HealthFeed

Using Social Media Indicators to Visualize Outbreaks and Diseases

CS 4365 Into to Enterprise Computing, Spring 2019
Final Report

Maddie Brickell
Jackie Elliott
Kristen Goldie

Project Description and Motivation

Information regarding diseases and their outbreaks is available from various reputable source including the World Health Organization (WHO), Center for Disease Control (CDC) and Health.gov. These sources provide a collection of diseases worldwide as well as a list of reported outbreaks classified either by disease or outbreak time. While these sources provide accurate data, there is a lengthy process involved in verifying outbreaks that leads to a delay in alerting the public of any disease outbreaks. To fully investigate an outbreak, the CDC lists a 13 step procedure that involves cross evaluation with other studies, field work, and the establishment of the existence of an outbreak (2). This long process of confirming an outbreak prevents the near instantaneous dissemination of knowledge which humanity has come to expect and demand in the age of the internet.

Aside from being delayed in reporting, the knowledge of the outbreaks is not typically gained directly from the source (WHO, CDC, or health.gov). Public knowledge of disease outbreaks is typically acquired from television news rather than online resources. Based on a study conducted in 2016 and published by the Pew Research Center, 57% of United States adults get their news from the television and 38% of adults from online sources (3). The traditional news may only report regional outbreaks unless the outbreak has affected a large enough percentage of the population to warrant national interest. From watching the news on TV, it is difficult to develop a big picture regarding the diseases and outbreaks occurring worldwide, resulting in limited and delayed awareness.

HealthFeed redesigns the way one visualizes and learns about disease and outbreaks. HealthFeed uses social media to detect and classify potential outbreaks. The dissemination of information via the social network is nearly instantaneous, and this speed is unrivaled. In disaster scenarios, social media sources may be the first to report on the incident. HealthFeed uses this beneficial aspect of social media to provide an almost real time visualization of disease outbreak and prevalence around the world. Additionally, HealthFeed provides a comprehensive view of disease outbreak that may have been overlooked by traditional news sources.

Changes from Proposal

The project proposal outlined the work envisioned for this project prior to beginning the implementation process. Through the course of implementing HealthFeed we have made adjustments considering time, server specifications, and information availability. The adjustments resulted in changes to the scope of the project. Despite these adaptations, we have accomplished our goal of using social media as source for information about diseases and outbreaks and have developed a way to visualize the information we obtained. Although the project has been simplified, we are still achieving our goal and have a viable deliverable.

The changes made to the project from the proposal will be outlined in the Modifications to Scope section and detailed in the Implementation Specifics and Details section. In the implementation section one will see how the architecture and flow of the project changed.

Modifications to Scope and Challenges Faced

During the implementation of HealthFeed, we faced many problems inherent with project development as well as problems specific to the enterprise computing paradigm and related technologies. We integrated many services, software and hardware components including MySQL, MongoDB, streamers, web scrapers, servers, automation scripts and Flask. While we were familiar with some of the individual technologies, we had no experience integrating them. We were challenged with integrating many components in order to create a fault tolerant application that solved an identified need in society. Some of the challenges faced resulted in a change in scope in order to complete the project on time.

Only Using Twitter as a Media Source

Originally we had intended to gather data on diseases and outbreaks from multiple social media sources. However, due to time constraints we decided to only use Twitter as our source of information. It would not be difficult to add other media sources such as news outlets and Facebook. The LITMUS team has already created the code for streamers to scrape data from new sources such as Google News as well as Facebook posts about landslide events. In order to complete the project we had to limit our data source to Twitter.

We did not anticipate the difficulty we would have adapting the streamer code to fit our needs for Healthfeed. Some of the difficulty we had was due to our inexperience with streamers and unfamiliarity with the code. Although this might be considered a standard and almost trivial obstacle, it was quite significant given this project's short timeline. Changing the streamer code required a solid understanding of the functionality of each file, not just the output of the social streamer. We also encountered a lot of bugs we had never seen regarding running the program on the server and running the code in general. Ultimately, we were able to adapt the streamer code to the Healthfeed project so that it could download relevant tweets from Twitter.

