# Tweet Rises: Twitter Sentiment Analysis

Aleksander Bello
California Institute of Technology
abello@caltech.edu

Alexandru Cioc
California Institute of Technology
acioc@caltech.edu

Victor Duan
California Institute of Technology
vduan@caltech.edu

Archan Luhar
California Institute of Technology
archan@caltech.edu

Louis O'Bryan
California Institute of Technology
lobryan@caltech.edu

## ABSTRACT

This paper focuses on the work of the California Institute of Technology CS 145 group: Tweet Rises. It focuses on a combination of Twitter sentiment analysis and effective web-based visualization.

The group worked with several forms of natural language processing and machine learning, and with two primary visualization methods.

## Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services; H.5.3 [**Group and Organization Interfaces**]: Web-based interaction

## General Terms

Visualization

## Keywords

Twitter, sentiment analysis, 2D visualization

## 1. INTRODUCTION

How is the world feeling right now? That is a hard question so to make it easier and narrow the scope down to the quantifiable, we ask, how is Twitter feeling right now? We propose here an application and underlying infrastructure to categorize Tweets based on emotional content, create a representative sample, and visualize the sentiments on a map in a web browser in real-time.

## 2. FRONTEND

In order to make our work available to the widest audience, we decided to work on visualizing our results within a web browser. A major challenge was in, first, developing a working prototype and in, next, iterating upon it on a weekly basis. Among our most important priorities was to prevent our clients from being flooded with too much information and, from the other standpoint, still providing enough information to make viewing the website meaningful.

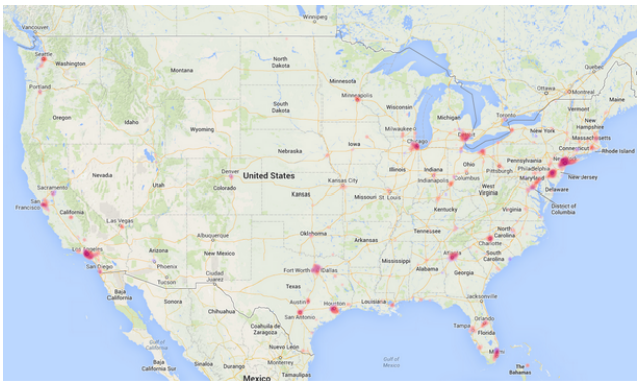


**Figure 1: Heatmap of Twitter sentiment.**

Our development primarily utilized the Google Maps API in order to achieve a fast and effective visualization. Google Maps allowed us to focus on the visualization itself, and saved us the time of having to create a scalable map of the United States.

Originally, we intended on visualizing our results using a heat map, but quickly discovered that a more understandable mode of communication was to use a "state map." Our primary concern with using the heat map was that the Google Maps API heatmap naturally scales based on the amount of data it receives. This means that, within seconds, states and cities with small populations have their data points effectively reduced to obscurity, and our entire heat map therefore only shows data for cities like Los Angeles, San Francisco, and New York City. As previously mentioned, we therefore focused on a different type of visualization, the "state map."

Our idea for using our "state map" came from analyzing various forms of 2D visualizations, and seeing that a particularly understandable visualization came from geographic electoral maps during U.S. elections. This forms of visualization depicts an entire state with a solid color that indicates that

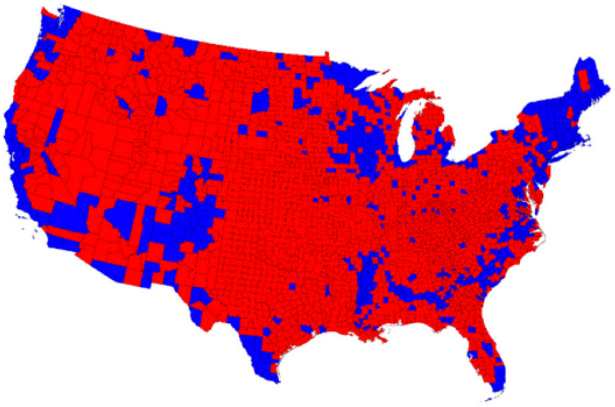

**Figure 2: Example US electoral map.**

state's political preference. We believed that, since people have already been predisposed to understand these maps from news coverage, we could lower the barrier to entry for understanding what our data actually depicts. Therefore, in our own work, we decided to create an independent geometric shape for each state using the Google Maps API, and then colored the state as the average color of the Tweet sentiments it received. For visualization purposes, we let negative emotions be depicted in red and positive emotions be depicted in blue.

