Leveraging Elasticsearch and Botometer to Explore Volunteered Geographic Information

Samuel Lee Toepke
Private Engineering Firm
samueltoepke@gmail.com

ABSTRACT

In the past year, numerous weather-related disasters have continued to display the critical importance of crisis management and response. Volunteered geographic information (VGI) has been previously shown to provide illumination during all parts of the disaster timeline. Alas, for a geospatial area, the amount of data provided can cause information overload, and be difficult to process/visualize. This work presents a set of open-source tools that can be easily configured, deployed and maintained, to leverage data from Twitter's streaming service. The user interface presents data in near real-time, and allows for dynamic queries, visualizations, maps and dashboards. Another VGI challenge is quantifying trustworthiness of the data. The presented work shows integration of a Twitter-bot assessment service, which uses several heuristics to determine the bot-ness of a Twitter account. Architecture is described, Twitter data from a major metropolitan area is explored using the tools, and conclusions/follow-on work are discussed.

Keywords

Crisis Management and Response, Elasticsearch, Social Media, Volunteered Geographic Information, Botometer.

INTRODUCTION

Situational awareness is a critical component in the field of crisis management and response. Knowledge of a disaster space with high spatiotemporal resolution is necessary for directing supplies/responders, monitoring for further disaster, and responding to needs as they arise. Unfortunately, an area that has recently suffered a disaster usually faces compounding challenges including loss of power/water/environmental services, inaccessible roadways, limited communications channels, etc. With the recent modern proliferation of Internet-connected, robust, low-power smart devices, individual humans in a disaster space can provide a wealth of information (Goodchild, 2007). Online social media communities also enable the rapid distribution of information for an area. Provided the data is geospatially annotated with latitude/longitude, volunteered geographic information (VGI) (Zook et al., 2010) can be found in multiple forms, e.g. text messages from Twitter, images/videos from Instagram, Facebook stories, etc.

One of the difficulties of using social media data is processing the massive quantity of VGI and creating a cohesive product that can be used by managers and responders. This work will show an inexpensive enterprise architecture, comprised of open-source technologies, that can process streaming Twitter data from multiple geospatial areas in near real-time, and provide a flexible, web-based interface for data exploration. The entire environment is made from industry standard tools/software/infrastructure and can be readily replicated for new geographic areas, with limited human and financial cost.

VGI requires the user to be aware of trustworthiness (Halse et al., 2016) issues, as the producers cannot always be verified, leading to current decision-making models being resistant to its integration (Tapia et al., 2011). Inaccurate data, especially published by automated accounts called bots (Chu et al., 2010), can misrepresent the status of a geospatial area. This work will leverage a web-based service that processes information about a Twitter account and returns an assessment as to the account's bot-ness. Using this value, the aforementioned web-based interface can dynamically filter Tweets from accounts that have bot-like characteristics. This process

greatly increases the trustworthiness of the social media data and provides a higher quality picture to responders for a disaster area.

BACKGROUND

Understanding a geographic space has long been of importance to human communities; during a disaster, managing dynamic knowledge of a space is even more challenging. Social media is a relatively new concept and has been shown to support many steps in the lifecycle of a disaster, from pre-planning to recovery (Veil et al., 2011). Use cases are numerous, from community members sending images during the Deepwater Horizon oil spill (Merchant et al., 2011), to using wikis and text messages to affect response for the 2010 Haiti Earthquake (Yates et al., 2011) (Caragea et al., 2011), to processing Tweets from the New York City area during Hurricane Sandy in 2012 (Shelton et al., 2014).

Social media data, which can be explicitly collected by motive, or implicitly collected from ongoing streams (Aubrecht et al., 2017) can be challenging to obtain programmatically. Some social media services provide Application Programming Interfaces (APIs) which allow modern programming languages to communicate with them via web services. Consuming social media data via code allows for complex analysis and creation of interfaces. GNIP, a social media aggregator combines and delivers streams from many social media services, but the cost is out of scope for this work. GNIP sources include Facebook, Flickr, Instagram, Panaramio, Stack Overflow, YouTube, etc. (Sources, n.d.).

Once the data has been consumed, stored and processed, cohesive presentation to an end user is necessary. The data must provide some value, or meet a need, and be easily accessible technically and visually. Previous work has shown a web application that can display Twitter-sourced population estimations as a heatmap overlaid on a Google Map for a day of the week, and hour of day (Toepke et al., 2015). The application showed illumination into population movements for an area, but was slow, inflexible, and difficult to replicate. The Digital OnLine Life and You (DOLLY) (floatingsheep: DOLLY, n.d.) project utilizes a custom, enterprise grade Twitter aggregator and viewing interface; allowing for numerous publications, and easy insight into Twitter data. The Sahana Foundation publishes open-source software that can be used for social collaboration during a disaster situation (Li et al., 2013); the information management system supports preparedness, response and recovery.

While large/complex Twitter consumption projects exist, they require a large technical skill-set if a new researcher intends to replicate and/or extend the work. Twitter API access tools e.g. Twitter4j/Tweepy are well known and can be leveraged with online examples and custom glue-code; however, authoring a multi-user enterprise application to securely visualize the data while affecting rapid ad-hoc queries is non-trivial. This work shows the utilization of easily repeatable free and open source analytics tools, adapted to Twitter aggregation and visualization.

Also challenging is programmatically integrating trustworthiness assessments for each Twitter account, which can create a higher quality product and remove noise generated by Twitter bots. This work shows integration with the Botometer (Davis et al., 2016) service, previously known as Botornot, which was created by the Observatory on Social Media, a joint project between the Indiana University Network Science Institute and the Center for Complex Networks and Systems Research (OSoMe: Home, n.d.). Botometer is designed to take input about a Twitter account, including user information, Tweet timeline and Tweet mentions; after running the data through custom algorithms, bot-ness information on the Twitter account is returned.

The heuristics consider several factors for each account (Varol et al., 2017):

- Friends, investigates followers, follows, etc.
- Sentiment, analyzes mood/emoticons/etc. in the textual content of the Tweets.
- Temporal, considers times/dates/frequencies of the account's Twitter activity.
- User, evaluates metadata pertinent to the account, e.g. how built-out is the profile, the account's age, etc.
- Network, processes information diffusion patterns focusing on retweets, mentions, hashtags, etc.
- Content, invokes natural language processing to inspect the textual content of Tweets.

The analytics tools and Botometer are leveraged to process Twitter data, with the downtown area of San Diego, California in the United States of America being the geospatial area of interest. The coverage area is approximately 3 kilometers square and encompasses the downtown core. Data was collected from five cities in the United States, and one metropolitan area in Portugal; the selection of San Diego for this discussion was arbitrary.