Obtaining a List of Diseases

In our projected proposal, we outlined how we would gather a collection of diseases from reputable sources like the Center for Disease Control (CDC) and the World Health Organization (WHO). The WHO had the best list of diseases so we started by scraping those diseases from the site using the python libraries Beautiful Soup and urllib. We were able to successfully place these diseases into an array. We collected 71 disease names from WHO and, in order to keep progressing in the project, we decided this was an acceptable number of diseases and decided to not scrape the CDC's site to augment our list.

When running the code on the server for the first time near the end of our development cycle, we encountered errors with the python library urllib. The library was functioning properly, but the WHO site that linked to the disease page was causing issues. We were unable to determine why this specific site was not loaded and erroring (we were able to successfully load other pages). To focus on solving other more critical tasks in the project, we decided to hard code the list of diseases into one of our files on the server. Although we are getting less data from a limited set of diseases, we are still able to demonstrate the functionality of our project. The disease scraper was easy to remove due to the segmentation of the project. This segmentation allowed us to handle errors such as this with only minor adjustments. The disease scraping is no longer a dynamic process, but we are content with this change due the infrequency that new diseases are discovered and added to the list of diseases on the WHO website.

If the disease list was changing frequently, this error would have been more significant to success and quality of our project. We decided as a team that writing a python script to frequently scrape the list of diseases for the WHO site and update the values in the disease database was not critical to functionality and realism of HealthFeed.

Symptoms Collection

Originally we had planned to get our symptoms from WebMD and other sources. However, when we began implementing the symptom collection we were unable to gather data from WebMD due it's unpredictable format and small set of results. Due to the time constraint of the project, we decided it was not feasible to develop a comprehensive scraping algorithm for WebMD to gather symptoms. Symptoms are listed in many different location on the WebMB page if they are mentioned explicitly at all. Another problem was that some of diseases collected from WHO had no results in WebMD. Due the complexity of collecting symptom information from WebMD we decided to only focus on collecting information from Wikipedia.

Wikipedia had a relatively consistent format for disease pages. Most information regarding symptoms for a disease was found under the header "Signs and Symptoms". Using the python library Wikipedia we were able to easily search Wikipedia and load the page of the first result and scrape the page for the information contained under headers like "Signs and Symptoms". Only 14% of the diseases in the disease list had no search results on Wikipedia. Although we chose to narrow the scope of the symptom collection and use an unofficial source like Wikipedia, we demonstrated HealthFeed's ability to find symptom information from 86% of the diseases collected from WHO. In order to improve the accuracy of our ESA matrices used for keyword identification, we only included diseases in the hardcoded list that had search results in Wikipedia using the Wikipedia python library.

