid-19 Fake News Detection team. Our predecessors left us with a lot of important information and resources to help us get started. In addition, their contributions have helped us with many things throughout this project. The presentation slides and report were especially helpful.

Client and Professor: Dr. Mohamed Farag:
Email: mmagdy@vt.edu
References