ARCHITECTURE

During the recent massive growth/adoption of cloud technologies, a need has arisen to glean insights from disparate servers and applications in near real-time. This is difficult across security layers, during dynamic deployments of environments, with different applications logging different metrics, etc. To address this need, a set of tools has been developed: Logstash, Elasticsearch and Kibana (the ELK stack) (Gormley et al., 2015). The ELK stack products are open-source and maintained by Elasticsearch Global BV in Amsterdam, Netherlands, which has undergone rapid growth and expansion in the past several years.

Logstash is a small, Java based log shipping application (Logstash, n.d.). It is meant to be installed alongside any application, or inside any server/environment that needs introspection. Logstash is then configured to include the locations of logfiles on the system to ship to an Elasticsearch instance, security/connection data for the target Elasticsearch location, and any transformations that need done to the logfiles before shipping occurs. The flexibility of the transformations allows almost any software product or environment to provide meaningful data without changing the product itself. E.g. a log file for the Apache Web Server can have each line read into Logstash, the timestamp, user, log level, message etc., extracted, and then wrapped as JavaScript Object Notation (JSON) for sending to the Elasticsearch instance.

Logstash does have the native ability to directly attach to the Twitter Streaming API, which creates a low barrier to entry for a new researcher. The Twitter input plugin takes security values for Twitter, automatically makes the connections, and starts shipping Tweets. Since this work performs the intermediate step of attaining Botometer data for each account, this functionality isn't directly pertinent.

Elasticsearch is a schema-less, distributed, JSON based search engine (Elasticsearch, n.d.). The product is meant to be deployed to the cloud, accessible through secure connections, and it can concurrently absorb many large streams of data. A custom query language exists for Elasticsearch called Querydsl, which allows an end user to create powerful, low latency, ad-hoc queries.

Kibana is the user interface component of the ELK Stack and functions as a portal into an Elasticsearch installation (Kibana, n.d.). A user can login, and navigate to several pages, which allow for exploration of the underlying data. Kibana components include:

- Query terminal to directly explore the raw data.
- Charts that show trends over time, which can be filtered/populated based on keywords/queries.
- Map visualizations that display geospatial data, heatmaps, clustered information, and external layers from compliant Open Geospatial Consortium Web Map Services (WMS).
- Mashups of visualizations can be combined for a near real-time dashboard.
- A full enterprise solution with active user-management. Many users can login to one Kibana instance for data exploration using web browsers, e.g. Firefox, Chrome, Internet Explorer, Safari, etc.

This work leverages three freely available web services:

1. The Twitter Streaming API (Consuming streaming data --- Twitter Developers, n.d.).
2. The Twitter Representational State Transfer (REST) API (Standard search API --- Twitter Developers, n.d.).
3. The Botometer service via Mashape (Botometer API Documentation, n.d.).

To implement the reliable consumption of data from the Twitter Streaming API, a solution is created using Java, Docker and Amazon Web Services (AWS). The code is repeatable, resilient to failure, and is highly available via the cloud.

The primary data processing and visualization toolset is an implementation of the ELK stack. Elasticsearch and Kibana as-a-service is implemented via Elastic's Elastic Cloud (Elastic Cloud: Hosted and Managed Elasticsearch | Elastic, n.d.). Full architecture can be visualized in Figure 1.

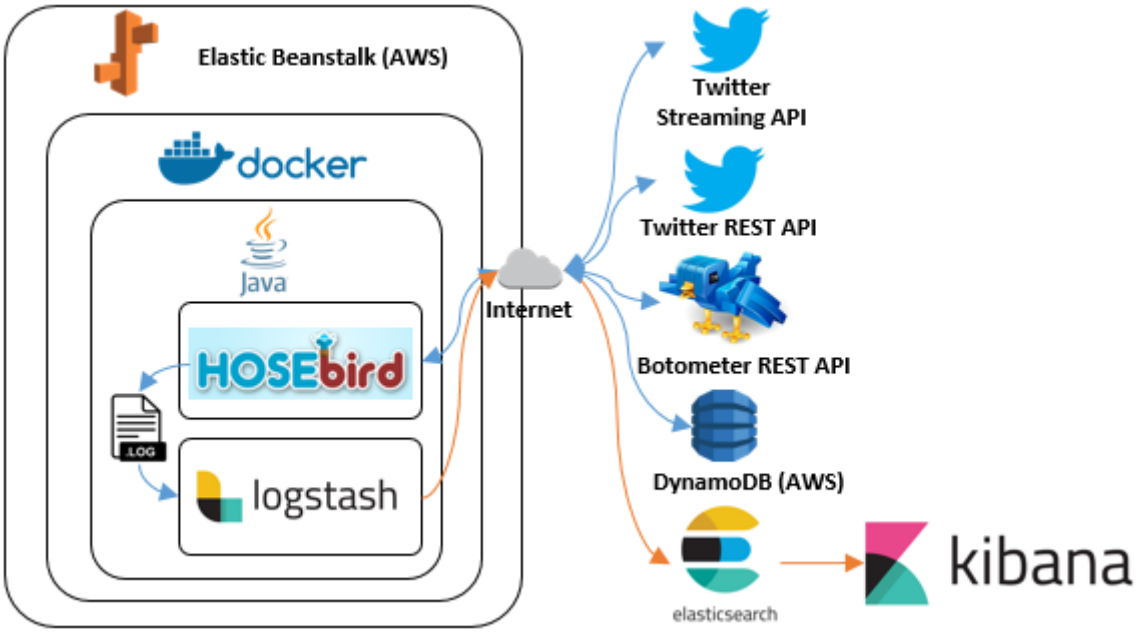

Figure 1. Architecture Diagram

Twitter APIs and the Botometer Service

Twitter provides two primary APIs for developers, the Streaming API, and the REST API. While the REST API is convenient to query, it has limitations on the number of queries, and the number of Tweets that can be returned for a time frame. While useful for testing and prototyping, the request/return model is not suited for large geographic areas.

The Streaming API operates on a different principle: if a connection can be made/maintained over the Internet, the API will constantly stream Tweets back to the consumer. This requires a different architecture from the REST API, where issues with processing/connections/streaming need to be resilient and handled gracefully such that the connection is maintained. In this case, before the streaming connection is made, quadrangular geographic boundaries are set for each metropolitan area of interest.

The Twitter REST API is used during the Botometer query process to populate the information passed to the service. Botometer requires two queries per new account to the Twitter REST API; to get information about the user's Tweet timeline, with up to 200 Tweets, and to get information about the user's Tweet mentions, with up to 100 mentions. This process is non-trivial, and a Java library was created to abstract this functionality (GitHub - samueltoepke/Botometer4J, 2018).

