MyWeather: The Social Media of Weather

An Honors Thesis (HONRS 499)

By

Nathan Foss

Thesis Advisor

Fu-Shing Sun

Ball State University

Muncie, Indiana

December 2013

Expected Date of Graduation

December 2013
Abstract

Social media is a major part of Western culture today and more than 60 million Americans have accounts on various social media websites. People talk about a variety of topics on these sites such as pop-culture, breaking news, and even weather. Major news reporting agencies have begun using social media for leads on developing stories because social media users report news quicker than other sources. Reporters are able to take the leads, check facts, and report breaking news faster than ever. However, weather services have yet to utilize social media in this way. I set out to discover if social media is a viable source of weather data by creating my own website specifically focused on weather.

Acknowledgements

I would like to thank Dr. Fu-Shing Sun for his assistance throughout the semester. He helped me through the difficult aspects of the project and encouraged me to remain on schedule.

I would like to thank Mary Meadows and Spencer Lanning for helping me publish the website. Without their help, this project would merely have been theoretical, but they helped make it a reality.
The point of this project, at least from the Computer Science perspective, was to discover a way to effectively use a readily available, yet often unreliable data source. The data source for my project is posts on a social media site—which is often unreliable because users are not held to a standard of honesty. In order to effectively use this readily available data, my program must decipher reliable inputs and disregard false information automatically so that the program could fit seamlessly into any social media site. These demands presented a series of challenges that my program had to address. The main challenges were: finding a reliable weather source, accurately representing weather within the scale of the project, retrieving accurate location data from each user, allowing the program to determine true or false user input, preventing user abuse of the system, and performing all of these fast enough so that the user does not notice. I was also able to test the website on the internet and receive an impressive sample size of data while learning more ways to improve the site. Due to the time constraints on the project, there are aspects that I could continue to improve in the future.

Part 1: Finding a Reliable Weather Source

The heart of this project is in the program’s decision to accept or reject weather data entered by the user. Therefore, the first step in developing this project was to set an accurate baseline for current weather conditions. My partner advised me to use a type of weather reports called METARs. Basically, a METAR is a report sent by a weather station that is monitored by the National Weather Service and gives basic and fundamental weather conditions for the immediate area. According to my partner, METARs are an industry standard and meteorologists across the country use the data METARs provide.
METARs were exactly what I needed as far as their basic content and accuracy, but I also needed an output that my program could read. Fortunately, the National Weather Service maintains many different formats for METAR data—one of which is called XML. XML is a very simple and very standard format that utilizes tags to label data. This makes it very easy for a computer to recognize and use this data. I was able to quickly write functions to select the specific conditions that I needed and save the values to my database.

Part 2: Representing the Scope of My Project

Once I found a way to retrieve reliable weather data for my system, the next decision I had was defining the scope of the project. The scope involved two criteria: geographically speaking, what area can the METARs accurately represent, and how many stations do I need to balance accuracy and speed?

Major social media sites receive the vast majority of their posts from users in North America and Western Europe. While my project specifically focuses on weather data, it still requires users to post the information that my program can analyze. Based on this principle, my scope is immediately limited to users in North America and Western Europe. However, because my trusted weather source is the NWS, the only trusted data my program receives is from North America—specifically the United States.

However, the United States has a population of over 300 million people and an estimated 65 million have accounts on social media sites. These people live anywhere from dense cities to the vast countryside and my program needs to account for as many of them as possible by accessing weather stations in their vicinity. In order to maximize the utility of each station, my partner compiled a list of 505 weather stations across the country in areas with a significant
population. Once again, because the program relies so heavily on human interaction, significant population is a major criterion. My system receives data from a multitude of stations from states that have larger populations like California, Texas, and New York whereas states like the Dakotas, and Alaska only have a few. The result is that my program contains a general understanding of weather conditions that affect a high percentage of the population across the United States without surpassing the capabilities of the internet connection.

**Part 3: Retrieving a User's Location Information**

Once I had my database of weather stations across the country, I needed to determine a user's location. Without getting too technical, internet connections are established by sending a signal from a device—such as a computer, phone, tablet, etc.—to a router which sends the information to a local hub which then goes to the Internet Service Provider (ISP) and goes the remaining steps to the destination. Using Ball State as an example, the signal goes from a user on the bsusecure network to the router nearest the user which goes to the local hub in LaFollette and then on to the ISP and so on.

Every time the information changes hands, it receives a new destination address and sender address. I was able to use a free service (http://freegeoip.net/json/) which takes the sender address and gives a frightening amount of location information. However, with our example, the addresses within Ball State's campus are too localized to retrieve any location information. Also, the address of the ISP is not local enough to give accurate location information. With our example, the address I need is the sender address of the hub in LaFollette because it is local enough to accurately represent Ball State and the location service recognizes the address and gives the location data I need—i.e. latitude, longitude, and state.
Part 4: Allowing the Program to Determine True or False User Input

As I mentioned earlier, this single decision is the thrust of my project. Everything leading up to this decision focuses on making the decision more accurately while everything after this focuses on making the decision process quicker.

The first step in this decision-making process is analyzing the content a user is posting to the site for weather-related keywords. These keywords include basic weather terms which users with no meteorological experience would talk about, such as: rain, snow, hot, cold, or even severe weather terms like thunderstorm, hurricane, and tornado. The program can also match these as partial words, so if a user puts the word “rainy” or “raining”, the system will still recognize “rain”.

However, this information by itself is just one part of the decision. It is possible that a user could refer to a hot drink or use the keywords in a context other than one referring to the current weather conditions. This is where the trusted weather data comes in. If the program recognizes a keyword, it will proceed to search for the closest weather station in the database to the user’s location. This happens by calculating the distances between the station’s latitude and longitude coordinates and the user’s. In order to speed up this part of the process, I limit the search to stations in the same state as the user.

Once the program retrieves the closest station, the system can begin to compare the two weather inputs. However, there is a major design challenge because the trusted weather source only updates hourly. The system must be able to account for conditions that may have manifested after the trusted data last reported. My system needed to forecast for weather events that could happen an hour in the future, and I once again consulted my partner. Fortunately,
some weather conditions, like temperature, are unlikely to change drastically in an hour span. However, isolated rain showers and especially tornadoes can develop very quickly.

To solve this problem, I had to search for secondary conditions that can lend credence to an input that cannot be validated. For example, in the event of rain, a METAR has a field that shows the amount of rainfall in the past hour in inches that my program can check. If that has any value other than 0.0, the decision is an easy yes. However, rain could roll in after the METAR updated last, so the rain level on the METAR would be 0. In that case, my program searches for the secondary conditions of rainfall—falling atmospheric pressure, and cloudy or overcast skies. While falling pressure and cloud cover do not always lead to rain, it allows for a more educated assumption because rain rarely occurs with clear skies.

The trickiest weather event to check for is a tornado. There are definite signs that meteorologists use to issue tornado watches, such as severe thunderstorms, rotating cells, and large local pressure differences. However, these are too specific and localized for METARs to reliably report. In the event of a tornado, my program looks for a tornado report in the METAR, which returns an easy yes. Secondarily, it checks for reports of funnel clouds and can use that to return a yes as well. However, these can develop quicker than the 60 minute interval that the weather data has.

Part 5: Preventing Abuse

Social media often reports events much faster than news sources, which is a major benefit. However, the reason social media has this ability is because users are not held to any standard of credibility. Social media users do not have to check sources and verify information which speeds up the process; but it makes data highly unreliable. Because my project needs to be
as accurate as possible, I needed to account for the possibility of abuse. My program handles this, in part, by checking weather data from the users with trusted data from the NWS; but there is still a possibility false data could fall through the cracks due to the hour interval between reports. My program must allow for slight deviations in the weather reports to account for potential accurate weather changes; but these allowances provide room for exploitation which necessitates more protection in my program.

The situation I envision that I need to protect against is a small group of users who collaborate to post a large amount of false data in a short amount of time just to disrupt the system. Even if these users do not know the weather conditions my program employs, there is a possibility the users could get lucky and ruin the integrity of the output. To handle this, I added a requirement that is invisible to the user. Simply put, once a user posts weather-related content, the program will not analyze any content posted by the same user for the next 15 minutes.

This solution raises an issue when users submit multiple reliable weather-related posts during one continuous use. This issue is an inevitable side-effect of increased security. I would much rather exclude some accurate data as opposed to accepting false data.

**Part 6: Improving the Program’s Efficiency**

My program needs to perform a number of functions to validate weather conditions that a user could post. Yet, at the same time, the program needs to handle the heavy traffic that social media sites receive. Social media sites allow users to post their own thoughts and browse through other users’ posts. My functions analyze the content that users post, so it makes sense to perform the functions as soon as a user submits their content. When handled this way, users who simply browse the site do not have to wait for the functions to execute. Also, even when a user posts
content that is not weather related, my program recognizes that and performs no further analysis. Furthermore, if a user has posted weather content within the past fifteen minutes, the program performs no analysis on the post. These steps prevent the majority of users from experiencing any delays that the analysis functions may cause.