Front End Challenges and Reduction in Use Cases

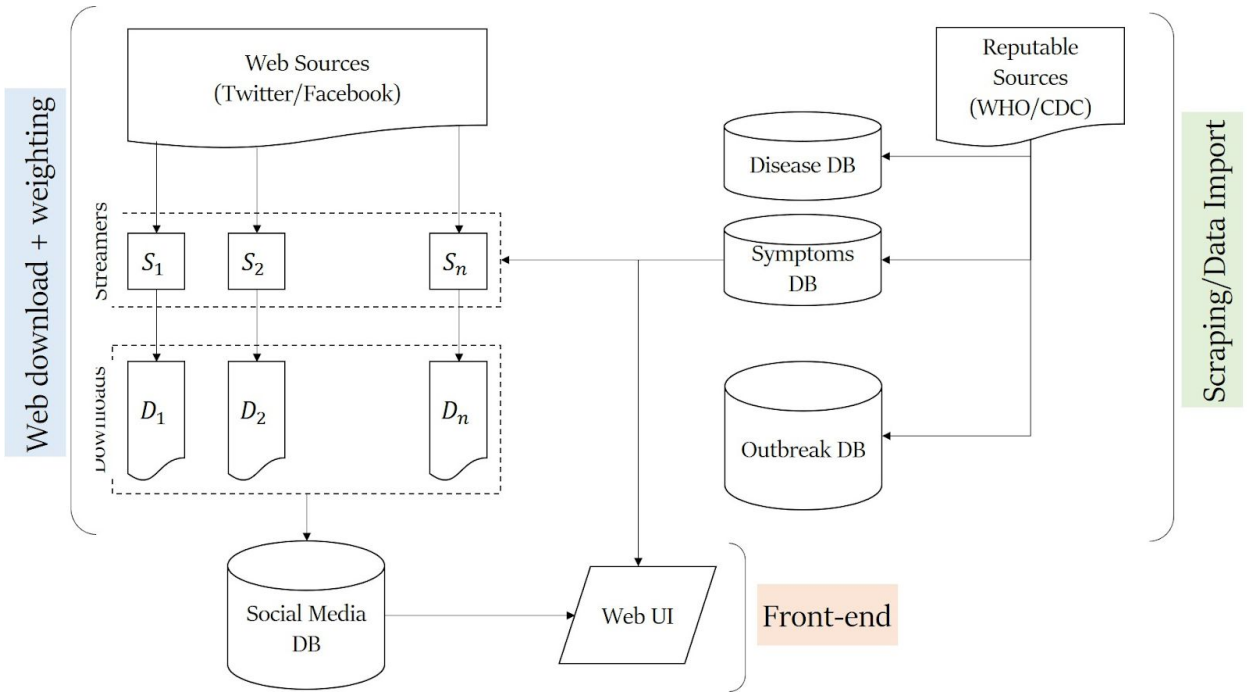
Kristen faced a few challenges when implementing the UI. The first was unfamiliarity with frameworks in general, and with creating maps in a website. It took a lot longer than expected to get Flask and the D3 map up and running, meaning that there was not enough time to

implement other less important aspects of the UI proposed in the original design, like the list of symptoms for each disease. Another challenge faced was encoding/decoding errors when trying to display the text of a tweet in the UI. It proved extremely difficult to resolve this problem, and so we decided to only display the link to the original tweet instead of the body of the tweet itself. It turns out LITMUS faced the same problem and implemented a similar solution.

Initially we had planned on using React and Node.js for the front end of our project due to our front end lead's experience with React. After further investigation, we discovered that Flask was a more feasible option with our server. However, our team had no experience with Flask and we had to adjust the complexity of the use cases we defined in the proposal to be able to allocate time in the schedule to learn and develop skills in a new framework. The main components of the website design proposed earlier in the semester remain in the final design. The most important feature is the world map, since this is how users are able to visualize the disease outbreaks. The next important feature is the list of diseases, which users can search and select to be able to choose which disease they want displayed on the map.

Features that were dropped from the original design include additional filters for the map, the list of symptoms for each disease selected, and the pop out tab including social media data that lacks a location. Additional filters like a data range were not included due to time constraints and are left for future work. A list of symptoms was not included due to the change in what information is scraped from websites. Instead of getting a list of each symptom per disease, we collect the entire paragraph describing the symptoms of the disease. In the future, a frame could be added that includes this paragraph and a link to the source, but time constraints prevented this from being accomplished during this semester. A tab including locationless social media posts was not included due to the volume of social media posts that lacked a location. It seemed that having a list of thousands of posts about a disease would be cumbersome to scroll through and difficult to deduce conclusions from. One of the main benefits of this project is the unique visualization of disease outbreaks on a world map. Locationless data cannot add to the main goal of the project and was therefore not included.

