Dynamically Prioritized Aggregated Multi-source Social Media Scraping Framework Integrated for Disaster Response Utilizing Twitter

Sistla, Pavan De Broux, Evan Cho, David
Millan, Christopher Post, Henry Pham, Trung
Renteria, Raul Valdez, Hasani Haswah, Rena

April 2019

1 Abstract

Tweets on Twitter[1] about true disasters are hard to easily detect and are often hidden among a sea of irrelevant tweets about only tangentially related content. We introduce a suite of tools to obtain tweets and classify them as relevant or irrelevant with regards to house fires and other disasters. In addition to this, other tools to aid in collecting, collating, and analyzing tweets are also offered.

2 Introduction

The American Red Cross wishes to use tweets collected from Twitter to respond to and monitor fire-related disasters. Traditionally, this is done by tens of volunteers who are limited in speed and accuracy compared to automated systems. In addition to this, the American Red Cross only performs a manual account search of a handful Twitter accounts attempting to identify incidents that they can respond to in the greater Chicago-area. It may be possible to do a keyword searches involving a keyword searches that may be relevant to incidents that the American Red Cross could respond to, such as "fire", "flood", "storm", and others keywords for types of incidents the organization could respond to. However, many tweets regarding "fire", "flood", et cetera on Twitter are either unrelated to an actual emergency or are re-tweeted from a different source, making detection of fire-related disasters on Twitter very difficult and a considerable waste of time and human resources that the American Red Cross could allocate elsewhere.

We propose an analytical model and web application by which tweets are collected and monitored, while also continuously analyzed in order to classify tweets as either referring to fire-related disasters or irrelevant.

[1] `https://www.twitter.com/`

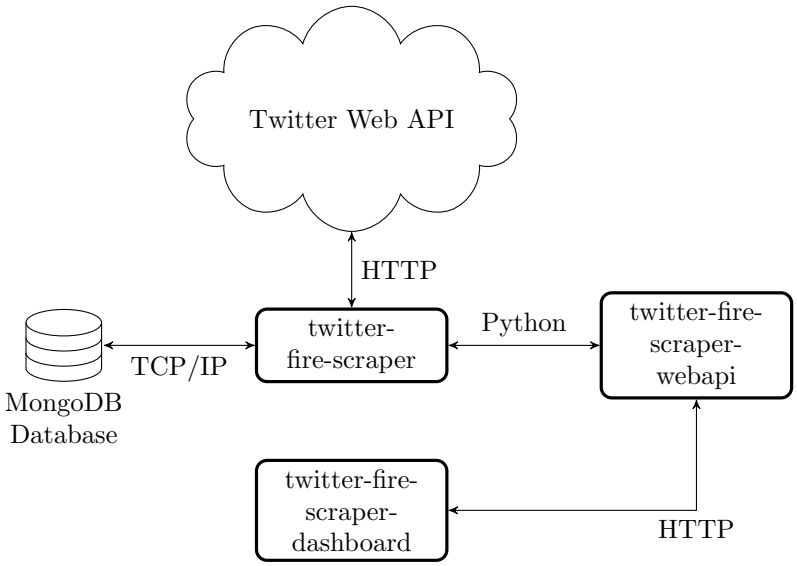

Figure 1: Application diagram

3 Model

Evan's model goes here

3.1 Regression Analysis

Math goes here

4 Scraper

4.1 Dashboard

The dashboard component, called "twitter-fire-scraper-dashboard"[2], works by querying a web API we have developed which allows a Node.js [3] web server to display a web interface for users to scrape and view tweets.

4.2 Web API

The web API component, called "twitter-fire-scraper-webapi"[4], exposes the functionality of our backend component to any applications which wish to consume the services it can provide over HTTP.

[2] `https://github.com/raaraa/IPRO497-Analytics-Team/tree/master/coding/twitter-fire-scraper-dashboard`

[3] `https://nodejs.org/`

[4] `https://pypi.org/project/twitter-fire-scraper-webapi/`

See figure 2 for the routes that the web API serves.

```
[GET]
/info

[GET]
/scrape
    ?terms=["house fire","mudslide", {...}]
    &accounts=["@NWSChicago","@MABASIllinois", {...}]
    &count=3

[GET]
/scrape_terms
    ?terms=<"house fire","mudslide", {...}>
    &count=5

[GET]
/scrape_accounts
    ?accounts=<"@NWSChicago","@MABASIllinois", {...}>
    &count=200

```

Figure 2: API Routes

4.3 Backend

The backend component, called "twitter-fire-scraper"[5], is a Python module which can send HTTPS reqeusts to Twitter's web API to retrieve, filter, and sort tweets from Twitter.

It has a variety of different tweet scraping filters that can be applied to it, such as location-based filtering, keyword-based filtering, and retweet-dependent filtering.

It facilitates saving scraped tweets to various sources, such as comma-separated value files, JSON files, and MongoDB databases.

It is designed to be used by programmers and data scientists, and thus simple to use yet powerful. This section contains demonstrations of how its interface can be used to perform scraping, collection, and saving of tweets.

4.3.1 Scraping by keywords

The first method that it can use to gather tweets is by keywords that the tweets contain. The scraper can take a single keyword and a count of results, or many keywords. Below is code showing the usage of the keyword-scraping feature:

```
# Returns a total of 600 tweets
results = scraper.scrape_terms(
    terms={"fire", "#housefire", "firedamage"},
    count=200
```

[5] https://pypi.org/project/twitter-fire-scraper/

```
)
```

Listing 1: Scraping multiple keywords

This snippet of code will return 200 tweets each about the term “fire”, the hashtag “#housefire”, and the term “firedamage” for a total of 600 tweets.

4.3.2 Scraping by keywords restricted by location

The next feature our scraper has is one which limits the areas that the scraper operates in. This is an augmentation of the scraper’s ability to search for keywords and hashtags, and acts as a filter on that function.

This is important because most of the work that the ARC wishes to do takes place in or around Chicago.

Below is a code example of the same search, but restricted to within 50 miles of Chicago’s center.

```
# Returns 600 tweets that occur 50 miles from Chicago
results = scraper.scrape_terms(
    geocode="41.8297855,-87.666775,50mi",
    terms={"fire", "#housefire", "firedamage"},
    count=200
)
```

Listing 2: Geo-location scraping filter

This time, we still get 200 tweets per term, but only ones that have been tagged within Chicago.

Instead of searching all of twitter for the three terms, only tweets that individuals have chosen to tag with a geographical location within a 50 mile radius of latitude 41.8, longitude -87.6, which is Chicago’s center.

4.3.3 Scraping by account

This feature allows one to scrape the most recently tweeted tweets by one or more accounts.

This is to make it easy to repeatedly query accounts, or to get large amounts of tweets from a set of accounts.

Below is a code example of the scraper getting the top 100 tweets from various Twitter accounts that the RedCross account follows.

```
# Returns 300 tweets from 3 red cross accounts, 100 each.
results = scraper.scrape_accounts(
    accounts={"@RedCross", "@NWSChicago", "@MABASIllinois"},
    count=100
)
```

Listing 3: Scraping by account

The syntax is very similar to that of the previous scraping methods. This snippet of code would result in a total of 300 tweets, 100 from each Twitter account.

4.3.4 Saving to a CSV file

This feature allows one to save scraped tweets to a comma-separated value file, to perhaps later be analyzed in a spreadsheet program like Microsoft Excel or be loaded into another data analysis program.

```
# Returns 300 tweets from 3 red cross accounts, 100 each.
results = scraper.scrape_accounts(
    accounts={"@RedCross", "@NWSChicago", "@MABASIllinois"},
    count=100
)

# Saves above results to a CSV file called 'output.txt'
scraper.save_statusdict_to_csv(results, 'redcross_tweets.csv')
```

Listing 4: Saving to CSV

4.3.5 Other features

The scraper allows you to scrape many more tweets than twitter normally allows you at any one time, and will wait when it hits the Twitter API rate limit[6] to finish scraping tweets.

5 Conclusion

"I always thought something was fundamentally wrong with the universe" [1]

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.

[6] https://developer.twitter.com/en/docs/basics/rate-limiting.html