Due to current Twitter REST API constraints, only 180 requests are allowed per 15-minute window (Rate Limiting --- Twitter Developers, n.d.), thus limiting the number of actual Botometer queries to 90 per 15-minute interval. The collection code makes use of a cloud data table as a cache for Botometer values per account, which helps limit calls to Botometer. Nonetheless, if the Twitter API reaches its rate-limit, then a 15-minute rest occurs, and any Tweets with account names that haven't been previously loaded into the cache are shipped to Elasticsearch with "null" Botometer values. The consumer code immediately starts querying the Botometer API, populated from Twitter REST calls, as soon as the rest is over.

The Mashape Marketplace is an online service that allows authors to host their APIs. Botometer is deployed via Mashape and is also constrained by rate-limiting rules. Botometer allows 17,280 free requests per day, per authenticated user; if a customer needs more, they can register for a "Pro" account, and pay for more requests. Since two Twitter REST API queries are required per each Twitter username, this architecture is throttled to the Twitter REST rate limit, and the Mashape free limits will not be reached.

When the Botometer service is queried for a Twitter user name, a JSON string is returned with a score for each of the aforementioned categories. Also returned is an average score for all categories for English Tweets, and a universal score for non-English Tweets, which removes the content and sentiment scores. Each unique Tweet contains a language field which can be accessed to programmatically decide whether to utilize the English or

universal score. Each score is a decimal value from 0 to 1.0, with values towards 0 being less bot-like and values toward 1.0 being more bot-like. An example response from Botometer can be seen below.


```
{
  "categories": {
    "friend": 0.45,
    "sentiment": 0.34,
    "temporal": 0.55,
    "user": 0.29,
    "network": 0.43,
    "content": 0.41
  },
  "user": {
    "screen_name": "IUNetSci",
    "id": "2375509748"
  },
  "scores": {
    "english": 0.34,
    "universal": 0.36
  }
}
```


The score breakdown per the Botometer documentation is as follows.

- 0.0-0.39, likely not a bot.
- 0.40-0.60, unable to tell if account is a bot or not.
- 0.61-1.00, likely a bot.

The Botometer documentation clarifies that these classifications are difficult, and while they provide insight to an account, can be incorrect.

Amazon Web Services

AWS provides a variety of cloud-based computational services, that are well documented, inexpensive and offer high uptime and reliability. The code that connects to the various APIs and the Elastic Cloud is implemented using the following AWS products.

- Elastic Beanstalk (ELB), an orchestration service that provides an execution environment; essentially an application server as-a-service. The Docker execution environment is utilized, where a custom Linux based container hosting the Java consumer code resides. The container is designed to be secure, repeatable and ephemeral. If anything goes wrong with the underlying virtual machine, ELB will terminate it, and immediately create a new machine with identical functionality.
- DynamoDB, which functions as a datastore as-a-service. A single table is used as a cache for the Botometer data where each row contains information for a single Twitter account. This table is queried before Botometer is called, with the idea being mostly the same people are Tweeting in mostly the same geographic boundaries. Over time, the table is loaded such that far fewer calls to Botometer are required. Each row also has a flag containing a timestamp for most-recent-update. As a Twitter user's facets are continually dynamic, if the Botometer data for a user is over a week old, the data is discarded and updated via a new call to Botometer.

The Twitter Streaming API consumer code consists of the following components.

- A standalone Java (Arnold et al., 2005) application that begins running when the Docker container is started. A combination of Cron/flock manage the Java process and will restart the process if it is down. The Java code uses Hosebird Client (HBC) (GitHub - twitter/hbc, 2014), a 3rd party Java library for managing the connection to the Twitter Streaming API. HBC handles security, making connections to the API, acting on received Tweets, re-enabling failed web connections, etc. The Java application is multi-threaded, spawning a new thread for each received Tweet. Populating a request for, and reaching out to Botometer can take time (between 5 and 10 seconds on average). Threading the application prevents the web service calls from blocking the processing of incoming Tweets. Apache Log4j 2 is used for configuring and rotating the log files. Each new thread takes the Tweet and checks the DynamoDB cache for Botometer information. If data for that Tweet's account is not there, or is there and over a week old, then the Botometer library is used to get the data from the Botometer service, assuming a global rate limit break is not in place. If the rate limit break is active, then the Botometer data remains null. The local Tweet is updated with retrieved Botometer data, and the data is also added

to the DynamoDB cache. Finally, the Tweet is printed to the log in JSON format. The main program flow for the Java application can be found as pseudocode in Appendix A.

- A Logstash installation, which ships logs to the Elasticsearch/Kibana stack. Logstash is Java based and is run when the Docker container is started. Cron/flock also manage this process and will restart Logstash if it fails.

The above describes an enterprise solution for Tweet consumption and log shipping to an Elasticsearch/Kibana stack, though is not entirely necessary. Crisis management and response is fluid, and the overhead needed to leverage AWS may be a blocker. The consumer code and Logstash are both Java based, which has an implementation for many modern operating systems. Any user with a laptop and Internet connection could easily run the two products from the command line in a time-constrained situation. The setup would not be as resilient and require human over-watch; but would reasonably attain and ship Tweets in a crisis.

The ELK Stack

This work uses the Elastic Cloud, a fully managed Elastic/Kibana as-a-service product, which greatly abstracts the configuration, deployment and management of the underlying servers, for low cost. Once setup, web endpoints and security values are provided, which are used in the consumer code's Logstash configuration. AWS does provide Elasticsearch/Kibana as-a-service, though configuration is non-trivial.

The described architecture is deployed in cloud environments, uses a fusion of open-source products, and deploys an interface such that a multi-user application is available for a minimally trained user to make powerful queries/visualizations on near real-time, textual data.

RESULTS

Once the entire stack is deployed and shipping log files, Kibana's user interface can be used for active inspection of the data. Figure 2 shows the main Kibana screen in the Firefox web browser. Points of note as follows.

- The top right of the screen shows "Last 5 Days"; clicking here allows the user to dynamically select any custom time period, which quickly repopulates the graph, and list of Tweets.
- The main portion of the screen shows a running list of the most recent Tweets, and all pertinent data. Refreshing the page will display Tweets that have been received since the last page load.
- At the top of the screen, there's a query field, which allows the user to explore the data using the Lucene query syntax (McCandless et al., 2010).
- The left side of the page is a menu for visualizations, dashboards, management, etc.