An issue that quickly arose was that, in averaging the color over every Tweet received, every state would, given an adequate amount of time, become the same shade of purple - roughly an equal amount of "red" and "blue" sentiments. We believe that this makes sense since, over a prolonged period of time, we should notice an equal amount of both positive and negative sentiments since Twitter has a very wide audience so whenever a wave of positive or negative sentiments are shown, people often respond with a contradicting sentiment.

In order to alleviate this, we decided to allow for specification of how many Tweets to average over. Thus, by lowering the number to something more manageable, like 10 Tweets, we see a more meaningful rise and fall of positive and negative sentiments.

Our final touches to our frontend's visualization came in the form of sidebar indicating trending topics and their overall sentiments. Clicking on a topic allowed users to view the visualization for that single topic. This provided more meaningful information for users who aimed at gauging overall sentiments for a single topic as opposed to the overall Twitter Tweet stream. Even further, the sidebar allows users to estimate, at a glance, what a topic's sentiment is. This could potentially be seen as an exploratory tool, since it gives users a chance to notice outliers - topics that might be heavily weighed towards one sentiment, and then easily gives them access to see the map for that topic.

Overall, we believe our techniques provided for an effective visualization of our Twitter sentiment data.

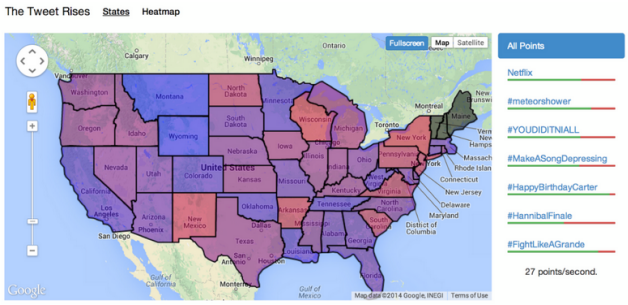


**Figure 3: State map of Twitter sentiment.**

## 3. NATURAL LANGUAGE PROCESSING

The first step to analyzing the Tweets is natural language processing (NLP). The techniques used to classify the Tweets all focus on a bag of words approach. As such, the NLP portion of the project focused on developing an efficient way to get the best bag of words from any given Tweet.

Initially, we eliminate obvious stop words that don't contribute to the content of a Tweet. This list includes words such as 'a', 'I', 'she', etc. Afterwards, we define "words" as a string of alphabetic characters with whitespace on both sides. Note that this ignores things such as numbers and emoticons. Once the set of words in each Tweet has been computed for each Tweet in the training data, mutual information is used to determine the words that provide the most insight to the content of the Tweets. For our purposes we used about 1000 words. With these, each Tweet was thus characterized by which of these 1000 words appeared. For example, for the Tweet "I am not happy. He is not happy" and the mutual information words "not" and "happy", the Tweet would be characterized as ["not", "happy"]. Note that the number of times a word appears is not taken into account.

Once a Tweet has been characterized by the above steps, it is passed along to the machine learning portion of the classification.

## 4. MACHINE LEARNING

Our machine learning methods consisted of four algorithms: Naïve Bayes, Stochastic gradient descent, Support vector machines, and Maximum entropy. Our first implementation was Naïve Bayes, due to its simplicity. Naïve Bayes predicts the classification of an observation by providing a particularly simple formula for the probability that an outcome C is observed given that there are features $F_1, F_2, ..., F_n$ in the observation. These probabilities can be compared for each outcome to find the most likely one. Specifically, the model assumes that the features variables $F_1, F_1, ..., F_n$ are independent, and the assumption implies that

$$p(C|F_1, F_2, ..., F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i|C)$$

where $Z = p(F_1, F_2, , F_n)$ is the 'evidence' for these features. In our case, the outcome C was whether the Tweet had

