Time to Check the Tweets: Harnessing Twitter as a Location-indexed Source of Big Geodata

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Arts in the Graduate School of The Ohio State University

By

Kiril Vatev, B.S.

Graduate Program in Geography

The Ohio State University

2013

Thesis Committee:

Karl Ola Ahlqvist, Advisor

Ningchuan Xiao
Abstract

Twitter, a popular online social network based on the microblogging format, is an important and significant source of both locational and archival data. It lends to a wide range of topics, from predicting the results of political elections, analysis of natural disasters, and tracking of disease epidemics, to studying political movements and protests, notably in cases of protest organization in Moldova and Iran, and recent events in the Arab Spring.

However, many obstacles and barriers to entry are posed by Twitter’s system. Twitter has two main ways of obtaining data with different API requirements, and both returning different formats of data. System-wide authentication further complicates the data collection process. This research is based in creating an open-source software package to overcome the difficulties and barriers to obtaining a Twitter dataset, for both programming aficionados and those only interested in the data. The goals of the software are to ease the collection, storage, and future access to those interesting and important datasets. The resulting software package provides ways to drill down into the data through advanced queries not available from Twitter, and retains the data for extended analysis and reference over time. The software provides a friendly user interface, as well as plug-in capabilities for programmers.
Vita

2011 ........................................... Bachelor’s of Science in Geography, specialization in GIS, The Ohio State University


April 2012 ............................... “Green Revolution Geogame Demo.” Innovate OSU, Steal My Idea Showcase. Columbus, OH.


June 2012 to August 2012 .......... Esri: Applications Prototype Lab Intern

May 2011 to present ............... Research Assistant and Software Developer – The GeoGame Project, The Ohio State University: Department of Geography

Fields of Study

Major Field: Geography
# Table of Contents

Abstract .......................................................................................................................................... ii

Vita ................................................................................................................................................. iii

List of Tables .................................................................................................................................. v

List of Figures ................................................................................................................................. vi

Introduction ................................................................................................................................... 1

Background ................................................................................................................................... 3
  Understanding Twitter ................................................................. 4
  Sample previous work ............................................................... 7
  Geocoding ................................................................................. 11

Software Architecture .................................................................................................................. 13
  Data format ................................................................................ 13
  The development environment .............................................. 14
  The database .......................................................................... 17
  Geocoding service .................................................................... 19
  Other considerations ............................................................. 20

Software Implementation ............................................................................................................ 22

Sample Dataset and Testing ......................................................................................................... 25

Discussion ..................................................................................................................................... 30

Conclusion and Distribution ........................................................................................................ 33

References ................................................................................................................................... 34

Appendix A: The Manual ............................................................................................................... 38

iv
List of Tables

Table 1: Supervised location categories ................................................................. 26

Table 2: Supervised location counts ..................................................................... 27

Table 3: Unsupervised location categories ............................................................. 28

Table 4: Unsupervised location counts ................................................................. 28

Table 5: Classification problem areas ................................................................. 29
List of Figures

Figure 1: Software architecture .................................................................................................... 21

Figure 2: The task creation form ................................................................................................. 43

Figure 3: Task data page .............................................................................................................. 45
Introduction

Locational data is no longer confined to the ubiquitous shape file (.shp), and today can be gathered from countless sources and data services: RSS feeds, social media, and Open 311 feeds to name a few. With over one billion GPS-equipped smartphones in use, a number predicted to double in the next three years (Strategy Analytics 2012), and advancements in Internet infrastructure, accurately placed geographic data can be produced more easily than ever before. However, putting this data to full use in geographic research is still problematic for a number of reasons. The data can come in any format that the data service deems appropriate, with any number of barriers to entry, and are almost always inaccessible to conventional geographic information systems (GIS), such as ArcMap. To further complicate data collection, location is often not standardized within a single data service – let alone across the various services – and in itself presents a challenge to data collection, often resulting in the necessary discarding of large amounts of data – at times as high as 99%, as I have found in my preliminary work.

Twitter, in particular, is a major source of interesting and relevant geographic information. Twitter is a rapidly growing social network site created in 2006. It is currently the fourth largest social network globally (Global Wed Index 2013), and the largest with an open access. Twitter adopts – and arguably created – the microblogging format. The network is based entirely on users sharing 140 character messages, called tweets, openly with the public (Gonzalez et al. 2011). These messages can spread quickly through the general public, and as such, Twitter is often regarded as “electronic word of mouth,” (Tumasjan et al. 2011), consisting of small bits of information that are important to and reflective of the opinions of the author.

With the wide array of data Twitter can provide, and in its wide array of formats, Gao et al. (2011) argue that there is no standardized system to collect such data, and not even a comprehensible one. Past efforts to do so in academia have been ad-hoc, and often the most
problematic part of research. However, data can be used on its own, or as a complement to other sources, to answer many questions raised by geographers.

The first part of my thesis will focus on defining and examining the uses and limitations of Twitter as they exist today, while the second will be to develop a software package suitable for research purposes. My research collects and assembles the latest techniques in geographic data collection and location extrapolation and geocoding, and combines them in a standardized tool for collection and distribution of geographic data. Importance will be placed not only on the functionality and optimization of the software, but also on the user experience and software friendliness. The resulting software package, called Point Bank, provides ways to drill down into the data through advanced queries not available from Twitter, and retains the data for extended analysis over time. The software provides plug-in capabilities for programmers, who would be able to focus on processing data, without a concern for the layers of code necessary to interface with Twitter, database software, and the computer hardware.
Background

This project was envisioned during my work for a major GIS company, where some aspects of Twitter’s Terms Of Service agreement, or TOS, were revealed. Twitter, for the benefit of its data partners, protects its network from commercialization by third parties, and prohibits the development of software packages that rely on its data as an industry commercial product. Such uses of the network are permitted, however, for the purposes of research and creating derived analysis, though tools which ease that work cannot be commercialized and cannot possess any inherent monetary value. Due to these industry restrictions, such tools must be created in academia and for the benefit of academia.

Twitter is defined by its real-time nature. Data is available immediately after creation, but historically is only accessible for about one week. Natural disasters like storms and tsunamis only presents days of advance notice, and earthquakes present no notice at all. Political actions and protests, as well as disease epidemics, are often well on their way by the time they are detected. This leaves researchers little to no time to create a data collection plan. Dealing with this immediacy, it is impossible to begin developing a tool to collect and analyze the data after the research questions are perceived. To address this time sensitivity, software has to exist to handle the immediate collection of data, in a form that will be easily deployable at a moment’s notice.

Twitter is a very important archival source of social network interactions in large part due to its openness (Schoneboom 2011, Gladstone 2013). Information is shared in public, and no special permissions are required to access the opinions, thoughts, and ideas of those using the network – this idea of openness is embedded in the structure of the network (Farnsworth 2010). The two larger networks and competitors to Twitter, Facebook and Google+ (read as Google Plus), do not provide open access to user posts, and are instead largely based on the friend system, where
two users have to mutually recognize a relationship before they can see the posts the other makes. This makes it very difficult, even impossible, for a researcher to observe the natural communication of the members of these other networks. More social networks are embracing Twitter’s ideas of openness, such as Instagram and App.net, but these networks are still very young and have more of a niche user group, therefore presenting less data-gathering potential than Twitter.

The remainder of this background will be split up into three distinct sections, addressing the value of Twitter as a research tool, the nature of the social network itself, current and potential example uses, and a brief overlook of the validity of the data. I will also look at the nature of geocoding and geolocating data from the Internet.