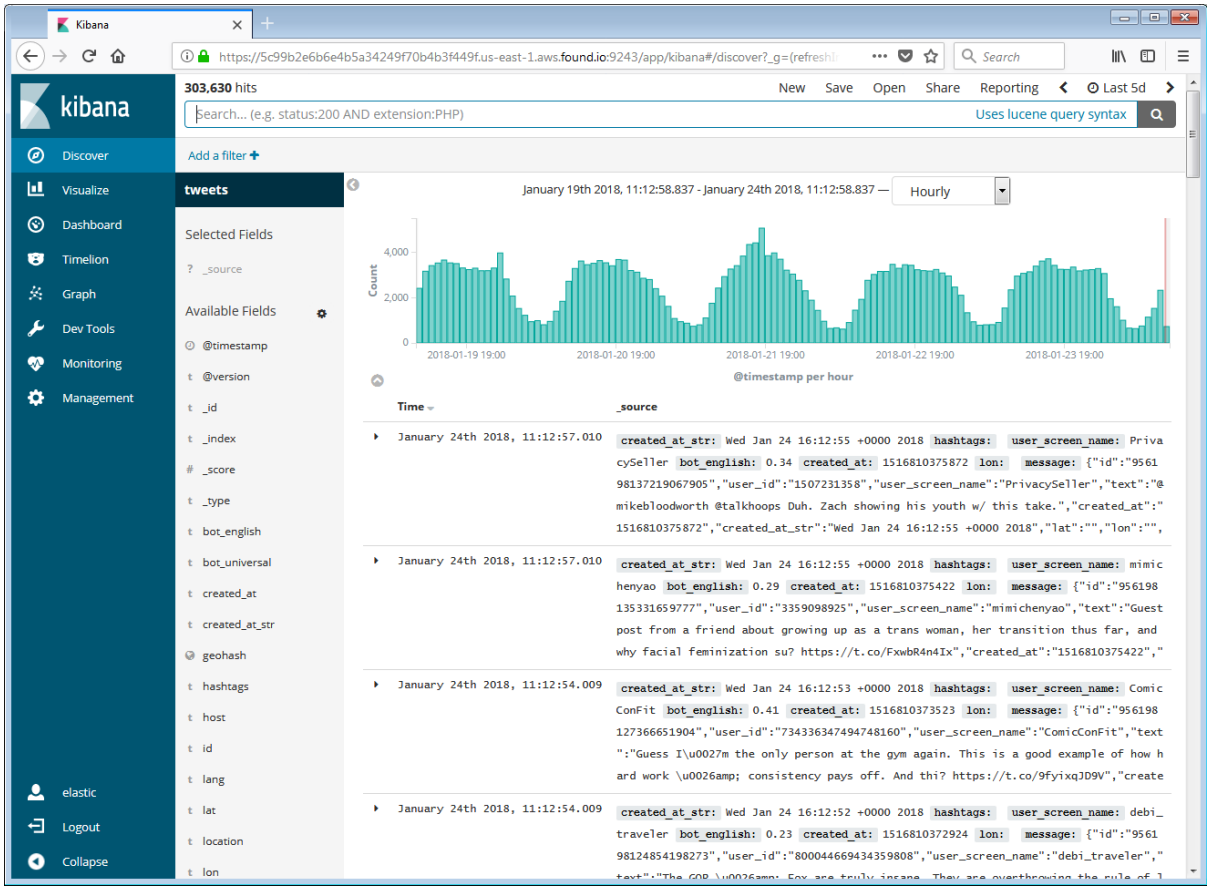

Figure 2. Kibana Main User Interface

During the development process, it was apparent that unexpected Tweets were being returned from the Twitter Streaming API. The only inputs passed during initialization are six latitude/longitude bounded, quadrangular, geospatial query areas corresponding to the metropolitan areas of interest. Nonetheless, many Tweets are returned that do not have corresponding latitude/longitude values but do have a pseudo-location populated. E.g. Tweet with id "952263569093025792" has no coordinates, but has "location: San Diego, CA". It appears that the Twitter Streaming API is not hard-lining the returned Tweets based on the requested geospatial borders but is returning all Tweets that are approximately in that area. For trend watching and sentiment analysis, these Tweets are useful; but if one is using the data to generate heatmaps representing population, these Tweets should be pre-emptively filtered (Standard Stream Parameters – Twitter Developers, n.d.). This can be accomplished by adjusting the query when creating the heatmap visualization to remove Tweets from outside of the bounding area of investigation.

For example, in the last 5 days from time of querying, 303,630 Tweets have been shipped to Elasticsearch/Kibana; using Query 1, only 48,004 of the Tweets have lat/lon populated; all queries can be found in Appendix B. It is also necessary to ensure that all the Tweets that have lat/lon populated are inside of the requested query areas. Per Query 2, only 14,873 Tweets are inside of the areas of interest. All further discussion will assume that the bounding filters for San Diego are applied to the data, unless otherwise specified.

Kibana offers the ability to create many types of visualizations: charts, data tables, gauges, coordinate maps, region maps, etc. In Figure 3, a panable/zoomable coordinate map, with Twitter data overlaid as a heatmap can be seen. Creation of this map is simple, essentially selecting which field in each Tweet is the 'geospatial' field. Note in the top right, the time period selector is still present for dynamic time period changes. Also, notice the filter for San Diego does not have negatives for the longitude values. Elasticsearch organizes the data lexicographically and does not understand negative numeric values as configured. There are workarounds, including padding numeric fields to a fixed width, and adding n/p symbol to delineate negative/positive respectively, but such functionality was unnecessary for this prototype (Numeric Range Searches in Lucene, 2009).

Kibana also offers integration to modern geospatial information systems (GIS) using a WMS interface. WMS is a standard for sending/displaying raster based geospatial data across different platforms, and is a common

functionality for popular GIS products, e.g. ESRI ArcGIS Server, GeoServer, MapServer, etc. A user can add their own geospatial data to a WMS server, then configure Kibana to leverage the layers. Many government and

