INTEGRATING SOCIAL MEDIA WITH LAW ENFORCEMENT NEEDS

DANIEL VILLA, MOHAMMED ALI, ANKUR TEREDESAI

Abstract

Twitter, Facebook, Tumblr, and Instagram among others have made their collected social media publicly available to various degrees. Companies such as DataSift have taken advantage of this data by offering business intelligence into worldwide trends. While that is a good use of the data, there are many other ventures that stand to benefit from public, user generated content. In this paper we will outline our software that bridged a crucial gap between this wealth of knowledge and crime investigations.

1. Introduction

Law Enforcement relies on a number of software programs to keep the public safe. Most of them deal in the realm of analytics with a goal of predicting where crime is likely to occur based on historical trends. They analyze months if not years of police reports and have shown some level of success[1]. Another up and coming technology integration idea is known as Text-to-911[2]. While data analytics intends to support predictive policing, Text-to-911 intends to equip victims with tools to protect against offenders. These technological integrations reflect concepts presented in what is commonly called the Crime Triangle[3].

As shown in Figure 1, the triangle outlines how *controllers* in the light blue influence *actors* in the dark blue located on the same side of the triangle.

- Handlers influence offenders,
- Guardians influence victims,
- Managers influence places.

Every person has some set of tools. Every crime involves one or more offenders, victims, and places respectively. A potential problem arises when offenders and victims are at the same place, especially when controllers are absent from the scene. As The Center for Problem-Oriented Policing website states, "The effectiveness of the people involved will depend, in part on the tools they have available" [3]. Social media can be used as a tool, according to this theory.

Figure 1

Social media often has three key pieces of information: some text or picture, a user name, and a location. This information if generated around a crime scene, could help investigators to have a better idea of what happened at a crime scene. Most social media apps that are well used provide ways to access and query this data.

This paper will examine our Windows Form Application, Social Media for Law Enforcement (SM4LE) that provides the ability to query, organize, store, stream, and visualize social media. This software has begun to equip guardians, victims, and managers with a tool that can help reduce loss and prevent crime. In section 2 we will cover how we managed the project and ensured industry usability. Section 3 will cover the functionality of the system in detail. Section 4 concludes the paper referencing future work and a few acknowledgements. Section 5 lists the references.

2. Project Management Methodology

In order to ensure that our efforts on the product held value, we established working relationships with Trina Cook and Tony Berger. In 2012, Trina became a Certified Law Enforcement Analyst and currently works directly with the Tukwila Police Department providing predictive policing services. Tony is currently a program manager at

Washington State's first Real-Time Crime Center (known as LARIAT). Over the three-month duration of this project we met with Tony and Trina on a weekly basis to check the usability of the software as well as receive feature requests.

All requests would get inserted to Trello, a project management tool. Each week, after meeting with our customers, the team would prioritize requests in Trello based on customer interest and software needs. This produced short feedback loops and short development times ensuring that development efforts were driven from customer needs.

3 System Design

3.1 Overview

We pursued two popular data sources to integrate into our application; Twitter and Facebook. Each web app provides a web service API through which any application can send a HTTP request and receive a JSON or XML response. Twitter became our sole provider after discovering Facebook's immature API and privacy polices. In order to access Facebook data a developer has to sign in with an account. This would likely be his personal account. Searches performed are limited to his friends and their friend's data. Thus, the selection pool was hardly broad enough for practical uses. In conjunction, the location based search functions provided by the Graph API turned out to be under developed. Data returned did not seem relevant to the search parameters and none of it seemed to have location data attached. This provider constraint on our system is not to be overlooked. The fidelity of the data and its providers directly impacts the effectiveness of the system.

In contrast, many social media providers have opt-out policies like that of Tumblr; "By default, all sharing through the Services is public, and when you provide us with content it is published so that anyone can view it."[4] Twitter follows suit.

Once we signed up our app on Twitter we received credentials used to make API calls. A .NET Library called Linq2Twitter allowed us to interface with the API by using C# objects. The library would use our credentials, build the HTTP request, parse the returned JSON, and populate the returned C# objects.