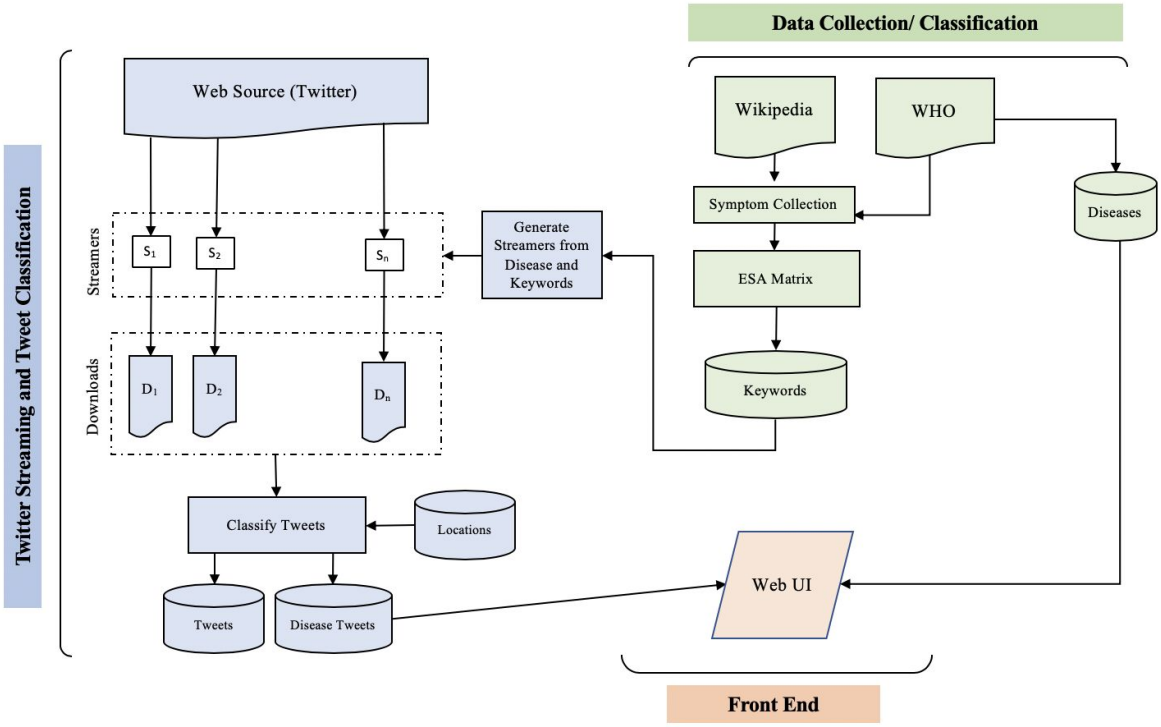
Original System's Architecture



Beyond the changes made to the scope of the project, we modified the system architecture of HealthFeed. The main framework and segmentation highlighted in the original architecture design still remains in our current model. It was imperative to preserve the segmentation in order to maintain the system of fault tolerance we originally outlined. Despite the unexpected challenges we encountered and the difficulties of implementation, the main framework of the architecture did not change. Prior to implementation, we successfully envisioned the major components of the architecture and their relationships, but we did not account for many additional features needed in implementation.

Implementation Specifics and Details

Current System's Architecture Diagram



Our system has many component which are segmented into the three categories: Data Collection and Classification, Twitter Streaming and Tweet Classification and Front End. The components were created with the assistance of the open source community and available language frameworks and modules. Additionally, we used off the shelf software tools for the web scraper and the streamers from the LITMUS group at Georgia Tech. HealthFeed is hosted on the Guangzhou server.

Architecture Components

- *Disease Collection* - A python script using the Beautiful Soup library scraped the WHO site at https://www.who.int/csr/don/archive/disease/en/ for listed diseases (5). 71 disease names were collected from the cite. The diseases are hard coded into our project. We intended for this process to be dynamic as we explained in the Changes to Scope section. The diseases can be stored in a MySQL DB, *diseases* if the script is run locally not on the server.

- *Controller.py* - Controller.py is python script that controls the process which scrapes symptoms from Wikipedia, populates the disease pure DB, uses TF-IDF, constructs the ESA matrix, and lastly populates the keywords DB.

- *Scraping Symptoms From Wikipedia* - The Wikipedia python library is used to search for each disease on Wikipedia and return a reference to that page as a Wikipedia Page object. If there are multiple search results for a a disease, the top search result is selected to parse. After obtaining the Wikipedia Page object we collect the text under the "Signs and Symptoms" header. 80% of diseases that had a search result on Wikipedia had a "Signs and Symptoms" header. For the remaining 20% of diseases that had a search result on wikipedia other symptom related headers were checked to see if they existed. If no symptom related header existed, the summary for the page was collected. 14% of the diseases collected from the WHO did not have search results on wikipedia. These diseases were discarded from our list of diseases.