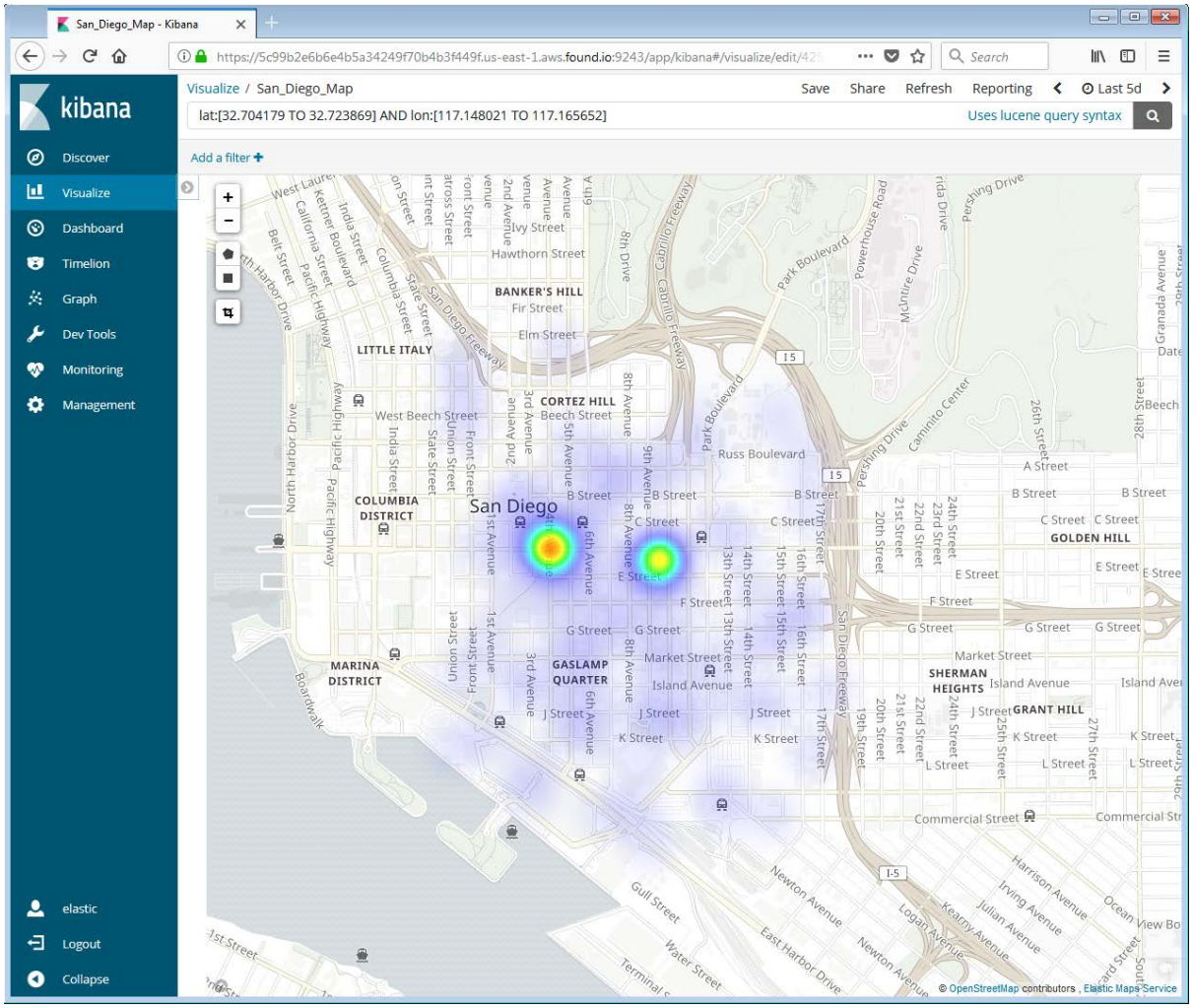

Figure 3. Kibana, Coordinate Map for San Diego, CA.

independent groups also publish GIS data for public consumption. In Figure 4, the San Diego heatmap is shown, overlaid on the National Flood Hazard Layer from the United States Federal Emergency Management Agency (FEMA). One can see that the bottom right of the map includes Twitter hotspots that are over a flood hazard zone.

The different charts and visualizations can be combined to create dashboards. Figure 5 shows a dashboard for San Diego with visualizations convenient for crisis management emergency responders.

- The heatmap of Tweets over the downtown area, allows a user to visualize groups of people, and where they are congregating.
- A chart of Tweet quantity over time provides insight into the social media patterns throughout the time period in question.
- A pie graph to show most popular users, who could be contacted to assist in providing information from “the ground”, or to directly distribute information to others.
- A pie graph to show most popular hashtags, provides insight into the content of what is being Tweeted about. A rise in crisis-related hashtags can warn about an impending/developing emergency in a geographic area.
- A hashtag cloud which allows immediate visual inspection of the popular hashtags in each Tweet.

An emergency responder could also leverage the Lucene query box and the time period selector to draw precise spatiotemporal constraints, e.g. over a city block in the last 3 hours, and have the dashboard immediately re-

populate with pertinent data. This functionality can support fluid and dynamic resource/response allocation (Shamoug et al., 2014) to where they are needed most. Kibana also implements responsive design (Gardner, 2011) in its interface, so it will automatically optimize for the physical device being leveraged, e.g. laptop, tablet, phone, etc.

Immediate inspection of the dashboard in Figure 5 shows many posts from "tmj*" with the hashtag cloud and pie graph showing many of the Tweets being pertinent to job seekers; direct inspection of the Twitter accounts shows they are likely automated. Using Query 3, the Botometer data is leveraged to remove bot-like data, and only display

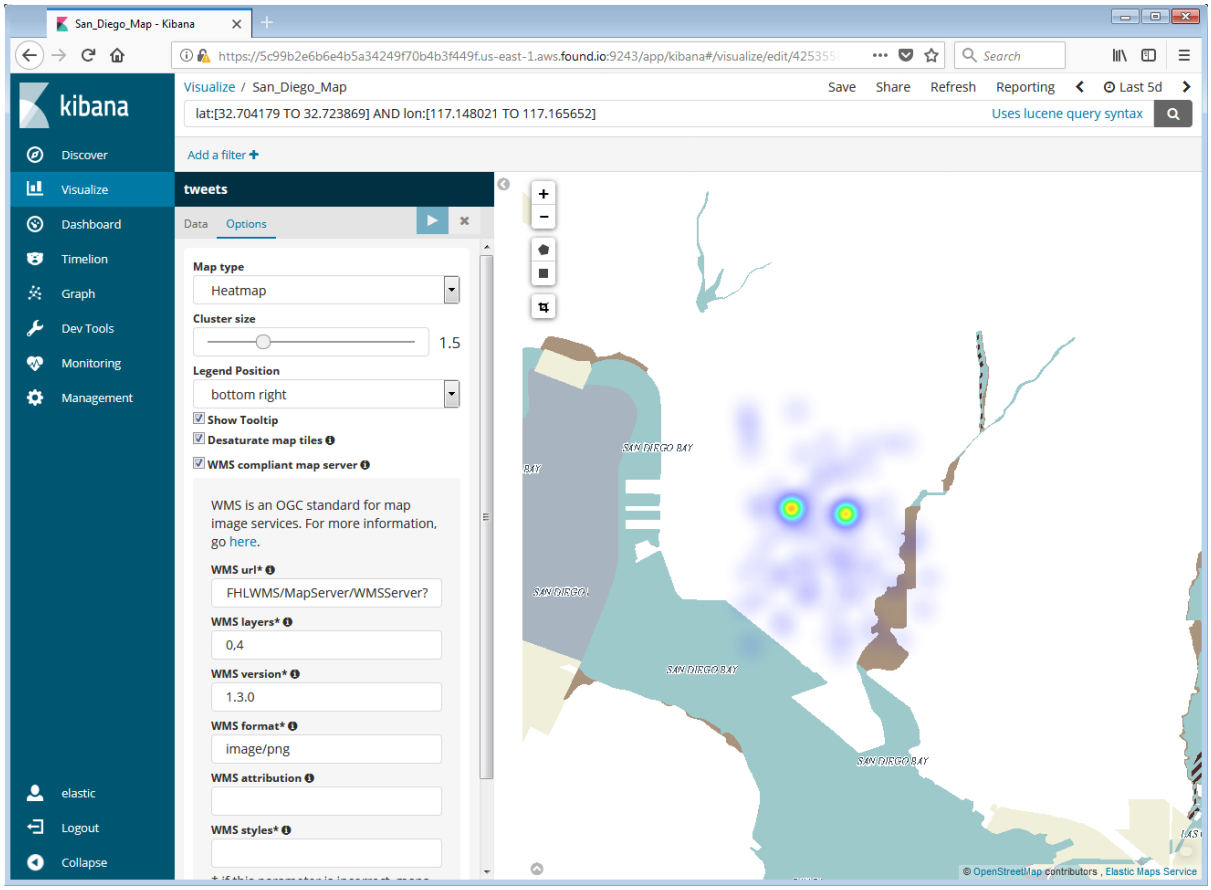

Figure 4. Kibana, Coordinate Map for San Diego, CA. with FEMA's National Flood Hazard Layer as WMS