Figure 2 displays the simplest layout of the architecture. The user is provided with various ways of inputting a location, radius, and time window (Step 1). If the location is in the form of an address our system will take Step 2 which geocodes the information into a latitude and longitude. After all inputs have been properly translated, we take Step 3 which calls the Twitter search function. Step 4 indicates a crucial step where we write the mapped Address-to-Geocode data and all search results to disk. Finally, the results are sent back the GUI for display in Step 5.

Figure 3 shows the start up view of the application.

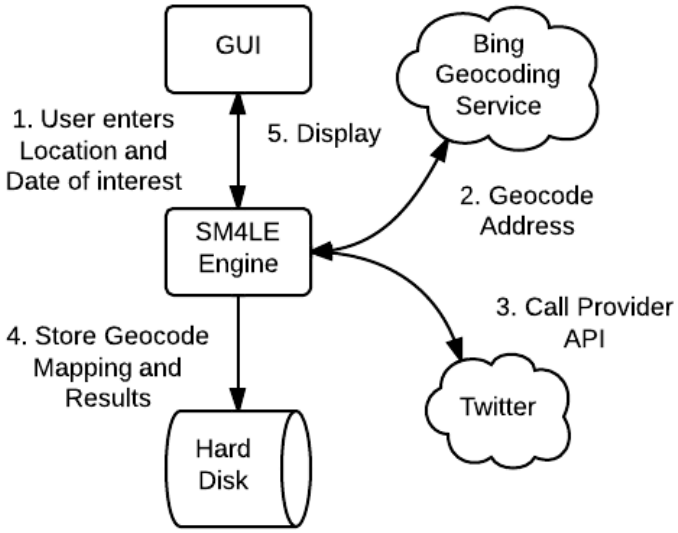

Figure 2

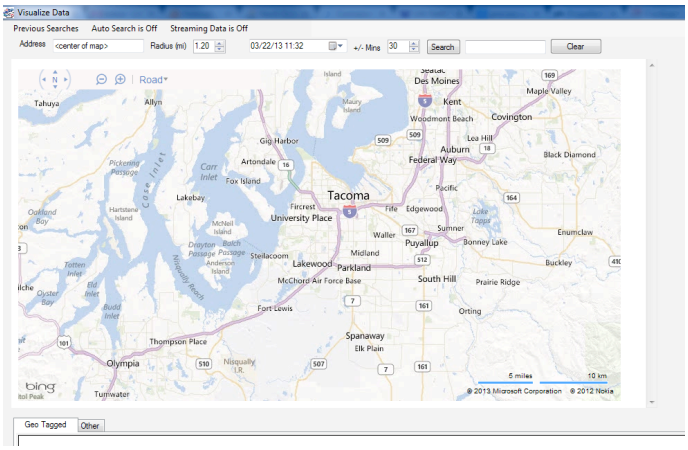

Figure 3

3.2 Data Source Implementation

In order to describe the mechanics of the app, we will describe the use case of a search.

Collect Query Parameters. The customer wanted various ways to search for data, thus we developed four different ways to perform searches. The first is a classic method. The GUI holds a set of text boxes and radials to allow the user to enter an address, a date and time, a radius (mi), and a minute amount (Fig. 3). The minutes define the window before and after the time given. A second method is to leave the address text box at the default value of "<center of map>". If this method is used, our application will pull the latitude/longitude location from the center of the map display while using the rest of the parameters as they are entered. This circumvents the need to geocode as described in Fig 4. Alternatively, users can Shift + Click anywhere in the map

view to collect the needed coordinates. Finally, if the user wants to monitor an area, they can toggle the Auto Search menu button. This will perform the previous search every 15 minutes allowing the display to auto update with each search.