- *Disease Pure DB* - The disease pure DB stores diseases and the raw symptom text collected from Wikipedia

  {_id: disease name, symptoms: "Symptom text"}

- *TF-IDF* - This stands for term frequency-inverse document frequency. This algorithm ranks how important a word is to a particular document. For our implementation, we used this algorithm to determine how important a symptom is to a particular disease.

- *ESA Matrix* - Also known as explicit semantic analysis, this matrix shows the particular TF-IDF ranking of a symptom, disease pairing. Every disease is normalized to prevent skewed data and this matrix will be used to create the keywords for our streamers and determine the relevancy of a tweet to a particular disease.

- *Keywords DB* - The keywords DB is a mongodb and contains the top five symptoms or strongly associated words with each disease. The controller.py obtains these keywords from the ESA matrix and adds the entry into the db collection. The information in this DB is used in the Twitter Streamer and Tweet Classification segment to initialize the streamers.

  {_id: disease_name, keywords:[“keyword1”, “keyword2”, ...] }

- *SocialStreamer.sh* - SocialStreamer.sh is shell script that sets up the environment to run the streamers and controls the streamer construction. The streamers run indefinitely and the script periodically checks if the streamers have terminated and re-initializes them if they’ve crashed.

- *Streamers* - The web streamers will twitter for relevant tweets pertaining to the diseases in the DB of diseases using the keywords DB to determine relevance. The results of the streaming are downloaded and stored on the server. The tweets are read from the downloaded files and ranked using our relevancy ranking system and placed in our tweets DB. There is one streamer per disease. When a new disease is added to the DB for diseases a script is triggered which will automatically create and run a new streamer.

- *Relevancy Ranking System* - Every tweet has a corresponding disease array that maps a disease to a value between 0 and 1 representing our confidence that the tweet represents that particular disease. This specific tweet/disease confidence value is obtained by iterating through the words of the tweet and summing the ESA Matrix value located at that disease/word combination.

- *Tweets DB*- After applying the relevancy ranking algorithm to the tweets downloaded by the streamers and stored on the server, the tweets with the ranking information are added to a mongo database.

{_id: tweet_id, text: tweet_text, disease: most_probably_disease, disease_array: [disease1:relevancy_of_disease_1], ner_location: tweet_location, ner_coordinates: tweet_coordinates, user: tweet_user}

- *Runner.sh* - Similar to the Social Steamer script, runner.sh will initiate an indefinite process of parsing tweets by extracting a location and classifying the tweet as a particular disease and adding the tweet to a list of tweets associated with that disease in the Disease Tweet DB. If the process crashes, the runner script attempts to restart the tweet parsing process from the last processed tweet in the Tweet DB.

- *NER* - Named Entity Recognizer is a software developed by Stanford. The software classifies words in a text as a PERSON, ORGANIZATION, LOCATION, etc. NER is used in healthfeed to identify the location of a tweet. Although you can tag a tweet and specify your location, the location may not be an actual location (for example "My bedroom"). To avoid this issue, we use the entire tweet text and specified location when applying NER.

- *Location DB* - One of the main issues with our algorithm is the processing time. Currently, it takes around 20 minutes to process 1 minute of data. Most of this processing time is dedicated to the NER software recognizing and classifying words. To improve the processing time, the locations mongo DB was intended to store the words that have already been mapped to a location. That way, these words would not need to be retagged by the NER software.

  Format of the database:

  {one_word_locations: ['Atlanta', 'Miami',], two_word_locations: ['San Francisco', 'New Jersey'], three_word_locations: ['New York City']]}

- *Disease Tweets DB* - The disease tweets database maps diseases with tweets relevant to that particular disease. This is used by our front end so that we can filter tweets based on the disease.

  Format of the database:

  {_id: disease, tweets : [tweet1, tweet2]}

- *Coordinate* - To obtain the coordinates for the front end from the location we used a python client, geopy. Geopy interfaces with many popular geocoder classes including OpenStreetMap Nominatim and Google Geocoding API. For Healthfeed we chose to use OpenStreetMap Nominatim. Nominatim accepts street names, landmarks as well as addresses. The location obtained by NER may not be a specific address so we decided it would be a good choice to use OpenStreetMap which didn't require a complete address to find coordinates. Not all locations found using NER have a coordinate result and in this case an empty coordinate value is assigned to the tweet.