Tweets that have corresponding bot-data of between 0.0 and 0.4. The interface immediately updates after entering the query, reflecting a decidedly more 'human' representation, which can be seen in Figure 6.

- Upon direct inspection, the Twitter usernames mostly correlate to real humans.
- Hashtags represent events that are occurring in the city from the play Hamilton visiting the San Diego Civic Center, to Monster Jam at Petco Park.
- One of the heatmap hotspots disappeared, the bot-created Tweets were likely using that location as a standard San Diego latitude and longitude. Filtering out this data can greatly improve human representation in the heatmaps.

This full dashboard for San Diego, using a Lucene query and Botometer data to remove Tweets from bot-like accounts, provides a flexible environment. As a crisis management and response actor, using the dashboard on any Internet-connected pervasive computing device can provide powerful near real-time insight into Twitter data, for a geographic space.

FOLLOW-ON WORK

This work presents a proof of concept and has many avenues for follow-on work.

- Actual deployment and use by a federal/state/tribal/local entity for the purposes of crisis preparation, management and/or response. Functional use will provide objective and necessary feedback about pain-points, utility and desired features.
- The current prototype only leverages the Twitter Streaming API for social media data. Integration with other services e.g. Panaramio, Facebook, etc. would provide ample opportunity for data fusion. The architecture of the system makes the Kibana interface amenable to integrating disparate data types. Once code is created to consume from a new service, a properly configured deployment of Logstash is all that is required to start shipping logfiles to an existing Elasticsearch/Kibana installation. Queries like the ones used here can be used to glean information and/or create visualizations.
- Investigation/creation/deployment of a service like Botometer for other social media services, using similar bot-detection heuristics. With a large percentage of Twitter accounts showing bot-like characteristics in this work's dataset, the necessity of filtering bots from VGI is apparent.

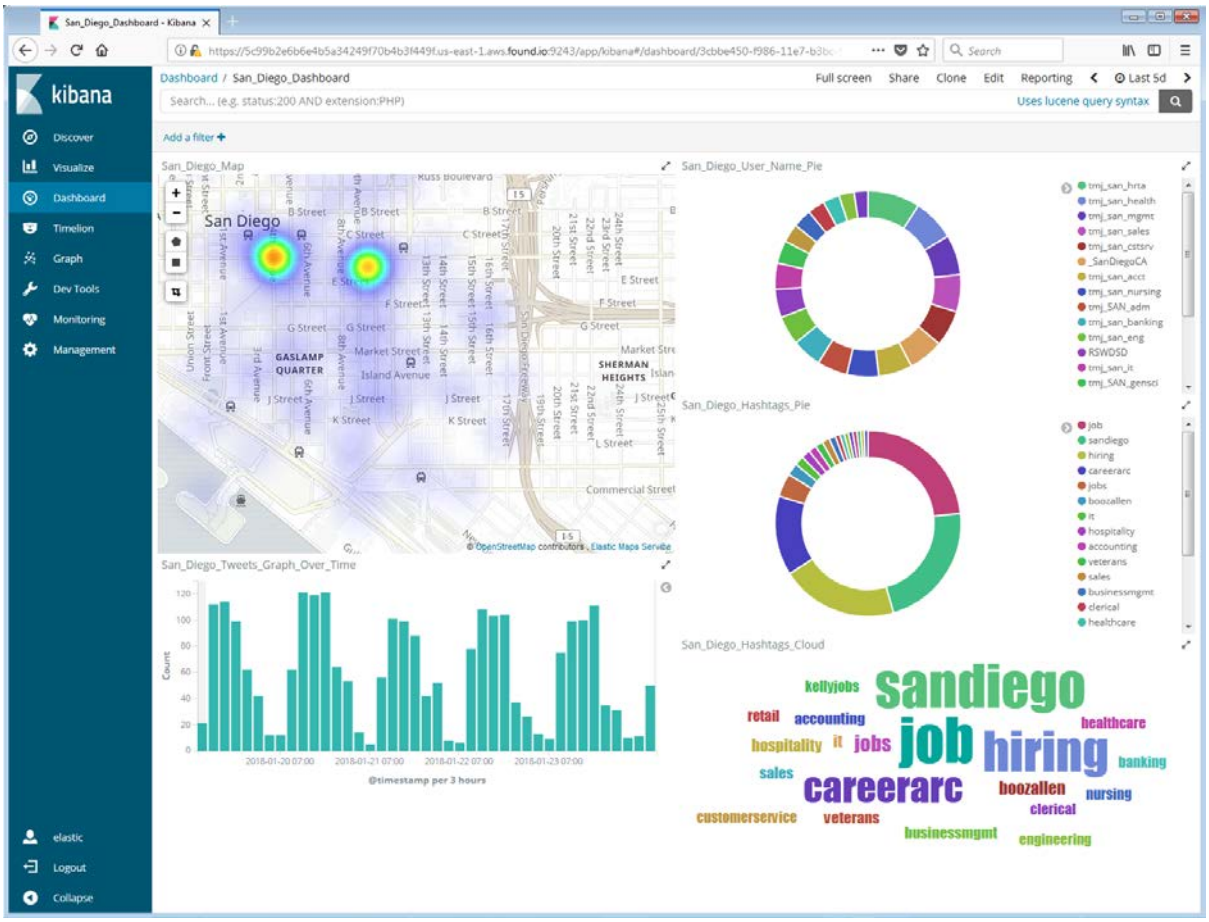

Figure 5. Kibana, Dashboard for San Diego, CA.

CONCLUSION

This work has shown an inexpensive, flexible, open-source software stack that can ingest Twitter API data and provide illumination into a population for a geographic location. The visualizations, dashboards and graphs are especially useful for supporting the full crisis lifecycle for emergency management, and can be deployed to the cloud, or run locally. Results are shown/discussed with full data, and again after filtering data from bot-like accounts, showing the importance of considering trustworthiness when using social media to represent a human population in near real-time. While the prototype does show promise, use by an existing emergency management entity (team/laboratory/agency/etc.) would be highly beneficial to understanding and improving its true utility.