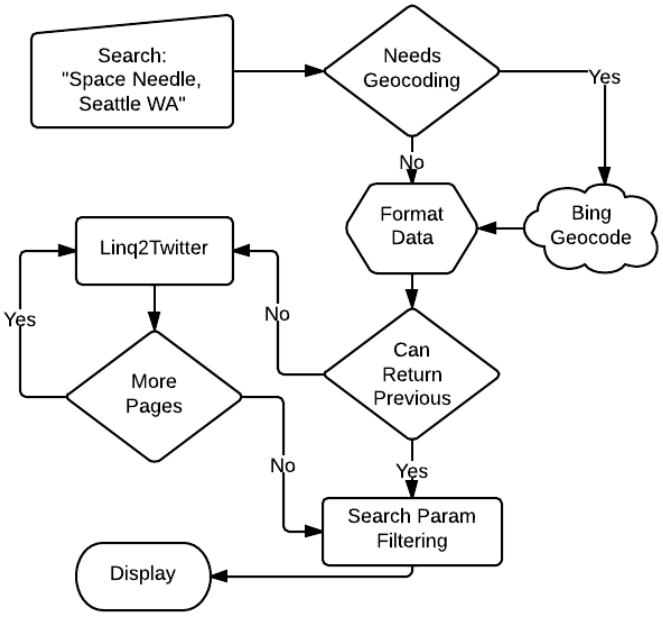

Figure 4

Geocoding. This step is necessary for textual addresses because most, if not all API's that provide geographical queries require latitude and longitude location definitions. We used Bing Geocoding services which does not hold restrictions on how we display the geocoded data. It is important to note that all geocode service providers mention the resource intensity of geocoding algorithms, thus, each address entered by the user is stored in a map fashion with the address as the key, and the coded coordinates as the value. This data is kept in memory at runtime and in a flat file for persistence.

Smart Queries. Twitter limits the number of searches a particular app can perform based on search complexity and frequency. This ensures their services remain stable for all users. Our application is attempting to gather as much data as possible, so performing a search 5 miles around Seattle between 8 and 9 pm on a Friday night will likely come up with thousands of points. We will talk shortly about why we can get thousands of points and why they are not all explicitly geotagged. These thousands of points come in batches of 100 maximum per page, and each new page request counts as 1 against our rate limits. In a typical user search, we end up needing to query Twitter more than 15 times as depicted in the "More Pages" decision on Fig 4. To complicate matters, the Twitter API only accepts date, not time. For example, if searching for an event at 11 am yesterday, Twitter would starts a 11:59:59 PM and work chronologically backwards. We could easily need to page 50 times before getting back to 11 am. Doing this search just a few times would lock out our app for at least an hour. To address this problem, we implemented two optimizations: *windowing* and *appending*.

Windowing occurs in relation to time and radius. If a user just queried a time period of 8pm to 9pm and are now asking for data between 8:30pm to 9pm, we can safely return the same results as the last query as long as all other query parameters have remained the same. We leave it up the UI code to filter data that is outside the requested window. This also works if the radius is less than or equal to the previous radius and the time window is less than or equal to the previous window, assuming the address has not changed. We found this to happen quite often in practice. Users do not expect the initial large quantity of data and tend to trim their parameters after an initial search. The other optimization is called *appending*. Our application can perform the same search every X number of minutes. When doing this, we may want to always look at a 30-minute window updating every 5 minutes. It is most efficient to only get the latest data to increase response time and decrease API calls. To do this, Twitter provides the Since_Status_ID and Until_Status_ID (not official parameter names). When searching for a 30 min window, we first check that the radius and address are the same. If so, we compare the previous searches oldest and newest Tweet IDs. If we see that only the latest X minutes are missing then we will set the Since_Status_ID parameter in order to optimize our API query. Although we will address streaming enhancements in Section 4, note here that this solves a problem that streaming cannot. Normal user operations could produce this scenario when asking for any data that overlaps their last time query.

Twitter Search API. To perform the query the Linq2Twitter library allows us to simply assign our values to the correct API parameters and expect back a list of Status objects. Twitter's search function demands a string of keywords. For this parameter we simply pass an empty string since we want to do the keyword filtering client side. This is another factor that causes our result set to be larger than normal.