**Understanding Twitter**

Twitter is currently the eleventh most visited website in the world, with Facebook and YouTube as the only social networking sites ranking above it (Alexa 2013). It is the fourth largest social networking site in the world in terms of active users (Global Web Index 2013). Like most other websites, Twitter does not release usage statistics. However, there are various different analyses and reports created by notable media and advertising companies. Twitter is estimated to have over 500 million users (Semiocast 2012), who post upwards of 340 million tweets per day (Twitter 2012). Geographically, these users are spread all over the globe. Although the numbers are contested, the United States has the highest percentage of users. Alexa (2013) pegs the U.S. userbase at 21%, Semiocast (2012) at 28%, and Orchid (2011) as high as 33%. Arguably, this could also be the result of higher global rates adoption of the social networking site. Other countries with a large user base are the United Kingdom, India, Indonesia, Japan, Germany, Brazil, Mexico, Spain, Russia, and Canada (Alexa 2013, Semiocast 2012, Orchid 2011). It is notable that the numbers in these reports do not account for differing population sizes in these countries.

Overall, Twitter accounts for 8% of the global Internet users (Pew Internet 2010), with age groups between 18 and 34 being slightly higher represented than the rest of the Internet (Alexa
Twitter also has a higher representation of users with postgraduate degrees (Alexa 2013). Overall, women are slightly more prevalent on Twitter (Beevolve 2012), and traffic to the site originates from schools at a higher rate than from homes or the workplace (Alexa 2013). The findings of Pew Internet (2010) also suggest that African-American and Latino minorities in the United States are more likely to be using Twitter than white Internet users.

In 2010, Twitter announced that 37% of users visit and contribute tweets from their mobile phones. This number has been estimated to have increased by 182% the following year (Orchid 2011), suggesting that tweets coming from GPS equipped smartphones is drastically increasing.

The main focus of Twitter is the tweet. A tweet is a 140 character long text message, that can also include links, images, message recipients, hashtags, and geotags. Aside from the obvious case of a simple, short message, links for content anywhere on the Internet can be included in the tweet, along with a message discussing the ideas or presenting an opinion on the linked content.

Images can also be embedded into tweets, allowing for visual content to be included in the context of the message. These are often used to share cultural event or phenomena, everyday objects or scenes, political events, natural disaster situations, etc.

Message recipients can be specified inside the tweet, allowing for the existence of two-way communication among the users of the social network (Gao et al. 2011). This is done by using the at symbol (@), followed by the username of the recipient. Any user can send a tweet to any other user, and any user can reply to the tweet, thereby creating a conversation. This is stored in the social network structure, and conversation can be viewed as such. This allows the unique opportunity to collect large amounts of public conversations and study the locational connections made on the web.

The hashtag is another feature of Twitter, providing some context to a tweet. A hashtag is a user defined topic, displayed as a single word preceded by the hash symbol (#). This allows a large group of users to talk about or provide opinions about the same topic. Hashtags are widely used
for events such as conferences, as in the case of #esriUC each year (Clark 2009), natural disasters, such as #Sandy (the 2012 superstorm on the East coast of the United States), as well as many others. This structure provides a consistent and convenient way to aggregate data on any one specific topic.

The geotag is what holds the locational data of the tweet. Twitter’s policy explicitly states that users should always be in control of location-based sharing, and the sharing should be explicitly turned on by the user. This serves two purposes: it adds a level of protection for the user, and location can be turned off, and also ensures intent on behalf of the user to publicly share their location data.

The location data in Twitter presents some limitation in itself. As most users do not enable location on their tweets or accounts, that information is simply unavailable. For the ones that do, there are three options to deliver the location data:

1. exact GPS coordinates provided by a handheld device,
2. a generated location name (usually in the form of a city name) generated by Twitter,
3. a free-text field where a user can enter any information (Davis et al. 2011).

The first category is the most valuable, as it provides the exact location of the user while submitting the tweet. However, this is also the least popular category, often available less than one percent of the time.

The second category, the automatically generated location name based on a Source Geography method of determining location. This method is based on the infrastructure of the Internet, as every transaction on the Internet can be associated with a relative location. This is usually based on IP (Internet Protocol) addresses, but can also use more modern techniques including or Wifi-based triangulation (Borges 2011). This data can most often be resolved to city-level accuracy of the user. Although this presents a limitation in the location variable, Davis et al. (2011) argue that city-level data is sufficient for many forms of statistical analysis, and is in fact the norm for health related or census based studies.
The third category, the free-text field is the most problematic location category, as well as the most prominent. The free-text field can present location as a street address (e.g. 30 Rockefeller Plaza, New York, NY), a named place (e.g. Manhattan), a named building (e.g. Madison Square Garden), an abbreviation (e.g. nyc), a formal name (e.g. Los Angeles, California), a jargon or slang name (e.g. Chitown), a specific Twitter client default (e.g. everywhere), or a comical non-location (e.g. other there). Techniques of geocoding can be used to address this form of data, since analytical software cannot process the text-based locations, and requires a geographic point or polygon. Conventional definitions of geocoding name it as the act of converting street addresses into geographic points. However, modern techniques extend that to any textual description of a location, including addresses, named buildings, place names or landmarks, postal (zip) codes or telephone area codes, and neighborhood or city names (Goldberg et al. 2007, Borges et al. 2011). Using online geocoding services, obvious free-text locations can be converted to points, and ambiguous, fictitious, or comical entries can be discarded.

Twitter’s inherent nature also presents several unique and powerful research incentives. Data is created at incredible speeds, and is available immediately for collection, allowing for automated and real-time collection and analysis (Gao et al. 2011). The data on Twitter is also largely public. The service allows users to set their accounts to private, allowing them to filter and choose with whom to share tweets, as well as choose to leave their accounts completely open and available to the public for browsing and searching (Clark 2009).

Sample previous work

Many have already realized the power of Twitter’s location data, and various studies have been done, both within and outside of academia, using data collected from Twitter. Among the many implementations are predicting the results of political elections (Tumasjan et al. 2011), futures predictions (Pang 2010), analysis on natural disasters, such as the earthquake in Haiti in 2010 and the tsunami in Japan in 2011 (Gao et al. 2011), and wildfire tracking in San Diego in 2007, today providing an important first-hand historical record of the event (Clark 2009), the spread of disease throughout a population (Davis et al. 2011), and tracking snowfall (Field & O’Brien 2010). Twitter is also being used to organize political action against governments, notably in
cases of protest organization in Moldova and Iran (Clark 2009) and the political unrest in Libya, Egypt, Yemen, Bahrain, Syria, and many other, collectively known as The Arab Spring (Gladstone 2013).

However, the most impressive examples of the power of Twitter come from outside of academia. The USGS has successfully implemented Twitter as an early warning system for earthquakes. Using live streaming, they are able to collect and analyze data quickly and distribute necessary precautionary information within a minute of first feeling an earthquake, often before the earthquake has travelled to all of the areas it will affect (Earle 2010). The study shows that users post tweets within 20 seconds of first feeling the earthquake, which is much faster than the 2 to 20 minutes it takes the USGS to collect the required instrumental data.

The power of Twitter is further showcased through a temporal and locational analysis and compilation of data across the United States during the 2009 NFL SuperBowl game between the Pittsburgh Steelers and the Arizona Cardinals. Data collected throughout the game was compounded and displayed on an interactive map as a tag cloud – a format where a word or topic varies in size based on its popularity. The data can be played through time for the duration of the game, where the different topics are scaled based on the specific popularity during that time (New York Times 2009). Interesting peaks can be seen in this data corresponding to scoring in the game, as well as a country-wide peak of the topic “Springsteen” during the halftime show performance.

Jer Thorp (2012), former data artist-in-residence at the New York Times has also demonstrated the powerful analysis potential of data from Twitter, yet done so as an enthusiast and artist, rather than a researcher. Noticing the phenomenon of users posting “Good morning” to the social network each morning, Thorp gathered this data over the course of a day, and visualized it on a three dimensional globe, aggregating the data based on time and frequency in each specific location. In essence, this analysis gleans into the sleeping habits of Twitter’s population of users.