- *Front end* - The front end uses Flask as the web framework, D3 for map functionality, and pymongo for interfacing with MongoDB. It uses a simple Jenja template for the web page. When the user goes to the website, tweet information is extracted from MongoDB, reformatted into a geojson document (a specific way to format geographical information), and fed into the D3 map, which displays the tweets as blue dots that correspond to the coordinates. When a user clicks on a specific disease, the same process occurs except that the information from the database is filtered so that only tweets pertaining to the chosen disease are displayed on the map. When users click on a specific tweet, information about the tweet is shown below the map, including a link to the original tweet.

Programmatic Flow

The initialization of HealthFeed begins by running controller.py on the server. This will populate the disease pure and keywords as described above. Next the shell file, called SocialStreamer.sh, for the the streamers can be run. Using the keywords found in the keyword database, this script

creates a streaming process to downloads tweets from twitter to the server. This process will execute indefinitely. Lastly runner.sh is a shell file than will execute indefinitely on the tweets obtained from streaming. In this process, we use the relevancy ranking algorithm described above to calculate the relevancy of the tweet for every disease. Again, we use NER to determine the location of the tweet. Afterwards, the tweet is stored in two databases - the tweets database that just holds all tweet information and the disease_tweet database that maps tweets to their most probable disease. From here, the data in these databases are ready to be queried from the front end and populate a UI.

Usability After Implementation

There are many ways Healthfeed can be utilized once it is put into production. For example, if a family was planning a vacation to a place on the other side of the world, they would want to know about any disease outbreaks that have occured recently that might influence their choice of destination. The family can simply look at HealthFeed's map, zoom in on the location they were thinking of, and look at any tweets that have been posted in the area, if any. In this way they can quickly get an idea of the state of disease outbreaks in the area they wish to visit. Another example would be a student researching a specific disease, or using the outbreak of a specific disease in a larger report. The student can visit the Healthfeed website, filter for the disease they are interested in, and see tweets about this disease from all over the world. They can then use the location information, or the tweets themselves, in the report they are writing.

Validation of Components

We initially validated this project by viewing our data and deciding whether or not the tweet accurately represented a disease. This method is not scalable and we are looking into a way to numerically validate our project and findings. Ultimately we will never be able to fully remove false positive result, but purpose of HealthFeed is to provide a general view of information regarding diseases and outbreaks . HealthFeed does not claim to be the most reliable source of information, but rather provides perspective and access to information which requires user validation.

Fault Tolerance

Although the three components of our project rely on one another to for data, each component does not need to be functional for the other components to continue operating. We successfully segmented the project so that a component could fail and as long the other segments could still access the data it needed, HealthFeed could continue to operate.

In the implementation phase it was critical to have this segmentation in order for the project to be fault tolerant in the case that a team member did not complete their assigned tasks and components. In general, even though our individual projects have large overlaps, we were still able to showcase our final project and no team member faltered.

Individual Responsibilities Achieved

Each team member was responsible for a section of related components for the project. Outlined below are the components each team member was responsible for creating and testing. Maddie and Jackie worked together on the backend components more than anticipated. The project still remained segmented for fault tolerance, but the backend implementation was more successful with Maddie's and Jackie's early integration and communication. Despite the collaboration, we were ultimately individually responsible for the completion or best effort completion (when significant effort is put into completing the project and overcome roadblocks) of our assigned components as described in our project proposal.

All team members completed their components. Although the scope and implementation details of the project change each team member adapted to the change and created an independently functional components of the project. We were able to successfully integrate our components and create HealthFeed.