As stated, Twitter provides a maximum result set of 100 posts per query. To page through the data we repeat the query while updating the Until_Status_ID variable which gathers older data with each iteration. Once a page is returned with less than 100 posts (our configured max) we know there are no more matches, so we stop paging and return the cumulative data collected. We also have a safeguard limit of 15 pages per search so as not to abuse the API when extremely large time/radius windows are requested by users. Pertinent attributes of each data

point are appended to a CSV file for later investigations. Posts contain not only text but also a plethora of information such as screen name, number of retweets, geolocation information, location, home city, and photo URL if any. Although we have specified exact coordinates in the search parameters Twitter handles location data in a peculiar way. Each post has three location attributes: the coordinates, the location, and the home city. Coordinates are available if the device that tweeted was GPS enabled. Location can be populated if a person "checked-in" or mentions a place in his tweet. Finally, everyone has a home city and by default that information is public. If Twitter finds a tweet that is *associated* with the provided coordinates and radius then that tweet will include it as a relative data point. For example, if my hometown is Seattle but I'm tweeting from New York, my tweet may be picked up in a search around the Space Needle. As mentioned, the engine grabs everything it can and lets the UI do the filtering.

A final note on the Twitter API is that no guarantee is made on retrieval of data more than 7 days old. Because of the amount of data collected every day, it would simply take too much space to store it all.

Displaying the Results. After acquiring all of the data, we take one more action before displaying to the user. If the returned data was actually geotagged, that is, if it has a non-zero latitude and longitude, then it will get sent to the map as well as the first tab of the data grid. All remaining data is left off of the map and shown in a secondary tab.

Twitter Streaming API. SM4LE also demonstrates the ability to utilize Twitter's Streaming API. One can monitor the public timeline or follow sets of users, among other possibilities. Twitter's Timeline is their implementation of the "fire hose", the ability to stream all new data as it is generated in real time. In our case, we follow 45 Twitter users collectively known as Tweet-by-Beat [5]. These feeds report crime as it happens around the Seattle area. Each Tweet follows a specific format such as: "Beat:B1, CRIME_TYPE at 13XX BLOCK OF STREET NW reported on dd/mm/yyyy mm:ss PM Call# 12345678". By providing these user IDs to Linq2Twitter along with a callback method, Twitter streams all new posts straight to our system. Our call back method uses a regular expression to extract the address and with the currently entered radius and window, calls the search API. With this functionality users do not even have to interact with our app. It silently collects social data for later use. This also shows the potential for other government entities to build on our system to gain social media benefits. It also shows how social media providers could be linked by searching provider B based on provider A's activity. Though streaming data from public accounts like Tweet-By-Beats is not in violation of any laws, there are considerations when dealing with streaming. 28 Code of Federal Regulations Part 23 (CFR 23) speaks in great detail to the exact laws of gathering criminal intelligence on individuals. See [6] for further details and possible pitfalls when collecting data.

3.3 User Interface Demonstration

In this section, we will take a look at the user interface, demonstrating how to use the software.

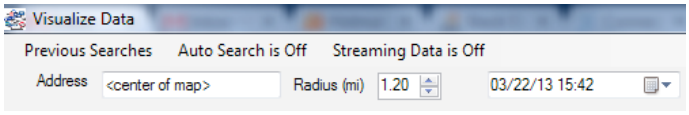

Figure 5

Users can set search data in the left half of the menu bar as shown. On start up, the address box contains "<center of map>" so that users can visually look at a location and click the Search button to search on area at the center of the map. They also have the menu options to auto search as well as stream data from the Tweet-By-Beat users.

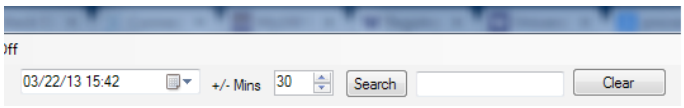

Figure 6

The right half contains the window of time to search and the search button. Users can also hit the Enter button after typing in an address to initiate a search. The text box on the far right is the filter box. Once data is returned, users can enter text into this box and hit Enter in order to filter the data based on textual status matches. The text entered into the box is converted to a regular expression like this: "*(t|T)(e|E)(x|X)(t|T)*" so that it is case insensitive, matching any part of a status.

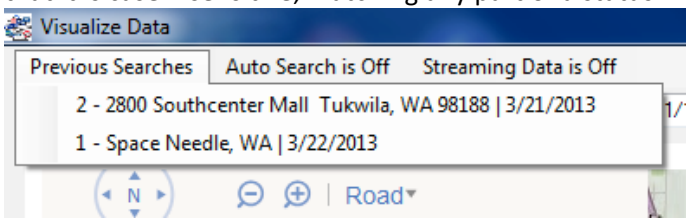

Figure 7

Fig. 7 shows the ability to research previous searches. This is currently not persistent but rather per user session.