Further, Thorp (2012) has conducted a second analysis, this time of travel data. Noting that people often tweet about having landed in a particular location after a flight, he gathered “just
landed” data, and mapped out the users’ home towns and destinations, visualized against time onto a three dimensional surface.

Studying social networks extends far beyond the realm of GIS. In fact, Twitter has made quite a splash in other areas of geographic work, namely that of media ethnography. Schoneboom (2011) describes social media as a vital and important archival source for ethnographic work due to the candid personal commentary available in its content. Unlike other social networks, like Facebook and Google+, Twitter does not require that an account be linked to a personal identity, allowing its users to do so pseudonymously, giving them a voluntary layer of protection and security to provide their candid experiences and thoughts. Further, Twitter exists as “a network that functions as a continuous background presence in the life of its active members, providing a steady stream of messages reflecting on their actions throughout the day,” (Crawford 2009). Through her work with workblogging – workers candidly talking about their roles and experiences at their workplace – Shoneboom (2011) has described the importance of social media data, as “writing about work and discussing it with other workers across industries can have a transformative impact on workers themselves. This process of ‘breaking the silence’ among workers and entering a sustained and honest discussion should be a part of the social scientific project.”

Online ethnographic work necessarily splits from the traditional ethnographic notions of field and fieldwork due to the sense of a networked world, and moreso exaggerated by the complexity of virtual networks and the multi-site research required by ethnographic work on the Internet. A higher emphasis should then be placed on the central role of object and technologies that organize those virtual worlds (Farnsworth 2010). Globalization presents the transformation of ethnography from an observation of “the exotic strangers of foreign lands” to a study of the others among us, or indeed the study of us. World-society is best described as a network of practices, in which social space holds the primary importance, and the physical space becomes secondary (Hepp 2004).

Rapid changes in blogging practices make such work more difficult for researchers, and among those difficulties is keeping up with ever-emerging new social networking tools, and the
continually morphing virtual field raised by these rapid changes. There is a growing sense of anxiety among ethnographers that in such a rapidly changing field, it is important to monitor and harness the field as quickly and regularly as possible (Shoneboom 2011).

Many argue that Twitter data is not representational, due to the population-specific biases introduced by the nature of an online social network, and therefore that the platform does not present a good source for popular opinion. However, as Tumasjan et al. (2011) argue, Twitter is not merely an opinion counter or a direct reflection of the opinions of the author. Instead, Twitter is a forum where users reflect and discuss the comments of others, as well as external sources, and weigh that information. Surowiecki (2004) presents conditions under which a non-representational, and large, sample of people can come to an accurate judgment. Those conditions are diversity, independence, and “decentralization combined with a mechanism to aggregate dispersed bits of information,” (Tumasjan et al. 2011). These conditions create what is referred to as the wise crowd effect (Surowiecki 2004), and Tumasjan et al. (2011) present a full argument that Twitter satisfies all of those conditions. The userbase of Twitter is indeed large, and is spatially decentralized and diverse. The pseudonimity of users – users don’t necessarily use their personal identity, but rather an online persona – ensures their independence from censorship pressures and each other (Schoneboom 2011). Due to the length limits of tweets, users often present outside links to discuss topics they are interested in. Users also directly engage one another in discussion, presenting not only their views, but those widely present online. All these mechanisms and behaviors allow the users to aggregate dispersed information, completing Surowiecki’s requirements for a wise crowd.

Twitter is also criticized for being spatially non-representational as well. The network is much more heavily used in urban and densely populated areas, with an unproportional amount of data coming from those areas (Field and O’Brien 2010). Both in visual representations and statistics, this positional inaccuracy may be misinterpreted as a lack of a certain phenomenon, rather than a lack of data. This is termed “cartographic confounding” (Oliver et al. 2005). However, instead of grossly exaggerating this problem, this should be treated as a definition of the data source. With that in mind, Davis et al. (2011) present that analytics tailored to urban data are valid for such a data source, and at its worst, represents a standardization problem.
**Geocoding**

As evidenced in the discussion of Twitter’s architecture, geocoding free-form location names plays a large part in collecting and analyzing located tweets. Geocoding arose as a system for matching street addresses to geographic coordinates, but has since been extended to include the conversion of any textual description of a relative location to a recognized point – this could include an address, a named building, a place name or landmark, a postal (zip) code or telephone area code, a neighborhood or city name, etc. – often done through the mixed use of a gazetteer alongside a conventional geocoding system (Goldberg 2007).

A geocoding system has the following four fundamental components: the input, the output, the processing algorithm, and the reference dataset. In the implementation of a geocoder, the input is the string form, or human readable, address or place name. It is first semantically pre-processed, cleaned, and standardized by the geocoder, in order to recognize the type of input given, such as an address, zip code, or place name, and the meaningful parts of the string, such as the house number and street name of addresses. The input is then recursively weighed against the reference dataset using increasingly relaxed restrictions, as defined by the processing algorithm, until a suitable match is found. Then, the correct geographic format is determined for the output of the location, in most cases geographic coordinates, or a polygon defining a region (Goldberg 2007).

Oliver (2005) points out that there are two major errors that can be made in geocoding: assigning an incorrect geographic point to a location – or positional inaccuracy – and differential match rates based on geographic regions. The former problem is the result of the processing algorithm. The biggest room for error in a geocoding algorithm is the production of zero or multiple matching results, or the production of tied results (Goldberg, 2007). In such cases, more data is necessary, or a supplementary dataset needs to be used to produce a better or more accurate result. Often times, geocoders take into account a bounding box for a query, in order to reduce the ambiguity of the query (Google 2013). A balance needs to be struck.
between comprehensiveness and accuracy, as a system with too much data will result in a lot of ambiguity, yet a system with too few data will yield a lot of empty or inaccurate results.

The latter is more difficult to catch and account for. Geocoding proves to be more challenging in rural areas, where less of the data is known. This results in non-random missing data, a phenomenon termed “cartographic confounding” (Oliver 2005). In particular, this systematically missing data is more concentrated in rural areas, disguising itself as an actual lack of locations, rather than a lacking reference dataset, and will therefore skew visual mapping representations and statistical analysis. Oliver also points out the obvious MAUP issues of geocoding using varying areal units. However, careful processing of the data can usually take care of such problems. For example, representing data points as scaled clusters will alleviate discrepancies between differing units. In statistical analysis, it is often appropriate to convert all data to the larger unit, as found in a study by Oliver (2005), yielding comparable correlation to using a more granular unit, in Oliver’s case converting census tract data to the county level.
Software Architecture

A major obstacle of technological work is that the state of modern software development does not exist within the realm of academia (Baranovskiy 2012). As such, materials on emerging technologies, the latest design patterns, and the open-source community do not exist in the expected journal articles, but rather on blogs, online forum communities, and social networks. I, however, do not wish to raise an institutional critique. Rather, I would like to note that the following section, where not specified, has been sourced from communication with technologists and software developers – including some of the developers of the specific software discussed – over online social networks and communities, as well as two and a half years of web development experience and a job position at a prominent online GIS company.

In the following sections, I will present the major parts of the software, the research that went behind selecting each technological piece, and the leading industry alternatives to those selected technologies. These sections span the topics of the format of the data collected, the development environment, the database software, and the live geocoding service. In each section, I will present the unique information about each topic, the criteria for its selection, and make an argument for the appropriateness of that selection to the project as a whole. When presenting the industry alternatives, I will only focus on the shortcomings of each, as each alternative is in itself a strong competitor.