Maddie - Backend Architect (Twitter Streaming and Tweet Classification)

At a high level, Maddie was responsible for taking the output of the streamers, classifying the tweets, and preparing them to be used by Kristen. Specifically, Maddie wrote the algorithm for the ESA matrix and generated the keywords used by Jackie to start the

Twitter streamers. Next, Maddie took all downloaded tweets and wrote the relevancy algorithm to rank the relevancy of the tweet for each individual disease. Together, Maddie and Jackie used the NER software from Stanford to classify the location and specific coordinates of the tweet. Finally, Maddie placed the resulting tweet elements in the mongodb databases to be pulled and used by Kristen.

Jackie - Backend Architect (Data Collection and Classification)

Jackie was responsible for scraping the WHO site for diseases, looking up symptoms for each disease, storing the data in a relational database and a document based database. Secondly, With Maddie's ESA matrix code, Jackie found the keywords and their ranking associated with each disease and stored that information in a MongoDB. To combine this work, Jackie wrote a script in python with the help of Maddie to integrate these processes. On the streamer side, Jackie wrote the function which allowed the streamer code to read from the keywords database to the create the streamers for each diseases. Using Stanford's Named Entity Recognition software to extract a location of the tweets, Jackie helped Maddie by getting the coordinates from the location found. Jackie helped put everything on the server and finalize integration.

Kristen - UI Designer and User Query Functionality (Front End)

Kristen was responsible for designing and implementing the UI for Health Feed. This involved creating a interactive map with clickable markers and implementing search functionality for the list of diseases. It also involved setting the website up with Flask as a framework and hosting the website on the server. Additionally, she integrated the front end with the database to obtain social media data and the names of the diseases. Once the social media data was obtained, she manipulated it into a geojson format in order to be displayed as points on the map.

Link to Github

https://github.gatech.edu/afnu6/HealthFeed

ReadMe provides more detailed instructions for connecting to the server, creating the relevant databases, and overall running the project!

Future Work

We have many ideas as to how we could improve and add to this project in the future. One thing that we would like to add would be a machine learning model that is able to classify and verify tweets and therefore reduce the number of false positives. We would follow the LITMUS machine learning model as an example as to how we could improve the accuracy of HealthFeed's tweet classification process.

Additionally, we would like to add more features to the website. We would like to add a section that gives some basic information about each disease displayed on the map, like a list of all the symptoms, and a link to where users can find more information. We would also like to add more filters, like a filter by date range, and add the ability to see more than one disease on the map at a time. Right now, users can see either all of the tweets we have collected on the home page, or click on one disease to see only that disease. In addition, we would like to add an analytics section or tab that analyzes the data we have collected and displays it in a user friendly way. Some things we could analyze would be the frequency of posts for a particular location or disease, or the amount of time between a cluster tweets about a disease and when CDC or WHO verifies that a disease outbreak did in fact occur.

We would also like to add more sources besides Twitter to HealthFeed. Our users would be able to get a larger quantity of information and a more accurate idea of diseases worldwide if we also pulled posts from social media sites used around the world. We would also like to pull from more reputable sources like news sights. Additionally, we want to include the most accurate information on diseases available, which is that from international disease oriented organizations like WHO and CDC.

We would like to improve the way that we determine location. Right now, we use one of the NER locations without doing any comparison or analysis to see if this is the most accurate way to use the NER software. This process also takes an extremely long time, so it would be an improvement if we could find a way to optimize this process. The way that Google and other

large companies deals with location determination is by utilizing a massive dictionary of places and names. We could try to incorporate this method into our design by saving the most commonly used location names in a dictionary; we would not currently have the storage space to save every location we encounter.

References

1. "About." *Mobile | HealthMap*, www.diseasedaily.org/about.
2. "Lesson 6: Investigating an Outbreak." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 15 Sept. 2016, www.cdc.gov/ophss/csels/dsepd/ss1978/lesson6/section2.html.
3. Mitchell, Amy, et al. "How Americans Get Their News." *Pew Research Center's Journalism Project*, Pew Research Center's Journalism Project, 14 July 2016, www.journalism.org/2016/07/07/pathways-to-news/.
4. "National Outbreak Reporting System (NORS)." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 7 Dec. 2018, wwwn.cdc.gov/norsdashboard/.
5. "beautifulsoup4." *PyPI*, pypi.org/project/beautifulsoup4/.
6. Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005),* pp. 363-370. http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf