COVID-19 Spike Prediction

Final Report

Bora Basyildiz

Enrique Gonzalez

Caleb Rotello

Samyuktha Senthilkumar

Client: Dr. Judith Klein, Associate Professor, Chemistry
Director of Bioscience & Engineering
December 1, 2020

Table of Contents

Introduction:

The COVID-19 Pandemic has taken over the world, causing massive shutdowns and changes in lifestyle. Despite stay-at-home orders and a myriad of restrictions in place to limit the spread of the virus, cases continue to rise. Our client is Dr. Judith Klein, an associate professor in Chemistry and director of bioscience and engineering at Colorado School of Mines. Our team works alongside Dr. Klein and the Catalyst Health Tech Integration (HTI), a networking hub for stakeholders in the health domain in Denver. After returning to campus in August, Dr. Klein wanted to see if there was a way we could predict these super-spreader events and potentially reduce the risk of further spread. That was when CSM Klein 1's project was born. We were tasked with utilizing the Twitter API to see if we could predict when spikes would happen based solely on what people tweet. These tweets would ideally reflect the lifestyle and actions individuals were taking, producing a source of keywords that would construct our pool for analysis. We hoped that with the live COVD-19 data that is currently available, we would be able to create a model that would be able to successfully predict large spikes in cases. Our model underwent two distinct phases: one to create a linear regression model, and one to utilize the power of machine learning and artificial intelligence to create a predictive neural network.

Requirements:

Functional Requirements

The functional requirements for this project include:

- Download historical infection data from John Hopkins website: github.com/CSSEGISandData/COVID-19
- Match geographical regions from John Hopkins to Twitter data
- Create a list of Twitter keywords that may be related to COVID19
    - Symptoms
    - All versions of coronavirus and COVID-19
    - Differentiate between SARS and SARS-CoV-2
    - Superspreader
    - Wearing masks
    - Events (church, parties, receptions, conferences, university openings)

    - Travel from places with high numbers (match dates to tweets)
    - Etc.
- Retrieve Twitter data from published data
    - https://publichealth.jmir.org/2020/2/e19273/
    - https://github.com/echen102/COVID-19-TweetIDs
- Retrieve our own Twitter data through the developer account
- Adhere to ethical guidelines provided by Twitter for use of data

Non-Functional Requirements

The non-functional requirements are as follow:

- The data utilized must allow Twitter users to remain anonymous.
- Adhering to ethical guidelines provided by Twitter for use of data

Use Cases

The users for our product are Dr. Klein and her team at Catalyst Health Tech Integration. The team will be able to do the following to get a prediction:

1) Enter Location
2) Enter a Keyword
3) Enter the Data Category
4) Prediction is Created

System Architecture

The design for our neural network can be seen in Figure 1 below. The diagram depicts how the Johns Hopkins data gets converted into an output vector and how the Twitter API goes through natural language processing to create an input matrix. Both of these are combined to go through the autoencoder neural network which then helps train and optimize a model to create a sound prediction.