Data format:
To begin this discussion, the data format is a natural point of entry. Twitter’s API – application programming interface, or the way of accessing data within the Twitter system – is designed for the web, and as such returns data in the JSON format (Twitter nd.). Since the latest version of the API, v1.1, JSON is also the only supported format by Twitter, removing support for other, outdated web formats. JSON stands for JavaScript Object Notation, and is a platform-
independent data-interchange format derived from the ECMA Script 3rd Edition revision (json.org nd.). It is, essentially, a minimalistic format for transferring any data using plain text for the purposes of web applications. Although all major platforms now support JSON – such as Microsoft’s offering of ASP.NET, as well as popular open web languages and platforms like PHP or Django (using the Python language) (json.org nd.) – this requires translation of the data into the native format for each language. JSON is most at home in a JavaScript environment.

Alternatives:
There are no alternatives in this case; the Twitter API now only supports the JSON format. It is possible to convert JSON to other data formats, though that takes an extra step that introduces unnecessary complications and possibilities for the system to break in the future. As will be evident in the following sections, it is possible and beneficial to stay within the native JSON environment.

The development environment:
The development environment includes the framework, tools, and packages used in the creation of a piece of software. I determined several requirements that need to be met for a successful development environment for this project, based on the most limiting use case of the software – namely independent researchers and graduate students working on personal computers or a computer lab. First, the environment should have a good implementation of JSON. Second, it must be able to handle large amounts of data both efficiently and quickly. Third, the environment should be independent of the operating system (Windows, Mac OSX, Linux, etc.) it is running on, since I would like this software package to be accessible to anyone on any computer they are already using. Fourth, it must provide easy to use and easy to understand development tools. Since this software package will be open to the end users (researchers), it is necessary that the software be written in such a way that it is easy for someone to read, edit, and otherwise work with the code without the need to understand the low-level functionality (such as eventing or networking).

I chose to work with a software package called Node.js, as it fit all of that criteria very well. Node.js is an open-source environment for running programs written in JavaScript, and it is
based on Google’s V8 Engine, the same engine that powers the popular Google Chrome web browser (McLaughlin 2011). This satisfies criteria point one; V8, as a JavaScript engine, has native JSON support up to all of the latest standards.

Node.js follows the evented pattern of programming – or an “if this, then that” pattern – which allows software to easily handle many events, both in type and quantity, all at once (Tilkov and Vinoski 2010). It is highly scalable and robust, and was designed to thrive in data-intensive and real-time applications (Node.js nd.). This more than satisfies point two of the environment criteria.

Node.js works on regular personal computers, web servers, and cloud hosted environments, and supports all operating systems (McLaughlin 2011). Further, this environment has a very small footprint on a computer and has virtually no requirements for installation, and is therefore truly suited to running on the personal computers of researchers, outdated or well below average hardware, as well as highly scalable cloud hosted platforms like Amazon EC2 or Nodejitsu. This meets criteria point three, as it will run on all hardware that the researcher may use, as well as many dedicated cloud environments.

Finally, JavaScript is regarded as a very good beginner language, and is the first language of choice for many when teaching programming to novices (Weinstein 2012). It is regarded as much easier to learn than other competing languages, like Ruby, Python, or PHP, by expert in coding education Zachary Sims, creator of the online educational tool Codecademy (Myers 2011). JavaScript is very tolerant, and can handle both object-oriented and functional programming, fitting many different styles and preferences (Johansen 2012), and is suitable to be edited and contributed to by both beginners and experts.

It is noteworthy to mention that JavaScript comes with a few limitations that may trip up those not familiar with the language. Since it is a loosely typed language, meaning that variables do not have types, but rather can hold anything, some problems arise when doing type coercion and comparisons. JavaScript environments will try to compare values across types. As an example, the numeric value 2 will equal the string value 2 when compared in this way: 2 == “2”.

15
The same works for other types. Both of the following are true: 1 == true; “1” == true. However, type coercion does not always provide the expected result. This kind of issue can be addressed by always using the “triple equal” operator, ===, instead of the double equal. This compares both type and value, and returns true only if both are the same. In this case, 2 === “2” is false, since one is a number and the other is a string. Another problematic area is closures, which can at times baffle even advanced developers (Baranovskiy 2012). In most languages, everything is executed with block scope. In JavaScript, however, code is executed with function scope and inherits the scope all the way up to global scope. In other words, each function is aware not only of its own variables, but also the variables of the function which created it, and the function which created the function which created it, all the way up the tree until it reaches the global scope. This can cause some unexpected behavior. However, using unique and descriptive variables names and maintaining proper indentation mitigates scope issues and allows for easier troubleshooting.

As a bonus to the criteria, Node.js has come to light at a high time for the Internet, and a lot of communities exist for it, which have provided great documentation and supporting libraries for the environment. NPM, the Node Package Manager, has many helpful libraries developed by enthusiast and professionals, and any JavaScript tutorial available online will function exactly the same in Node.js.

**Alternatives:**

**Native** – this option would mean developing the software entirely natively, in a compiled language like C, C++, or Java. This option, however, fails three of the four criteria points, and is the easiest to eliminate. These languages have strong classes as data formats, and presents the most obstacles for JSON data. It is very difficult to develop native software for all operating systems due to major differences in operating system implementations. Further, considering the scope of the project – requiring major components such as full networking, asynchronous data handling, parallel code execution and multithreading – it would require language mastery for an end user to edit, adjust, or add to the code if they would like to extend the project.
**ASP.NET** – this is a package developed by Microsoft for developing software in the C# language. The shortcomings of this package are in criteria points one and three, and in parts the last point. It inherits strong classes from the native C and C++ languages that C# is derived from, and therefore has problems with converting the JSON format. This adds unnecessary vulnerabilities where the software could break. Further, ASP.NET is a product built for Microsoft Windows, and it therefore will not run on any hardware available to a researcher, namely those running Apple’s Mac OS X or the various distributions of Linux.

**Django** – this is a framework for building both web-distributed software and packages software using the Python language. Like Node.js, many of the low-level necessities are provided with this framework. There is native JSON interpretation (although it is not the native data structure of Python), and it is capable of handling the large amounts of data Twitter may return. The main consideration for not using Django came from point three of the criteria. I believe that Python is less friendly for beginners, and it would be more difficult than JavaScript for the end user to edit, adjust, or add to the code.

**The database:**
Once properly collected, JSON data presents yet another challenge, and more specifically, the JSON collected from Twitter. Most computer data is governed by structured query language, or SQL, which defines how the data will be formatted and stored. It is a highly structured format. In most ad-hoc methods of collecting Twitter data used throughout academia, tweets are stored in SQL-based software like Microsoft Excel and Microsoft Access, or in formal database systems like SQL Server and Oracle. Popular GIS packages, like ArcMap, also expect SQL-based data. However, JSON is inherently – and importantly – not governed by SQL, and any data can be returned, including more or less data than expected. For example, Twitter sometimes provides a short-form of the tweets, in which a minimal amount of data is given. Other times, it provides a long-form, where various attribute categories exist, as well as entire attribute objects. The long-form can also arbitrarily drop some attributes. In the case of conversations, previous tweets will be embedded inside the current tweet. This will often result in recursive embedding – i.e. a tweet inside a tweet inside a tweet, an unknown amount of times until the first tweet from the conversation is reached. Further, these embedded tweets are often times not in either the
short-form or the long-form, but a level in between. Storing this kind of highly varying data in a strict, structured database is problematic, to say the least.

JSON data, and similar non-structured data, has spawned its own form of database software, called noSQL, which is often interpreted as data not governed by SQL – the literal “no SQL” – but in fact more closely represents data that does not rely specifically on the SQL format – the acronym case “not only SQL”. This is a relatively new format that has existed mostly theoretically until the mid-2000s, and is designed from the ground up to be a complete and competitive database format, on par with industry standards (Seguin 2011). Since the noSQL format is robust and capable of handling any combination of data, it is a perfect fit for storing Twitter JSON in its original form, as well as software-treated forms.

The database requirements, other than that mentioned above, are largely inherited from the requirements of the development environment. The database needs to be able to run on all operating systems, since it will be running parallel to the rest of the software, and it must be compact enough to run on personal computers, such as those commonly used by researchers.