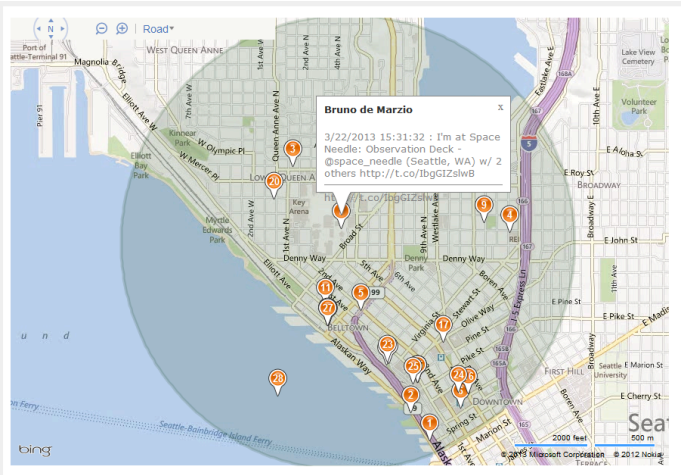

Figure 8

Fig. 8 shows how data is displayed on the map. (Refer to Fig. 3 for a full snapshot.) Hovering over a pushpin will bring up a floating text box with the text information found in the social media point. The shaded circle is showing the area searched.

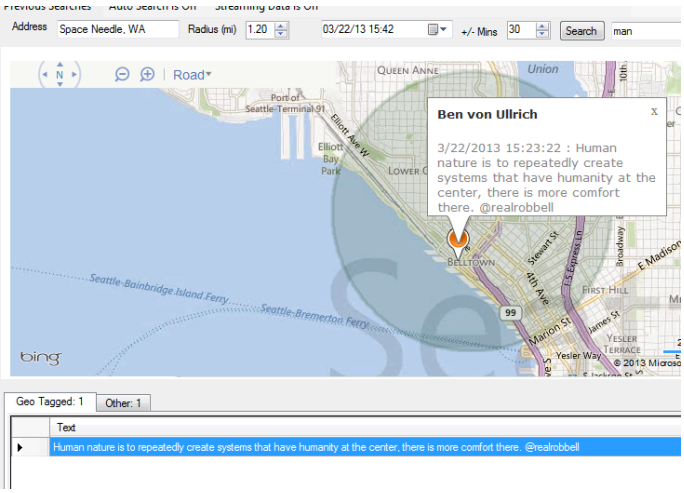

Figure 9

By entering in the text "man" into the filtered text box and hitting Enter, we see only the matching data. In Fig. 9 "human" is a match for "man" and thus is shown on the map. We can also see that there are a total of 2 matches, one with geographical coordinates, and associated point without coordinates. By removing the filtered text and hitting Enter again, we see the full list of data that was returned by our initial query. There are 28 points with coordinates and 18 without, as show in Fig. 10.

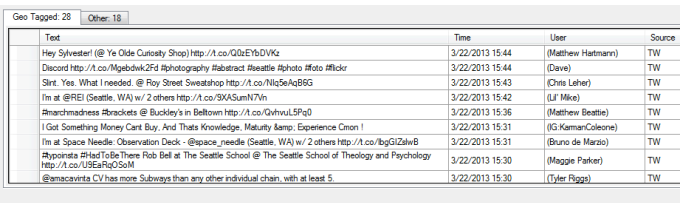

Figure 10

In Fig. 10 we show just a few of the attributes that is associated with each point in the grid view. One could easily add the number of retweets or other location information. Figure 11 shows a valuable feature where data selected in the Geo Tagged tab is highlighted on the map. It is brought to the front (since some data complete covers other data), allowing the user to hover over the pushpin.

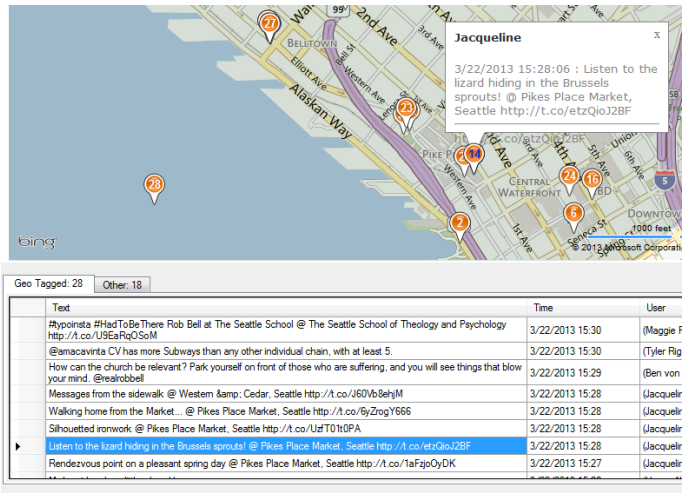

Figure 11

Finally on the left hand side is the streaming gallery. When streaming data is turned on new posts are displayed from top to bottom along with the photo of the owning user.

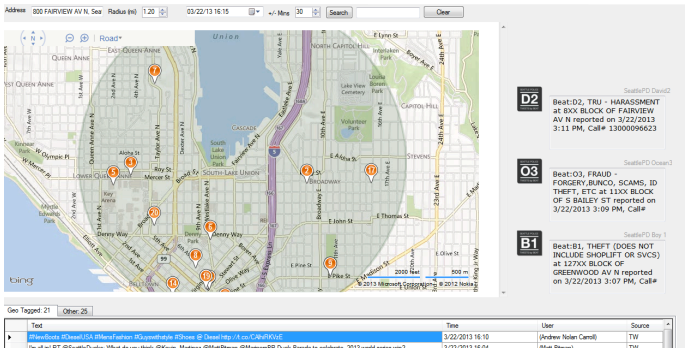

Figure 12

Other features that we have already described are available to search in different ways and efficiently gather the desired data.

4. Conclusion

SM4LE is feature rich so that users can easily interrogate the social media scene. Trina Cook wrote, "Dan and his team delivered a usable product. After downloading the program, it was simple to put to use. I can especially see using it for querying of in-progress or recently-occurred incidents and harnessing the real-time capabilities of the program. Being able to either input an address or just pan to an area on the map is very useful, as well as the

radius and date/time selections. The simple and clear format is exactly what is needed in law enforcement software."

There are several enhancements that could be made to our software. Firstly, auto searching might be more effective to make use of the streaming API. In this way we could monitor an area without risking any rate limits. Secondly, we could store the mapped address to coordinates and search results into respective tables in a database to increase search and storage efficiency. Thirdly, we could integrate Tumblr with our application and enable the ability to view linked pictures from data providers in the map view. Another idea for future improvements is to move away from neutral providers altogether and build a mobile reporting app. In this way the community could be directly involved in reporting data they believe to be beneficial to crime investigations. The SM4LE tool can help reduce loss and prevent crime by helping in criminal investigations and getting the offenders off the streets. If the system gains popularity, criminals, victims, and managers will all know that social media is at every ones fingertips and can be used to document and expose the truth.

I would like to sincerely thank Robby Oesch for working with me in developing this application as well as Trina Cook and Tony Berger for giving us valuable customer feedback along the way. Thanks to Dr. Ankur Teredesai and Dr. Mohammed Ali who also supported our endeavors.

5. References

1. Casady, Tom. "Police Legitimacy and Predictive Policing" *Geography & Public Saftey*. N.p. Web. Mar. 2011.

2. "What you need to know about Text-to-911" *Federal Communications Commission*. N.p. 22 Jan. 2013. Web. 20 Mar. 2013.

3. "A Theory of Crime Problems" *Center for Problem-Oriented Policing*. N.p. Web. 21 Mar. 2013

4. "Privacy Policy" *Tumblr*. N.p. 22 Mar. 2012 Web. 22 Mar. 2013

5. "Tweets-By-Beat" Seattle.gov. N.p Web. 22 Mar. 2013

6. "Criminal Intelligence Systems Operating Policies (28 CFR Part 23)" *Institude for Intergovernmental Research*. N.p. Web. 22 Mar. 2013