However, some users will still experience some delays. The toughest aspect of my program was updating the METARs every hour to keep weather conditions current. I handled this by comparing the current time and METAR time when a user submits a weather-related post. If the METAR is over an hour old, then my program updates all the METARs for the state in which the user currently resides. This is the longest computation that my system performs, which causes a noticeable delay for larger states. However, the worst case scenario is that one user from each state will experience this delay once an hour. While not ideal, this is an acceptable compromise that allows for a proof-of-concept. The other analysis functions provide no noticeable delay for the user.

**Part 7: Test Results**

My website was active for nearly two weeks and users all over the country created accounts and posted information. The majority of the users came from Indiana, Illinois, and Michigan, but users also posted to the site from Texas, California, and Florida. The last three states were very important for our research because they contained the largest concentrations of weather stations, which meant that users in these states would represent the worst-case scenario for my site performance-wise. If users that posted to the site from these states had any problems, then the project would not be viable. Furthermore, I had multiple users post to the site from
mobile devices over Wi-Fi or data connections which once again tested the performance of the site.

One particular user fit into both categories. She visited the site on a mobile device in Florida while on a public Wi-Fi or 3G data connection. If there is any connection that would not be able to handle my site, this is it. However, when I asked her if the site was slow and unresponsive when she posted to it, she did not remember anything out of the ordinary. Even on her device, my site sorted through all of the weather stations and analyzed her post—even deeming it accurate—all within a couple seconds. The users from Texas and California responded similarly, though they were using better internet connections.

My site achieved one major goal by performing tasks quickly, but I also needed to test whether the functions would properly analyze content users posted. I wrote my functions expecting users to post content that contained weather-related keywords that my program could find, quantify, and accept or reject. I expected these posts to happen naturally as I observed on other social media sites. However, due to time constraints, I specifically prompted users to post about the weather and left them to determine the conditions they deemed worth reporting. The results were different than what I was expecting.

Users often posted very specific data rather than the generic keywords I had expected. As such, my functions could not really handle that data effectively. Some users even copied the local weather report from the internet onto my site and my site failed to recognize the post as weather-related because no keywords matched. Users would also include “snow is melting” and “no wind” that would create keyword matches but my site would conclude that the data was inaccurate.
Nearly every post included a comment about the temperature, which makes sense as it often is the most important aspect of the weather as far as people are concerned. However, I had only accounted for relative temperatures like hot, cold, or warm. Most users did include those keywords along with the actual temperature which allowed the functions to still validate that weather condition.

The final glaring result from testing involved weather conditions relative to the climate, which is something I was unable to account for within the scope of this project. For example, many users posted about the cold, rain, wind, and snow—all of which are to be expected in November and December. However, users raved about the warm weather because they experienced 50 degree temperatures in December. Conversely, it is expected that users would complain about cold weather if the temperature is below 70 during summer months. These posts are accurate as most people would consider these temperatures to be warm or cold relative to the normal climate, but I was unable to take climate into account due to the constraints of the project.

Part 8: Future Improvements

My current version of the program simply saves the location, time, and user input of weather posts deemed valid. In my proposal, I stated that I would output the results on a map of the United States similar to conventional weather maps. However, for the sake of time I omitted that part of the project. Alternatively, I chose to store specific information in the database that would allow me to build a map in the future if necessary.

Another improvement I could make in the future involves my keyword matching algorithm. My algorithm simply looks for a small set of words and does not attempt to
understand grammar or context. In a later version, I could write a smarter algorithm that attempts to understand the context of the post in order to make a more accurate analysis.

Part 9: Conclusion

Overall, this program provided challenges at every step. The two types of data my program needed to compare came from very different sources and were different forms as well. My program needed to recognize both types of data and make an informed decision about which source to trust. My partner and I had to discuss the warning signs of the weather conditions that will happen after the weather source updates. These allowances must also be protected against abuse that is rampant on the internet so that my project remains accurate. Ultimately, this project took a great deal of work and I am very proud that it functioned so well with actual users visiting and posting to it.
Resources

Weather data came from the link:

http://weather.aero/dataserver_current/httpparam?dataSource=metars&requestType=retrieve&format=xml&stationString=KMIE&hoursBeforeNow=2

(This link is for the Muncie station specifically. My program accesses other stations by changing the station string)

Location data came from the link:

http://freegeoip.net/json/

Public Address for Website:

http://delaware.cs.bsu.edu/myWeather/