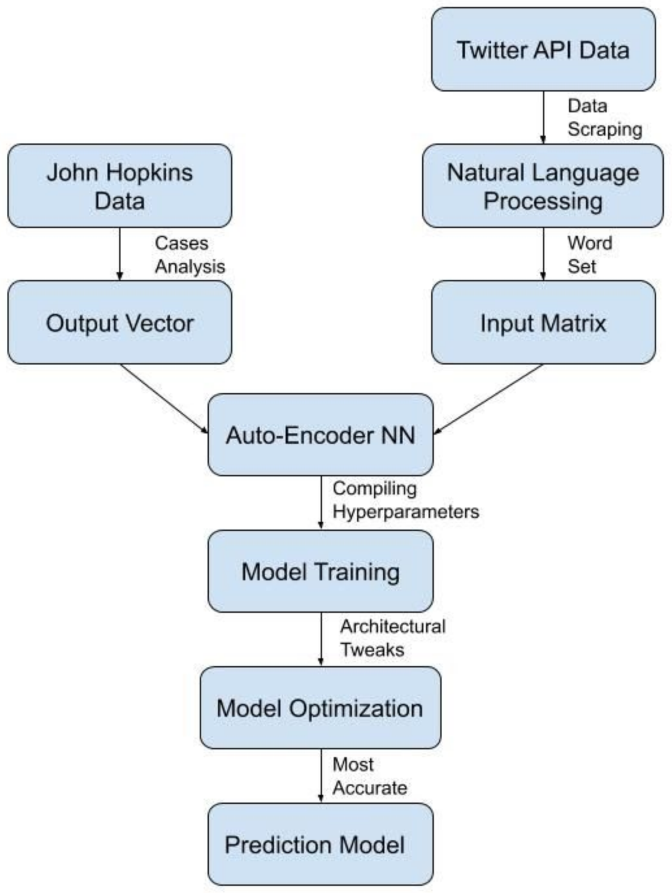

Figure 1: Depicts the architecture for the Neural Network approach

General Technical Difficulties:

- Twitter API Restrictions: We have Premium Twitter Developer accounts, which only have data going back 30 days. The API access level needed to access data older than 30 days is out of the client's budget, so this is the most data we can obtain. This level has a monthly tweet cap and a location radius cap.
- COVID Related Restrictions: The novel Coronavirus is documented, tested, and treated has evolved a lot since the first day in lockdown. New information emerges daily on how best to proceed during the pandemic. In general, positive cases are usually confirmed about 2 weeks from the initial event. We also know that no model can handle data about reinfected individuals well.
- Johns Hopkins data only contains confirmed cases & deaths
- Epidata API is not complete (incomplete telehealth and testing data)

- All data sets are formatted very differently and have varied methods of calculation
- Finding relevant keywords past the obvious “COVID”, “COVID-19”, & “Coronavirus” while being inclusive of keywords related to events (if possible)

Neural Network Specific Technical Difficulties

- Neural Network/ Deep Learning was new to most of the team
    - Implementation of the code for machine learning took more time than expected.
    - The number of labels for our targets was relatively minimal, which led to underfitting.
- None of the team had done Natural Language Processing before
    - The context and usage of words was hard to detect.
    - The natural language processing lead to a highly specific word-set, which could have caused some overfitting in our model.

Technical Design

One interesting aspect of our final project design is our approach to minimizing noise. Noise is a highly ambiguous term. Noise in what dataset? What type of noise? In our natural language processing we scraped tweets for a certain subset of words, and it is intuitive that some words matter much more than other words in relating to increases in covid cases. For example, the word “*shoe*” and “*mask*” should have contrasting utility for the model. The word *shoe* should have no utility for the prediction of covid cases in the near future, but *mask* should have a high utility. Thus some words will be regarded as noise and others as fundamental features. This is *not* just an intuitive observation.

We can observe in Figure 2 that most words do not occur in a high enough frequency to be considered valuable features. The only words that have a significant amount of occurrences can be seen in Figure 3.

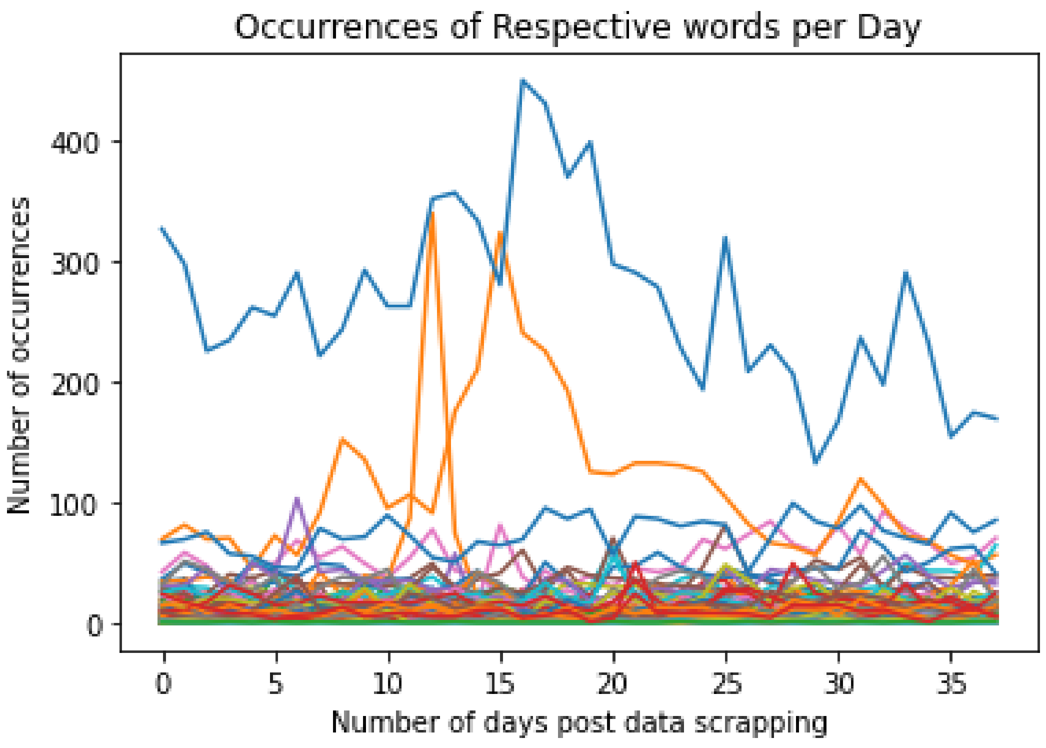

Figure 2: Graphs the occurrences of words over the period of data scraping

Aside from the words in Figure 3, there is a plethora of words that show up a negligible amount or not at all. These words are too sparse to separate their relevance to COVID from daily use, and are considered to be noise. To account for this, we can manipulate the neural network architecture to minimize the amount of arbitrary features needed. We can first start off with some Principle Component Analysis (PCA) to do an introductory minimization of the noise. PCA takes the amount of relevant features to 38. These do not specifically correlate to a number of words, as PCA transforms the feature space. Now we can orient our neural network to minimize even more noise. This was done with an auto-encoder structure that projects the features into a subspace. Through some testing, the most optimal subspace size was determined to be 15. Thus we can have a thorough auto-encoder structure that has many hidden layers and noise minimizing capabilities.

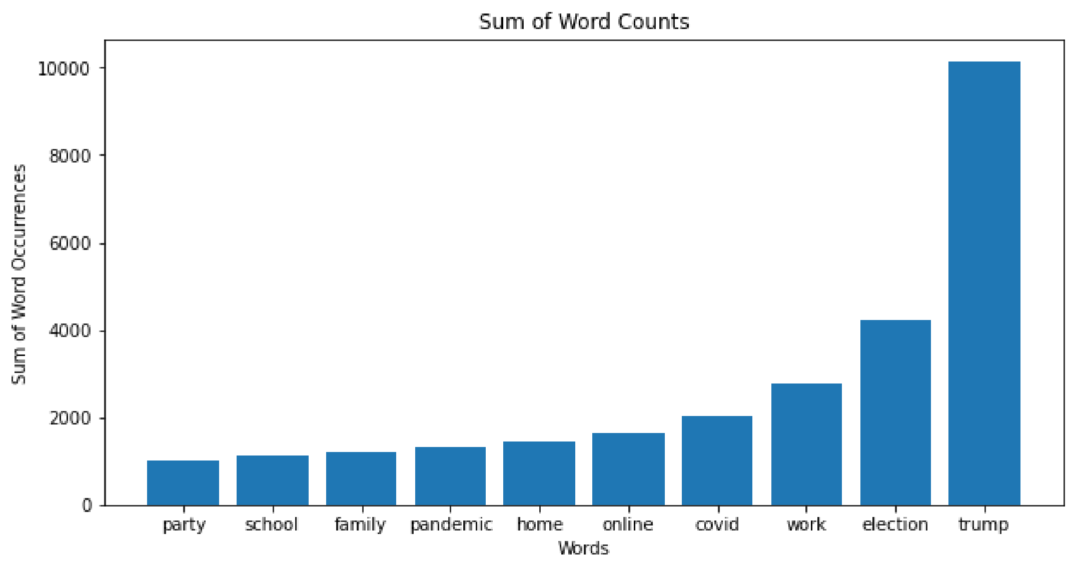

Figure 3: Sum of word counts in Figure 2

Another interesting aspect of the design is in our processes around the Twitter data itself. We use the Twitter API to get a histogram of keywords over time. This data is represented as a matrix of integers, with a column for each keyword and a row for each date of data. Because we have a premium Twitter developer account, we were subject to a series of restrictions on our data, such as:

- 30 day time frame of data
- 500 tweets per request
- 5,000,000 tweets per 30 days
- 5,000 requests per 30 days

Because of this, data is pulled from the Twitter API on a rolling basis. This restriction is especially limiting because the quality of our data is dependent on how we use the Twitter API at that time. During the initial data collection phases, we thought we were limited to 500 tweets per day for the first two months. This was due to a combination of the restrictions on data collection and the components required by the API for a GET request. And because we were limited to so few tweets, any spike in keywords which could have been correlational was indiscernible from noise. We did discover a way to paginate Twitter query results, which meant that we could get more than 500 tweets per day; we have 38 days of 5000 tweets per day. Even though this is a dramatic increase in the volume of our data, it limits the range of days for which we have data because the occurrence of any given keyword is very different for a collection of 500 tweets versus 5000, which makes all data more than 30 days old unusable. Because this functionality was discovered so late, we have been left with more days of unusable data than usable data.

Quality Assurance

Our software quality assurance is composed of two parts - preprocessing and processing. Preprocessing is the portion of our software where we initially developed our plan and model for COVID-19. Processing is the portion of our software where we finish training and create predictions based on our models. Below are the quality assurance items we used in both parts of our software development.

Preprocessing:

- Unit Testing: The model depends on a series of daily histograms of keywords as they appear in tweets. A Bash script collects 5000 tweets and outputs them to individual JSON files. A Python script collects these files and creates the daily histograms as a table. Unit testing using PyUnit was done to ensure two things. First, it ensures that the unicode tweets were transformed into ASCII format to compare with our list of keywords. It also ensures that the created histogram table has the proper counts for each keyword on each day.
- Code Reviews: The client needs to be able to get data in the future from the Twitter API, which is done via a shell script. We do code reviews to ensure that the script is clear and usable to someone with command-line experience. Each team member needs to be able to utilize the Twitter API script and corresponding Python cleaning script, so code reviews are also done to ensure that each of us are able to both use and modify each script as needed.

Processing:

- Unit Testing: We ensure that class types and pointers are conserved through computation. We also make sure inputs and outputs are correctly formatted and feasible. Since the processing is mostly done in Jupyter notebooks, we can use sections and code blocks of assertion statements that will halt the code if there are unit testing errors. This allows for incremental but effective testing throughout the developmental cycle.
- Integration Testing: Integrating the processing and pre-processing sections is vital to the effectiveness of our model. If we do not input the correct features and training/test data, we completely undermine the ability of our model to predict COVID-19 trends. Thus, we make sure that the integration with the Twitter data preprocessing was effective by utilizing similar approaches to the unit testing above.

- Code Reviews: Code reviews are extremely important Code reviews are extremely important to ensure that our code is efficient and that the model is trained correctly. There's no precedent or specific way that we know COVID-19 can behave so we have to be careful to adapt the model to make sure the data/methods we utilize are as complete and accurate as possible.

Results

Summary of Testing

- Each Tweet's contents are in Unicode format, so a direct string-to-string comparison was impossible if, for example, the keyword had an emoji at the end of it. We wrote tests to verify that non-ASCII characters were either transformed into an ASCII equivalent or removed from the string.
- We wrote tests to verify the following for processing the Twitter JSON files:
    - Every Tweet was being checked for its contents.
    - All keywords in each tweet were discovered.
    - A matrix describing a time series of histograms of the keywords was created and accurately depicted the data.
- The John Hopkins data was tested to be correctly collected, such that the number of cases per day and the per area was accurate and feasible for our model. We also made sure that the format within our data frames for the John Hopkins data was made in a way so that it could be processed.
- To make sure that the histogram from the Twitter data was correctly translated into our input matrix X, we made a many assertion statements and unit tests to make sure that our neural network was training correctly formatted and inputted data.
- All 4 of the Neural Network architectures were tested to have correct types, inputs, weights, outputs, and internal architectures. This ensures that if any errors arose, it would not be attributed to incorrect model coding.
- Results of usability tests were not relevant as we were not making a product for consumers.

Performance Testing Results

- Results aren't the best but there is correlation and that is a big step. Specifically, the early iterations of the models had an accuracy of 0%, and for a neural network, that is extremely low. Further optimizations raised the accuracy by 10-15% at times, but most of the models failed to find significant correlations within the data.
- We expected the neural network to be 80% accurate at the absolute best! Due to the nature of the input data, there was so much room for error, false positives, and other contributions to the compounding error that having an 80% was very optimistic. Thus a very low accuracy was not very surprising, as most of the error presumably came from the lack of accuracy and sparsity of our data. Inputs to the neural network were poor due to inaccurate, delayed hospital case reports and constraints of the Twitter API.
- There are two methods of collecting better data that we would have utilized given more time. First, we would have found a company in the areas of Boulder and Miami that more accurately reports COVID cases; however, these companies sell data at a high cost. We could have also done the same thing with the Twitter API; we could have paid thousands of dollars for the premium Twitter API to maximize the size of our Twitter dataset. But, paying an egregious amount of money for more accurate data was not feasible within the scope of our field session.
- We used 4 different neural network architectures
  - Auto-encoder deep learning neural network
  - Simple Recursive Neural Network
  - Gated Recurrent Units Neural Network
  - Long-Short Term Memory Neural Network
- These neural network models have been proven to find intricate and complex patterns within datasets, and the consistent poor performers from all of them do not display the ineptitude of neural networks, but rather the problems of our data that were stated beforehand.
- Different hyperparameters for each model were tested
  - Activation functions
  - Number of layers
  - Internal Input and output numbers within layers tested

- With the testing of the hyperparameters, some proved better than others, but none stood out. Some activation functions and layer numbers upped the accuracy by 1-2%, but nothing was able to make significant strides within our model.

Future Work & Implementable Features

- The first thing we would fix is the keywords utilized within our neural network. The keywords we are currently using are the ones our team came up with based on what we thought Twitter users might utilize to describe experiences they were having with COVID-19. As a result, these keywords were biased. Additionally, any set of keywords we created would be unlikely to capture every superspreader event. Our client introduced the idea of grouping together keywords, which is something to be explored in the future.

- We want to improve the accuracy of our model. A part of this problem is the current lack of access to accurate data.
    - From Twitter, we are only able to pull the most popular tweets of the day; we would ideally want more random data that captures many demographics including age, race, gender, etc. Unfortunately, this is impossible with our current API subscription.
    - While the locations we chose were intentional, having access to more locations would lead to a more robust analysis of the pandemic in the country overall.
    - From the medical data point of view, we would also love to have more consistent and accurate data. The Johns Hopkins data is currently the most comprehensive, but it does web-scraping and utilizes the minimal hospital data it can find to present the viewer with a general map of COVID, which works for its purpose. In the future, we should explore the Epidata API that Carnegie Mellon creates. This API would likely create a more accurate model for the US that would better suit our needs. Another alternative would be to acquire Google's user data as it would be the most comprehensive if they decided they wanted to sell it.

Lessons Learned

- Data validation is important. Our model treats the Johns Hopkins data as a complete sum of COVID-19 cases, but the data is only an accounting of hospitalizations. This functionally causes our model to correlate Twitter data with hospitalizations, not cases. That correlation is less likely to provide any predictive patterns, as most COVID-19 cases do not report to a hospital and will therefore not be used in the data.

- Given the impact COVID-19 has on different people (symptomatic, asymptomatic, and general variances in symptoms), it’s difficult to try and pinpoint a specific rise in sales of a product or products as another data set to showcase spikes in COVID-19 cases.
- It’s important to have datasets’ timelines coincide with each other to ensure data can mesh together easily and efficiently. The John Hopkins git repository has multiple branches, each with a varying frequency of updating their dataset. The Twitter data we pulled is most efficient when tracking spikes from 5-14 days before the acquisition date, given the time it takes for those infected to begin showing symptoms.
- Even if a causal relationship exists between datasets, just comparing numbers from those sets may not establish that relationship. In the context of this project, the Twitter dataset has a lot of noise, and the Johns Hopkins data is sparse. This makes separating the strong relationships from coincidence difficult, making the underlying relationship between the datasets challenging to discover.

Appendix

Twitter Scraping Instructions

Twitter scraping can only occur on a device that can run shell scripts. With that in mind, the instructions are as follow:

- From the root directory of the project, navigate to the `Twitter` directory
- The Twitter API is queried with `twitter-grep.sh`, run the command `./twitter-grep.sh boulder colorado data/more-boulder 30 5000` in order to get the last 30 days of data, with 5000 tweets per day, from Boulder, Colorado and save it to the `data/more-boulder` directory.
- Now we need to transform this data into a csv formatted histogram, both so the data is small enough to store on GitHub and so the neural network can read it. Do this with `python3 TextToHistogram.py data/more-boulder keywords.txt`
- You should have a new .csv file that contains your new data formatted