One such database package is the open-source MongoDB (nd.). This package is widely used in the industry for noSQL storage. It is small enough to be used on personal computers with minimal hardware requirements, yet scalable enough to handle all of the traffic of Twitter if it is necessary. Further, it supports all major operating systems. Also significantly, MongoDB has geospatial indexing, allowing the data to be assigned coordinates and queried against location. MongoDB uses the BSON format (BSON nd.) for storage, which is the binary representation of JSON, which highly increases the speed of the database, without in any way limiting or hindering the versatility of the JSON format. Its versatility in data storage and speeds of data processing are a perfect fit for this research project. Further, MongoDB is often matched up with Node.js, and the use of the two packages in tandem has become ubiquitous in technology communities.

**The Alternatives:**

**SQL databases** – this includes all SQL packages, such as Microsoft SQL Server, Oracle, PostgreSQL, mySQL, etc. All of these packages have the same problems as those mentioned
previously. The small benefit in familiarity with the data storage is overshadowed by the conversion problems, non-standardness and unknown length of the data, and the general messiness of Twitter JSON.

**CouchDB** – this package is a direct competitor to MongoDB, and in fact, has most of the same abilities and functionalities. One notable difference, however, is that MongoDB has better indexing capabilities than CouchDB (Kovacs nd.). CouchDB is advised for the use of data-accumulation applications, where querying is done using predefined queries or MapReduce jobs (where all of the data is transformed at query-time, resulting in slow queries). MongoDB, on the other hand, provides data indexes and dynamic queries.

**Redis** – this is a very unique database package, which exists entirely in memory (Redis nd.). Although Redis does write to the disk for data persistence, the incredible speed it achieves is due to the fact that the database is stored in the RAM of a computer. Though it is a very popular package, it is not fitting for high volumes of data on personal computers, and is intended only for use with rapidly changing data or known size (Kovacs nd.) – the opposite of this software project – and generally requires a secondary database solution for more robust storage.

**Geocoding service:**

In place of doing the geocoding by hand – a very large task – I have selected to use a geocoding service. There are various services available to choose from, and all come with different strengths and weaknesses. Geocoding is generally a problematic service when it comes to large amounts of data, as most services are proprietary and impose usage limits on users. The major consideration for picking a geocoder was to use one that would not pose such limits on the amount of data collected. For this software package, I have decided to integrate with a service provided by Esri, a major GIS company. Esri provides an advanced geocoding service, and it is open for use by the public, and currently it is not rate limited. This service can take many forms of data, including street addresses, postal codes, and place names, and contains world-wide data. It is important to note, also, that the geocoding module for this software is developed separately, and can be replaced if necessary without disturbing the rest of the software in the future.
The Alternatives:

**Google Maps** – the geocoding service provided by Google Maps is known for providing some of the best results in the industry. However, this service requires an identification developer token, and places a 2,500 request limit per twenty-four hour period (Google 2013). This service would be ideal for small jobs, but is limiting for automatic collection of large datasets.

**Open Street Map** – Open Street Map (OSM) provides a Nominatim service, which works as a geocoder. It handles much the same requests as the Google and Esri alternatives, but it is powered by open-sourced data. This service also provides a bounding box or polygon data, in addition to point data, where appropriate. However, this service has strict usage limits for bulk use, requiring that requests be limited to one request per second, and that queries be checked and validated before making a request to the service (Open Street Map 2012).

**MapQuest** – this online mapping provider also hosts a version of the OSM Nominatim, which is free from usage limits. However, in my preliminary testing, I have found that this service is not as reliable as the one provided by OSM, and the results are not of the same quality.

Other considerations:
The Twitter API itself presents some more problems. Until late in 2012, it was possible to openly access archived tweets – that is, tweets from within the last week or so – through a search API. However, changes within that API now require user authentication to use, making it yet more difficult for novice programmers to use. Although the API is still considered open for use, it requires that every request be authenticated through the networks OAuth system (OAuth nd.), a popular system for online authentication without sharing of a user’s private credentials (such as a username and password pair). In the past, Twitter’s streaming service has had these authentication requirements, and as seen in the previously cited examples, this powerful service was rarely used in the academic setting, yet depended upon in the industry examples. If this is any indication, the authentication parameters are likely deferring academics from utilizing what is otherwise an open source of data.
Figure 1: Software architecture
Software Implementation

In summary, the software tool uses Node.js and MongoDB as the foundation. The resulting product is a complete package that can collect and pretreat data from Twitter, and prepare it for use by the researcher. The package is written entirely in JavaScript, and is capable of running on both personal computers and cloud-hosted servers – although, I advise running on a personal computer for research purposes. Algorithms in the software are fully modular and allow pluggable extensions by third-party developers with a minimum level of programming experience. Hooks have been provided in the code, both for collection time events and post-collection data processing, where researchers can create further data filters and processing without needing to change the core of the app.

The software allows for the collection of all tweets in a specific search. However, its power is in location processing. If enabled, the software processes each tweet for location, ensuring that each tweet collected has a valid location attached to it in a point format. The algorithm maximizes the amount of data that can be collected with a valid location, while still maintaining the quality of the dataset. Twitter returns data in a human-readable format, which is easy to read, but often not suitable for computational or automated analysis. For example, no piece of statistical or mapping software understands “New York City” as a location, and requires that the data be provided in the form [40.7142673320005, -74.0059698999996]. Esri’s live geocoding service is used to process string-based locations. However, extra checks are in place to ensure that the location provided by the geocoder is correct. The geocoder returns ranked results, and these are taken into consideration when dealing with ambiguity in place names. The software also performs a point-in-polygon operation for search tasks where a desired location is defined. Results are cached locally in the database and ranked, and are used the next time the location needs to be geocoded. The process is highly optimized, according to best practices, using a ranked database cache of all previously processed locations, as well as an in-memory cache of
recent and ongoing requests (Dunbar 2013), and will get better and faster the more that the software is in use. A ranking algorithm keeps track of location use, and over time, learns and stores popular locations, as well as optimizes access to those locations in memory. Further, any previously found location is stored locally, minimizing the amount of web requests made to the live geocoder. The more that the software is used by a researcher, the better it will become automatically at geocoding tweet locations. This data can be supervised through the software itself.

Twitter also provides time in a non-standard string format which is not computable on its own. The software converts this format into epoch time (also known as Unix time), which represents milliseconds since January 1, 1970. The date is then stored as an integer, and comparisons are meaningful (i.e. larger numbers are later in time). For example, the date March 12, 2013, at 2:31 pm would be represented as 1363113060869. This time format is a computer standard, and virtually all software understands how to compute with it. The time is indexed and fully queryable. For example, after collecting tweets for a few months, the researcher can ask the software to only return tweets for a specific day.

In fact, many query optimizations are in place, to allow for robust data extraction. It is possible to query against unique tweet IDs as stored in the database, as well as time. Further, when location is enabled during the collection process, it is possible to query the collected data against location. All data can be returned in ascending or descending order. These query parameters are available on every data endpoint, and can be combined to form more granular queries – something not always provided in the Twitter API itself.

As is, the software does not require any technological experience to use. Setup is as easy as unzipping and archived file in Microsoft Windows and filling out a web form. The Manual in Appendix A details the setup process, which only takes a couple of minutes. Further, the software is able to export the data into GIS or statistical packages through standard formats. It supports the GeoJSON format (a format understood by most open-source analytical packages) and EsriJSON (a format understood by ArcGIS products, and easily convertible to Shape files), as
well as the raw data and a shorthand format of the raw data, useful for manual inspection and software designed to accept data from Twitter itself.
Sample Dataset and Testing

Testing of the geocoding module requires a supervised dataset in order to ensure the validity of the results achieved using the module. As no dataset or benchmark was found the literature, I have created the following dataset to serve the purpose of testing this project, as well as for public use. This dataset was collected in the month of March 2013, during the re-election of the Pope. During this time, a popular meme emerged on Twitter – one that is reoccurring with various different popular events – under the hashtag #ReplaceMovieTitlesWithPope. The premise of this hashtag is simple – replacing one word in a popular movie title with the work Pope to create a humorous title – and it spawns a lot of tweets very quickly. This dataset was collected using a geographical query consisting of, mainly, the continental United States. It was defined as a 2000 miles radius from the point [39.8, -96.5], somewhere around the geographical center of Kansas.

The dataset is made up of 7,996 tweets. Since Twitter does not allow the distribution of collected datasets, I have simplified this dataset by stripping out most of the fields provided, leaving only those which represent the problematic attributes mentioned in this paper. To be representative of the original data, the tweet text has been included, as text, but all other identifying attributes, such as username, avatar image, user metadata, and tweet IDs, have been removed. The date the tweet was created remains in the dataset, as created_at, which can be used for the purposes of date standardization. Location is also presented in such a way that replicates the potential problems of the original dataset. Where coordinates were available, they are kept in their original format in the geo field. If they are not available, the field does not exist. String locations have also been kept in their original place, in user.location, and consist of the original string return by Twitter. This dataset also contains null locations and many duplicates, as these are often returned by Twitter as well.
The locations in the data were classified into seven categories. Three of those categories were classified automatically, and the rest were done by hand. The classifications appear under the `type` field in the dataset. The criteria for each category were as follows:

**Table 1: Supervised location categories**

<table>
<thead>
<tr>
<th>Category</th>
<th>Criteria</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>null (automatic)</td>
<td>No location parameters were available.</td>
<td>{ &quot;user&quot;: { &quot;location&quot;: &quot;&quot; }</td>
</tr>
<tr>
<td>coords (automatic)</td>
<td>Exact coordinates were available in the <code>geo</code> field of the tweet.</td>
<td>{ &quot;geo&quot;: { &quot;type&quot;: &quot;Point&quot;, &quot;coordinates&quot;: [38.51917546, -90.35800179] }</td>
</tr>
<tr>
<td>parsed (automatic)</td>
<td>Coordinates were available in the location string or the tweet, and were parsed out.</td>
<td>{ &quot;user&quot;: { &quot;location&quot;: &quot;ÂœT:40.702007,-96.702024&quot; }</td>
</tr>
<tr>
<td>real</td>
<td>A clearly recognizable and easily human-readable location. This can include any format, as long as the location is easily understood as a specific place by a human.</td>
<td>{ &quot;user&quot;: { &quot;location&quot;: &quot;Stillwater, OK&quot; }</td>
</tr>
</tbody>
</table>
| ambiguous       | A clearly recognizable location, yet one that is not clearly specific. Though this is a valid location name, it is not specific enough for a human to know where exactly it is. This includes, but is not limited to, places that are too broad, prolifically repeating placenames, and ones that may have a meaning but require more information. | { "user": { "location": "U.S.A" }    
                          { "location": "Windsor" }   
                          { "location": "OKC" }       |
| invalid         | A string which does not represent any recognizable place or placename. These are often humorous, or descriptive as generic places. | { "user": { "location": "Lost" }    
                          { "location": "My bed" }     |
Table 1 Continued

<table>
<thead>
<tr>
<th>slang</th>
<th>A human-recognizable location under a colloquial or slang name. This is a location that would be easily placed on a map by a human, yet indiscernible for a computer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;user&quot;: { &quot;location&quot;: &quot;SATX&quot; }</td>
<td></td>
</tr>
<tr>
<td>&quot;user&quot;: { &quot;location&quot;: &quot;Ttown, Oklahoma&quot; }</td>
<td></td>
</tr>
<tr>
<td>&quot;user&quot;: { &quot;location&quot;: &quot;Torontosaurus&quot; }</td>
<td></td>
</tr>
</tbody>
</table>

Even though there are four manual classifications – those that actually concern the geocoder – there are two meaningful groupings: valid and invalid. Completely valid locations are those classified as real. The rest are, in a sense, invalid, as the quality of the location string poses a significant level of uncertainty. However, they were classified in subcategories based on my beliefs of the abilities of the geocoder. For example, as geocoders evolve and get better, they increase their ability to handle colloquial or slang references to locations. One major example of this is “NYC”, a location string which does not exist in official datasets, but it is universally meaningful. There are other such locations, like “STL” for St. Louis, Missouri, and “SLC” for Salt Lake City, Utah, which may not be recognized by the current generation of geocoders, but could be recognized in the future. On the other hand, completely invalid locations, such as “in bed”, can never be recognized as mappable locations, and require a category of their own. Since the classification was mainly done to discern whether a string represents a valid place or not, and there are many complicated nuances of “non-real” locations, those classifications are overlapping and not very well defined, and can be misrepresented in the dataset. For example, placenames that fit more than one category may appear in either category, such as “Doooblin”, likely representing Dublin, may appear in either slang and ambiguous. Also, “Capsule Corporation, West City”, likely a reference to a location in the popular anime show Dragon Ball Z, in reality representing an invalid, fictitious location, may appear under slang as well.

The following is the breakdown of the locations in the sample dataset:

Table 2: Supervised location counts

<table>
<thead>
<tr>
<th>Category:</th>
<th>null</th>
<th>coords</th>
<th>parsed</th>
<th>real</th>
<th>ambiguous</th>
<th>invalid</th>
<th>slang</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>511</td>
<td>275</td>
<td>45</td>
<td>3285</td>
<td>2719</td>
<td>803</td>
<td>358</td>
<td>7996</td>
</tr>
</tbody>
</table>
After the classification, this dataset was tested using a test framework derived from the software. It tests the software in part, only running the specific code necessary for testing, and simulating all of the rest. Specifically, it tests the geocoder module. In the testing environment, this module returned four different categories (since, as before, the null, coords, and parsed categories were automatically detected). The categories are as follows:

**Table 3: Unsupervised location categories**

<table>
<thead>
<tr>
<th>Category</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>These values were retrieved by the location cache that is part of the software. The values for this category are considered valid locations.</td>
</tr>
<tr>
<td>live</td>
<td>These values were retrieved by the live geocoding service. The values for this category are considered valid locations.</td>
</tr>
<tr>
<td>outside</td>
<td>These values were found to be valid locations, however, they are outside of the originally defined query area. These values are considered to be invalid locations.</td>
</tr>
<tr>
<td>invalid</td>
<td>These values returned no location, and are considered invalid locations.</td>
</tr>
</tbody>
</table>

The results from this test are provided below:

**Table 4: Unsupervised location counts**

<table>
<thead>
<tr>
<th>Category:</th>
<th>null</th>
<th>coords</th>
<th>parsed</th>
<th>local</th>
<th>live</th>
<th>invalid</th>
<th>outside</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>511</td>
<td>275</td>
<td>45</td>
<td>5863</td>
<td>12</td>
<td>462</td>
<td>828</td>
<td>7996</td>
</tr>
</tbody>
</table>

There are obvious discrepancies between the supervised classification and the software classification. Although only 3,285 locations were recognized as real, the software was able to locate 5,875 (this is the local and live software categories, as they are inherently the same). Below, I have broken down the geocoder responses for each of the supervised classification categories.
Table 5: Classification problem areas

Classified *real*:

<table>
<thead>
<tr>
<th>Category:</th>
<th>local</th>
<th>live</th>
<th>outside</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>3105</td>
<td>5</td>
<td>25</td>
<td>150</td>
</tr>
</tbody>
</table>

Classified *ambiguous*:

<table>
<thead>
<tr>
<th>Category:</th>
<th>local</th>
<th>live</th>
<th>outside</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>2407</td>
<td>3</td>
<td>263</td>
<td>46</td>
</tr>
</tbody>
</table>

Classified *slang*:

<table>
<thead>
<tr>
<th>Category:</th>
<th>local</th>
<th>live</th>
<th>outside</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>163</td>
<td>0</td>
<td>111</td>
<td>84</td>
</tr>
</tbody>
</table>

Classified *invalid*:

<table>
<thead>
<tr>
<th>Category:</th>
<th>local</th>
<th>live</th>
<th>outside</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count:</td>
<td>188</td>
<td>4</td>
<td>304</td>
<td>307</td>
</tr>
</tbody>
</table>
Discussion

I have identified four problematic subsets in the data returned by the geocoder, each one corresponding to known problems of geocoding itself (Oliver 2005). These areas are:

- names classified as real, but determined as invalid by the software,
- names classified as ambiguous, but determined to be valid,
- names classified as slang, but determined to be to be valid,
- and names classified as invalid, but determined to be valid.