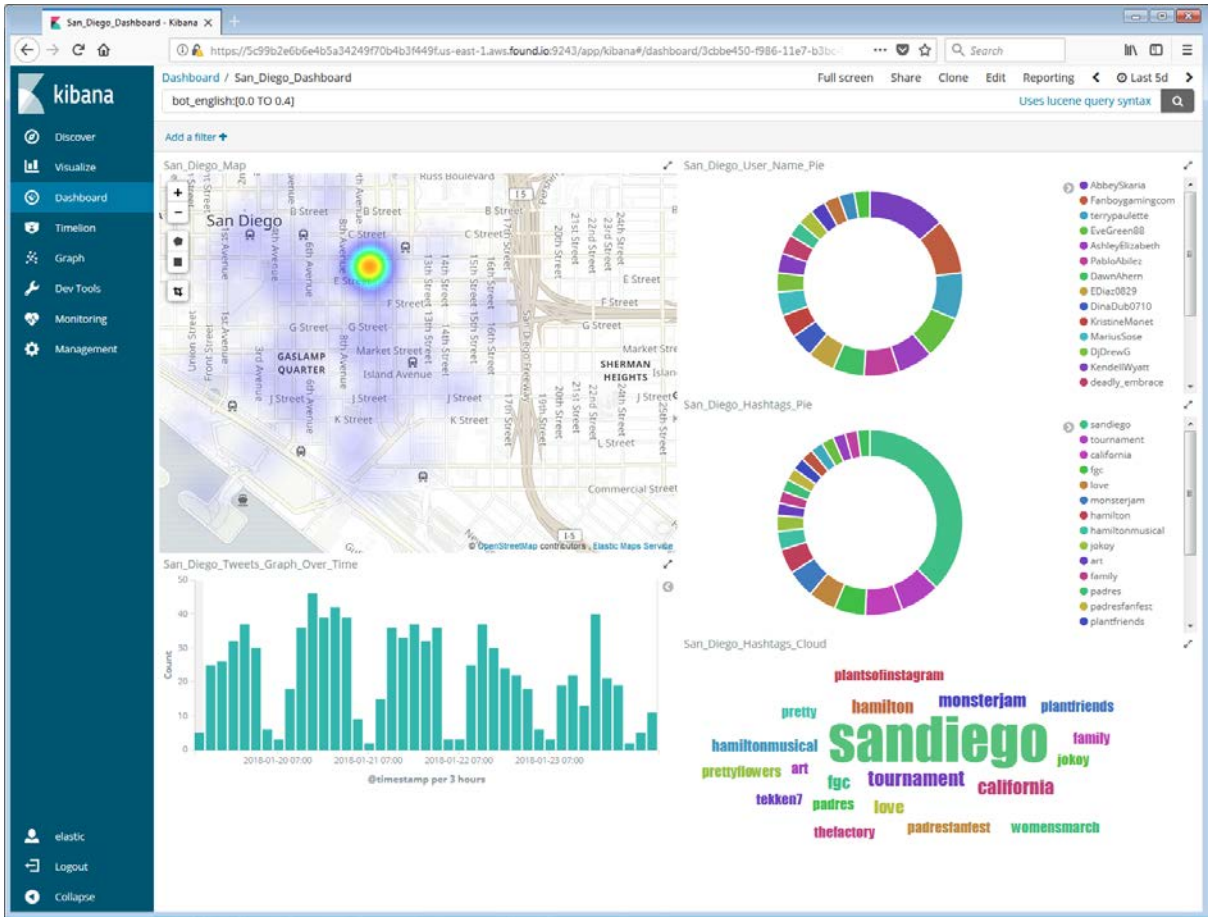

Figure 6. Kibana, Dashboard for San Diego, CA. with Bot-Like Tweets Removed

REFERENCES

Arnold, K., Gosling, J., & Holmes, D. (2005). The Java programming language. Addison Wesley Professional.

Aubrecht, C., Özceylan Aubrecht, D., Ungar, J., Freire, S., & Steinnocher, K. (2017). VGDI–Advancing the Concept: Volunteered Geo-Dynamic Information and its Benefits for Population Dynamics Modeling. Transactions in GIS, 21(2), 253-276.

Botometer API Documentation. (n.d.). Retrieved January 11, 2018 from https://market.mashape.com/OSoMe/botometer

Caragea, C., McNeese, N., Jaiswal, A., Traylor, G., Kim, H. W., Mitra, P., ... & Yen, J. (2011, May). Classifying text messages for the Haiti earthquake. In Proceedings of the 8th international conference on information systems for crisis response and management (ISCRAM2011).

Chu, Z., Gianvecchio, S., Wang, H., & Jajodia, S. (2010, December). Who is tweeting on Twitter: human, bot, or cyborg? In Proceedings of the 26th annual computer security applications conference (pp. 21-30). ACM.

Consuming streaming data --- Twitter Developers. (n.d.). Retrieved January 11, 2018 from https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data

Davis, C. A., Varol, O., Ferrara, E., Flammini, A. and Menczer, F. (2016). Botornot: A system to evaluate social bots. In Proceedings of the 25th International Conference Companion on World Wide Web, pages 273–274. International World Wide Web Conferences Steering Committee.

Elasticsearch: RESTful Distributed Search & Analytics | Elastic. (n.d.). Retrieved January 11, 2018 from https://www.elastic.co/products/elasticsearch

Elastic Cloud: Hosted and Managed Elasticsearch | Elastic. (n.d.). Retrieved January 11, 2018 from https://www.elastic.co/cloud

floatingsheep: DOLLY. Zook, M., Poorthuis, A., Morcos, M. Retrieved January 26, 2018 from

http://www.floatingsheep.org/p/dolly.html

Gardner, B. S. (2011). Responsive web design: Enriching the user experience. Sigma Journal: Inside the Digital Ecosystem, 11(1), 13-19.

GitHub - samueltoepke/Botometer4J. (2018). Retrieved January 11, 2018 from https://github.com/samueltoepke/Botometer4J/commits/master

GitHub - twitter/hbc. (2014). Retrieved January 11, 2018 from https://github.com/twitter/hbc

Goodchild, M. F. (2007). Citizens as sensors: web 2.0 and the volunteering of geographic information. GeoFocus. Revista Internacional de Ciencia y Tecnología de la Información Geográfica, (7), 8-10.

Gormley, C., & Tong, Z. (2015). Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. " O'Reilly Media, Inc.".

Halse, S. E., Tapia, A. H., Squicciarini, A. C., & Caragea, C. (2016, May). Tweet Factors Influencing Trust and Usefulness During Both Man-Made and Natural Disasters. In ISCRAM.

Java Platform, Enterprise Edition. (n.d.). Retrieved January 15, 2017 from http://www.oracle.com/technetwork/java/javaee/overview/index.html

Kibana: Explore, Visualize, Discover Data | Elastic. (n.d.). Retrieved January 11, 2018 from https://www.elastic.co/products/kibana

Li, J. P., Chen, R., Lee, J., & Rao, H. R. (2013). A case study of private–public collaboration for humanitarian free and open source disaster management software deployment. Decision Support Systems, 55(1), 1-11.

Logstash: Collect, Parse, Transform Logs | Elastic. (n.d.). Retrieved January 11, 2018 from https://www.elastic.co/products/logstash

McCandless, M., Hatcher, E., & Gospodnetic, O. (2010). *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co.

Merchant, R. M., Elmer, S., & Lurie, N. (2011). Integrating social media into emergency-preparedness efforts. New England Journal of Medicine, 365(4), 289-291.

Numeric Range Searches in Lucene. (2009). Retrieved January 14, 2018 from https://chase-seibert.github.io/blog/2009/09/11/numeric-range-searches-in-lucene.html

OSoMe: Home. (n.d.). Retrieved January 11, 2018 from https://osome.iuni.iu.edu/

Rate Limiting --- Twitter Developers. (n.d.). Retrieved January 11, 2018 from https://developer.twitter.com/en/docs/basics/rate-limiting

Shamoug, A., Juric, R., Paurobally, S. (2014). Semantic Representations of Actors and Resource Allocation through Reasoning in Humanitarian Crises. 47th Hawaii International Conference on System Sciences, 4169-4178.

Shelton, T., Poorthuis, A., Graham, M., & Zook, M. (2014). Mapping the data shadows of Hurricane Sandy: Uncovering the sociospatial dimensions of 'big data'. Geoforum, 52, 167-179.

Standard Stream Parameters – Twitter Developers. (n.d.). Retrieved March 5, 2018 from https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters

Sources. (n.d.). Retrieved January 26, 2018 from http://support.gnip.com/sources/#data-collector

Standard search API --- Twitter Developers. (n.d.). Retrieved January 11, 2018 from https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets

Tapia, A. H., Bajpai, K., Jansen, B. J., Yen, J., & Giles, L. (2011, May). Seeking the trustworthy tweet: Can microblogged data fit the information needs of disaster response and humanitarian relief organizations. In Proceedings of the 8th International ISCRAM Conference (pp. 1-10).

Toepke, S. L., & Starsman, R. S. (2015). Population Distribution Estimation of an Urban Area Using Crowd Sourced Data for Disaster Response. ISCRAM.

Varol, O., Ferrara, E., Davis, C. A., Menczer, F., and Flammini, A. (2017). Online human-bot interactions: Detection, estimation, and characterization. pages 280–289. AAAI Conference on Web and Social Media (ICWSM).

Veil, S. R., Buehner, T., & Palenchar, M. J. (2011). A work-in-process literature review: Incorporating social media in risk and crisis communication. Journal of contingencies and crisis management, 19(2), 110-122.

Yates, D., & Paquette, S. (2011). Emergency knowledge management and social media technologies: A case study of the 2010 Haitian earthquake. International journal of information management, 31(1), 6-13.

Zook, M., Graham, M., Shelton, T., Gorman, S. (2010). Volunteered geographic information and crowdsourcing disaster relief: a case study of the Haitian earthquake. World Medical & Health Policy, 2(2), 7-33.

APPENDIX A, PSEUDOCODE FOR THE JAVA CONSUMER

Algorithm 1 Twitter Streaming API Consumer Java Code

```
1: global waitFifteenMinutes ← false;
2: global dynamoDbTable;
3:
4: procedure MAIN
5:     queue localQueue;
6:     Create Connection to Twitter Streaming API with Geo Filters;
7:     Continually Place Tweets from Twitter Streaming API on localQueue;
8:     while 1 = 1 do
9:         if localQueue.size() > 0 then
10:            Tweet t ← localQueue.dequeue();
11:            NEWTHREAD(t);
12:
13: procedure NEWTHREAD(Tweet t)
14:    t.botData ← GETBOTDATAFROMDYNAMODB(t.accountId);
15:    if t.botData = NULL then
16:        if waitFifteenMinutes = true then
17:            if Now > 15 Minutes since (waitFifteenMinutes ← true) then
18:                waitFifteenMinutes ← false;
19:        else
20:            Timeline tL ← GETTIMELINEFROMTWITTER(t.accountId);
21:            Mentions m ← GETMENTIONSFROMTWITTER(t.accountId);
22:            BotData b ← MAKEBOTDATAREQTOBOTOMETER(t.userInfo, tL, m);
23:            t.botData ← b;
24:            UPDATEBOTDATAINDYNAMODB(t.accountId, t.botData);
25:    PRINTTOLOGASJSON(t);
26:
27: procedure GETBOTDATAFROMDYNAMODB(t.accountId)
28:    BotData b ← NULL;
29:    if dynamoDbTable.contains(t.accountId) then
30:        b ← dynamoDbTable.getBotData(t.accountId);
31:        if dynamoDbTable.getBotData(t.accountId).age() > 1 Week then
32:            dynamoDbTable.delete(t.accountId);
33:    return b;
34:
35: procedure UPDATEBOTDATAINDYNAMODB(t.accountId, t.botData)
36:    dynamoDbTable.insertOrOverwrite(t.accountId, t.botData, Now);
37:
38: procedure GETTIMELINEFROMTWITTER(t.accountId)
39:    Timeline tL ← MAKETIMELINEREQTOTWITTER(t.accountId);
40:    if Rate Limit Error = true then
41:        waitFifteenMinutes ← true;
42:    return tL;
43:
44: procedure GETMENTIONSFROMTWITTER(t.accountId)
45:    Mentions m ← MAKEMENTIONSREQTOTWITTER(t.accountId);
46:    if Rate Limit Error = true then
47:        waitFifteenMinutes ← true;
48:    return m;
49:
50: procedure PRINTTOLOGASJSON(t)
51:    JSON j ← t.toJSON();
52:    print(j);
```

APPENDIX B, QUERY STRINGS FOR ELASTICSEARCH/KIBANA

1. lat:[* TO *] AND lon:[* TO *]
2. (lat:[38.659419 TO 38.935678] AND lon:[8.993235 TO 9.253356]) OR (lat:[47.570555 TO 47.656297] AND lon:[122.269366 TO 122.423584]) OR (lat:[45.515404 TO 45.535094] AND lon:[122.6675029 TO 122.6851339]) OR (lat:[32.704179 TO 32.723869] AND lon:[117.148021 TO 117.165652]) OR (lat:[37.777479 TO 37.797169] AND lon:[122.387043 TO 122.404674]) OR (lat:[37.322802 TO 37.342492] AND lon:[121.876213 TO 121.893844])
3. bot_english:[0.0 TO 0.4]