Below, I will discuss each group separately, and look at the specific values that fit inside each subset. Although there are 7,996 tweets in this dataset, only 7,210 of those have a string variable as the location. Of those, there are only 2,036 unique location strings, with multiple tweets for each location. Numbers presented in the below section are concerned with the unique values, and not the volume of data as a whole.

**Real names determined as invalid**

This category is problematic in its exclusion of valid data – an error of omission. Although this can pose a problem to statistical analysis. It is noteworthy, yet ultimately irrelevant, that these are the same problems as discussed previously in this paper.

Upon further inspection, this discrepancy is the result of poor categorization and misspellings. Examples from this subset are “Atlanta,Goergia”, a misspelling, “DEVINVILLE, Mississippi” and “Bumtown,AZ”, which are not a real cities. There are seven unique values in this subset, consisting of 0.3% of the overall data.
**Ambiguous names determined as valid**

This is by far the largest problematic subset. It is problematic because it introduces low quality data into the dataset – a potential error of commission. On further inspection, this data shows to be the subcategories of ambiguous that are locations which are too broad and multiple locations with the same name. Among the examples are many state names and state abbreviations, and country names. Also included are placenames with multiple associated locations, or polysemic placenames. This is often the case for many country names, as they are often repeated as the names of small cities across the United States. An example of this is “England”. It is most often recognized as part of the United Kingdom, and that is its likely meaning in Twitter. However, the GeoNames database has 239 entries of “England” in the United States (GeoNames, nd.). The case is the same with many other popular names, such as Africa with 25 hits, Belgium with 59, Norway with 240, and so on. This subset is made up of 419 unique names, and makes up 20% of the unique names in this dataset.

This major issue should be addressed in further work and optimization of the geocoding algorithm. Possible implementations could include any or all of the following:

- format enforcement – such as placing more importance on placenames consisting of a city and state combination, such as “Stillwater, CO”
- result ranking based on additional datasets and attributes
- using a different or supplementary geocoding service
- a location blacklist

**Slang names determined as valid**

This category is problematic for the same reasons as the ambiguous category – it potentially introduces low quality data into the dataset. However, looking at the specific data in this subset, it appears to be less problematic. The terms in this category are the more easily recognized ones, and can actually be counted as a win for geocoding services. Examples of this category include “Big Apple”, which was successfully places as [40.7142673320005, -74.0059698999996], the center of Manhattan in New York City. “STL, Missouri” and “St. Louie” were also correctly identified as St. Louis, MI, the latter of which was actually places at the St. Louis airport. “SLC, UT” was recognized as Salt Lake City, and “NYC” was recognized as New York City.
This subset consisted of 32 unique locations, or 1.6% of the unique locations of the dataset.

**Invalid names determined as valid**

This is another subset that presents a problem, largely due to its size and difficulty of detection, and presents another error of commission. This dataset contains 81 unique locations, or roughly 4% of the unique locations. Examples of this subset are “Cloud 9”, “Narnia”, “Wonderland”, “Unknown”, “Heaven on Earth”, “Rock Bottom”, “Candy Mountain”, “milky way”, “Moon”, “Garden of EdEN!!!!!!”, and “Hell”, and exactly duplicates the problems pointed out in the ambiguous subset. Wonderland has 34 hits in the GeoNames (nd.) database, Unknown has 32 hits, Candy Mountain has 11 hits, and Narnia has one, a city in Maryland. The potential improvements are also the same as the ambiguous subset.
Conclusion and Distribution

Though geocoding is certainly a large part of this project, providing a complete tool for researchers to collect and analyze Twitter data without requiring any programming expertise is the main goal. The software provides a complete interface for the two most popular collection APIs, searching and live streaming, and allows the setting up of tracking jobs with just a few mouse clicks. From there, the software collects and indexes the tweets, and exposes a powerful querying method, allowing researchers to easily dig into the data. The software also interfaces with ArcGIS products, allowing the researcher to then export the data directly into software packages which they are used to using, as well as a JSON format for importing into other packages or custom software. Further, this software is available freely and for all operating systems.

In the hopes of forming a supportive community around this software, the entire source code is available on GitHub, and collaboration is highly encouraged. The testing dataset described in this paper will be released as well, providing useful ways for researchers to benchmark other solutions in their analysis of tweets. Further contributions to this software should include detailed documentation of the code and its uses, as well as updating the geocoding services to Esri’s brand new service, which was released during the time of writing this paper. Links to the software download are available in The Manual in Appendix A.
References


GeoNames (nd.). <http://www.geonames.org>


Json.org (nd.). Introducing JSON. <http://www.json.org/>


MongoDB (nd.). <http://www.mongodb.org/>


Node.js (nd.). NodeJS. <http://nodejs.org/>

OAuth (nd.). <http://oauth.net/>


Thorp, J. Visualizing the world's Twitter data - Jer Thorp. *TED Ed.* 


Twitter (nd.). API Documentation. <https://developer.twitter.com/>


Appendix A: The Manual

Getting Started:

The code and all downloads are available from the Point Bank GitHub repository, located at https://github.com/catdad/PointBank. This manual is a living document, and will be updated as the software receives updates. It is valid as of April 18, 2013, but the GitHub page will provide the most up-to-date version, as well as an updated date.

I: Running the application in Windows

There is a Windows specific package for the application. There is a 32-bit zip and 64-bit zip, which include all of the correct binaries and settings. Download the appropriate zip file, and unzip it anywhere on your computer. It is advisable to use an easily accessible location, like C:\Workspace.

When you unzip the archive, you will see a start.bat file. Double click this file, and the application will start – it will open two separate command-line windows, although interaction with those windows is not necessary.

Note: before running the application for the first time, you must obtain access tokens from Twitter. This is described in section III of Getting Started.

II: Running the application in any environment

Although only Windows zip files are available, the application can run on any operating system or environment. However, more steps are required to do so. First, download the
plain application zip, which only contains the JavaScript files. Next, find a guide online for your operating system. You will need to install two software packages, called NodeJS and MongoDB. This process is further described below.

Google will help with this. Searching for guides on setting up and configuring NodeJS and MongoDB, and can be found by searching this phrase: “Install and running [software name] on [operating system name]” replacing the names with the ones you are looking for.

On a Mac, there are pre-built binaries of all of the files that are needed. These will be available at http://nodejs.org/download/ and http://www.mongodb.org/downloads. For Linux distributions, there is a good chance you will have to build these packages from source. Follow the guide that you find online.

The default file for the application is app.js. This will be the file that needs to be used when starting NodeJS. The software is designed to work with the default settings of both NodeJS and MongoDB, but if you need to change anything – such as port numbers – you can do so in the config.json file.

III: Setting up oAuth access
Authentication is required for all calls to the Twitter API. The application is not distributed with access tokens, so each user will have to generate their own. There is a keys.json file distributed with the application, where you will need to paste your keys. Getting keys is a straightforward process that is done through the Twitter website.

1. First, you must have a Twitter account. Sign up accordingly at https://twitter.com/
2. Go to https://dev.twitter.com/apps and sign in. This is the developer area of Twitter.
3. Click the “Create a new application” button in the top right.
4. In the form, give your application any name and description. Since this is for private use only, these values are not important.
5. By default, this application will run on personal computers, and not be accessible to the web. Therefore, the Website URL does not matter, and you can use a placeholder, such as http://www.example.com.

6. Leave Callback blank. Agree to the terms and submit the form.

Submitting the form will take you to the specific app section. Here, among other things, you will find the fields “Consumer key” and “Consumer secret”. These identify your specific application to Twitter. These need to be included in the keys.json file under consumer_key and consumer_secret, accordingly.

Next, click the “Create my access tokens” at the bottom of the app page. This will generate tokens specific to your user account (rather than specific to the app). Creating these tokens may take a second or two, but after you refresh the page, there will be two tokens available, the “Access token” and the “Access token secret”. These need to be places in the keys.json file as oauth_token and oauth_token_secret.

All four tokens are needed when making requests to Twitter. Make sure to save the file and keep it in its original location.

**Using the Application**

**IV: The basics**

The default URL is http://localhost:8888. If it does not open automatically, you will need to open it in your web browser. Note: some features may not work in all browsers. Development was done using Google Chrome. Also, if you changed the port number in the setup process, you will need to use that port number in the URL. For example, changing to port 8080 will make the URL http://localhost:8080.
V: Collecting tweets

Collecting data is done through specific tasks – think of a sentence like “search for tweets with the word ‘cheesecake’ in the continental United States that have a location.” All of this can be set up through the form. Here are the basic steps.

1. Pick whether you would like to Search or Stream. Search is most appropriate if you are interested in getting already submitted Tweets. This will take a bit longer, and is not in real time, but is helpful if you just want the Tweets that were submitted yesterday or in the past several hours. If searching, you will need to provide an interval, so that the search is repeated every so often. Stream is most appropriate for real-time tasks and tweets that will be submitted in the future. This is useful if you would like to track an event and you can start the task ahead of time, such as conferences or storms with enough warning.

2. Pick what you would like to search for. This can be any term that you would regularly type into the search box on Twitter. For example, you can search for words, phrases, or hashtags.

3. Select how many tweets you would like to collect. This is done to limit samples to a manageable amount. Note: infinite collections are not yet implemented, but you can pick a large enough number that you are sure will not be exceeded.

4. If you want tweets from anywhere, leave the default location of “Everywhere”. If you would like to add a location, however, you can do so using the map. The map will be used for a bounding box for the data, so make sure your entire area of interest fits in the map area.

5. Check the appropriate boxes for the location attribute of the tweets:
   a. Collect coordinates: this will enable the location attribute. It is safe to always check this box.
b. Geocode location names: this will enable geocoding of string locations. This will likely lower the quality of the dataset (from pure coordinates to city-center coordinates). Select this if you would like the most tweets placed on a map.

c. Collect tweets with no location: this will save all of the tweets that are found, regardless of location. The previous two boxes will be honored, so locations will still be collected; however, tweets without a valid location will be saved as well. Check this box if you do not care about location data.

Click Submit. This will take you to the Data page, where you will see a card for every task that you have created. Find the task that you want, and click Details. On the right side column, there are Action buttons, which allow you to start and stop the task, as well as delete it after you are done with the data. Click the start button to begin collecting tweets. 

*Note: if you restart the software, you must start each task again in order to keep collecting tweets.*

Figure 2 on the next page shows the task setup form.
Figure 2: The task creation form
VI: Getting the data

When you click on Data in the navigation bar, you will see all of the tasks that you have created. Each task will have a Map button that will show you a quick map, and a Details button that will show you more information and options for each task. On the details page, there are two columns. The left column will give you detailed information on the task that was created, as well as statistics on the tweets that were collected. The right column has data and visualizations options. The Action buttons allow you to start and stop collecting, as well as delete all of the data associated with that task. These buttons are further described in section V.

Various visualizations are also available.

- basic map: this will open a simple map and display the tweets.
- ArcGIS Online: this will open the tweets in ArcGIS Online using Esri’s format of JSON data, where you can save them as a webmap and export to various places. This is an external service.
- Google Maps: this will open the tweets in Google Maps using the GeoJSON format. This is an external service, and is not yet fully implemented.
- WebGL globe: this will open the tweets in a 3D globe. This is an experimental external service.

From the details page, it is also possible to query the data – this is the way to retrieve JSON out of the database. However, the interface for this has not been fully developed. Currently, it is only possible to query the data using a standard API format through URL GET requests. You can go to http://localhost:8888/api in the application to find out more about querying using the URL.

Figure 3 on the next page shows the a sample task specific page.
Figure 3: Task data page

Developers

The application is written entirely in JavaScript, and has various different modules. Here, I will detail all the files and the gist of what they do:

- **app.js**: this is the main file. It handles routing and some setup.
- **taskr.js**: this file contains task-based code, such as start, stop, and delete actions, as well as finding tasks.
- **searchr.js**: this file handles search tasks. It interfaces with the Search API and retrieves tweets.
• streamr.js: this file handles stream tasks. It is a wrapper for the Tuiter library, and has the same calls as searchr.js.

• geocoder.js: this file handles all geocoding, and interfaces with Esri’s geocoder. This file can be replaced with one for another service, although it has to respect all the same calls and provide the same callback options.

• taskHelper.js: this file has common code for all tasks. Most notably, this file saves all of the tweets from search and stream tasks.

• dataStore.js: this file is used to set up the database. All other files use the database connection set up here.

• headers.js: this file handles returning some of the common mime types.

• keys.json: this file holds the access tokens for Twitter. Refer to section III.

• config.json: this file holds some configurable options for the application. Refer to section II. You will generally not have to change anything in here.

• package.json: this is the install file for the application, as necessary for NPM. This file holds some metadata for the application. Most notably, it will have references to all of the dependencies. This file is required in order to run “NPM install” on the package.

• other files: the rest of the files are old versions for outdated iterations of the application. They are generally unused, but may have helpful code snippets for further features. Most notably, enrich.js shows how to add plugins for data processing.

• /public: this folder holds files that are used by the website interface. It holds CSS and client-side JavaScript. It may be necessary to edit these for UI tweaks.

• /views: this folder holds the individual website pages, written in JADE (http://jade-lang.com/). Think of these as the HTML files for the website.

This application was written with the Express library (http://expressjs.com/), and relies heavily on its features. It was written while Express was still in experimental development,
and therefore relies on the 2.5.x branch version. It is not compatible with newer versions of
Express due to API changes in app creation and the Jade engine. These are the other notable
dependencies:

- **MongoJS** ([https://github.com/gett/mongojs](https://github.com/gett/mongojs)): this is the database driver. It is used to
  connect to MongoDB, and provides all of the necessary methods to interact with the
database. This is usually attached to the app as `db`.

- **request** ([https://github.com/mikeal/request](https://github.com/mikeal/request)): this is the de-facto NodeJS web
  request library, and is used to interface with the Search API for Twitter, as well as
  the geocoding service. This can be used anywhere that a web request is needed.

- **Tuiter** ([https://github.com/danzajdband/Tuiter](https://github.com/danzajdband/Tuiter)): this is used to interface with the
  Stream API for Twitter.

- **GZippo** ([https://github.com/tomgco/gzippo](https://github.com/tomgco/gzippo)): this library is used to gzip large
  datasets. It was chosen due to problems with the Connect package built into the
  specific version of Express. If the app is updated to a new version of Express, the
default gzip options will work just fine.

- **proj4js** ([https://github.com/temsa/node-proj4js](https://github.com/temsa/node-proj4js)): this package is used in converting
  coordinate systems when Esri products request projected coordinates.