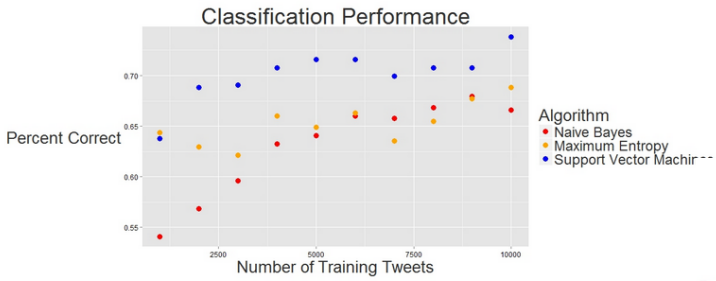


**Figure 4: Performance of Naive Bayes, Maximum entropy, and Support vector machines algorithms for up to 10,000 training Tweets.**

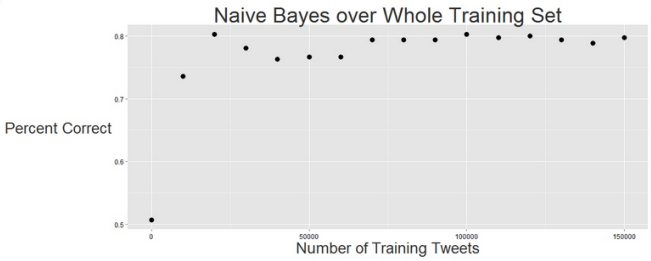


**Figure 5: Performance of Naive Bayes classifier for up to 1.6 million Tweets.**

positive or negative sentiment, the features were the words determined by the mutual information algorithm, and the probabilities $p(F_i|C)$ were determined from the training data depending on their appearance rate. So, the formula gave a way to compare the likelihood of the two sentiments given the words in the Tweet.

One problem with this approach was that if a word did not appear in both positive and negative sentiment Tweets, the probability $p(F_i|C)$ was zero for one of the outcomes. Although these terms were unlikely, they occurred in our calculation when we limited the number of training Tweets. We decided to simply leave these terms out of our calculation.

The other three algorithms, stochastic gradient descent, support vector machines, and maximum entropy, were implemented in the python scikit-learn package. So, our work with these algorithms mainly involved tuning the parameters to the scikit-learn functions. For example, we changed the loss function, number of iterations, learning rate, and whether or not to fit the intercept for stochastic gradient descent.

Using 10,000 training Tweets, all algorithms but stochastic gradient descent had an accuracy rate between 65% and 75% on the test data set. The performance of our stochastic gradient descent implementation was poor, so we left it out in the end. The support vector machines algorithm performed better than the other algorithms when the number of training Tweets was less than 10,000, achieving over 70% accuracy.

Only our Naïve Bayes algorithm was able to process significantly larger training sets in a reasonable amount of time. We were able to train Naïve Bayes on 1.6 million Tweets, which gave the algorithm almost 80% accuracy, outperforming the others. The other algorithms may have performed better with the same amount of training data, but they took significantly longer at only 20,000 Tweets, so this was impractical to test.

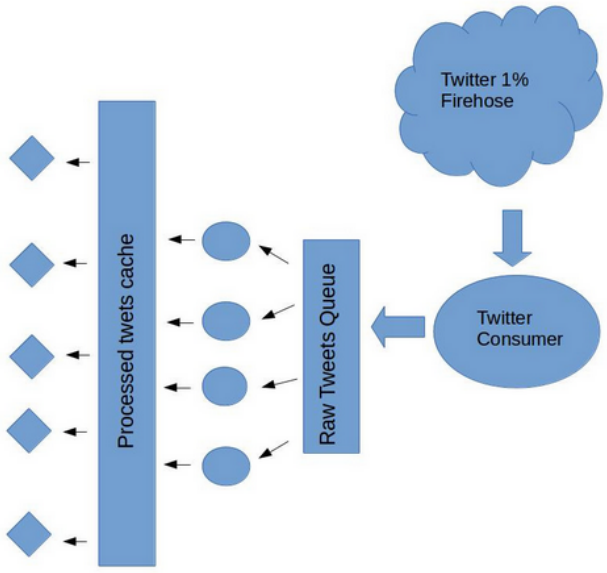


**Figure 6: An overview of the backend infrastructure. The ellipses represent the NLP workers, whild the rhomboids represent instances of the frontend servers.**

## 5. BACKEND

The first thing that needs to be done before we can produce any results is to have the raw Tweets, i.e. the text and geolocation information. This is obtained by the Twitter 1% firehose API. There is a persistent connection between our backend and Twitter that continuously streams new Tweets

in realtime. More specifically, we have two open streams with Twitter: one to get a 1% sample from all Tweets, and one to get a sample only for the specified trending topics. The trending topics are collected and updated by a periodically running script. This approach also allows for custom trending topics that we might like to add. All of these Tweets are stored on Redis, a simple in-memory database. There is one process worker for each stream, so that both streams can be consumed at the same time. It is worth noting that the official Twitter API documentation does not allow multiple streams. Moreover, these consumer workers have to be fast enough to keep up with the upstream Tweets, otherwise the connection will be dropped. To mitigate these two issues (and any other connection issues that might arise), several supervisor mechanisms are set up to restart these services.

After the raw Tweets are obtained, they need to be processed before they can be served to the frontend. The processing part consists of the NLP/machine learning workers categorizing the raw Tweets as positive and negative sentiments, extracting out the geolocation data, and then storing this information on a second database. This being the most compute intensive part, is fully parallelizable; more workers can be spinned up to parallelly consume from the raw Tweets queue. Unfortunately, Twitter does not support querying by both geolocation data and topic, thus not all trending topics will have geolocation data attached. We do, however, store all of them, so that we have a more complete sentimental assessment of the trends.

In parallel to the Twitter consumer and sentiment categorizer, we have a node.js server that interacts with the user client. This server has two objectives: handle the requests for the website's static files and send sentiment data points in real time via an open socket connection. We used several node.js libraries. Using node.js's built in http server, we mapped all http port 80 requests to a static content folder containing html, css, images, and javascript. Using the third party socket.io library, we structured the data point communication. On a new socket connection, the server sends all sentiment points less than ten minutes old. Then, for every current trending topic and hard coded permanent topic, the server sends the last 24 hours worth of points limited to a maximum of 1000 points. Independent of the individual socket connection logic, the server sends all points gathered in the last ten seconds to all connected clients every ten seconds.

# 6. FINAL PRODUCT

The final result of our project is a real time Twitter sentiment analysis tool. There are two modes: states map and heatmap. The states map collects the last five sentiments of Tweets from each state, and displays the overall sentiment of each state. The heatmap plots individual points for each Tweet it receives, and the colors on the map represent the overall sentiment of that small area.

Additionally, there are topics to choose from on the side. Some of the topics are determined based on what is currently trending on Twitter. Others are custom topics that we thought we be interesting for a new user to see upon first visiting the site. Due to the limitations of the API, we were unable to query for topic and location (Tweets with location tag in USA) simultaneously. As a result, it was difficult to get a lot of data for the topics that also had location data. The currently trending topics often had fairly little data, and the states are not all colored in. For the custom topics, we tried to hold on to the data over a longer period of time, giving us a chance to acquire more information on those topics and display a better map.

See below for figures of the final state of the project.

# 7. FURTHER WORK

We believe further work could be done on our project's frontend. Since our "state map", is an effective tool, we could create sub-shapes for each state in order to see sentiments for specific counties. Even further, we could continue work on our heat map. We switched to the "state map", becomes of inherent problems that the heat map caused, but we did not have time to return to the heat map and actually fix the problems we encountered. Thus, while our "state map", looks like a completed final project, the heat map remains in a rudimentary state. Lastly, for our frontend, we could try and speed up switching between topics. There is current slowdown after a large number of points have been added so optimization changes would prove effective.

# 8. ACKNOWLEDGEMENTS

We would like to thank Professor Adam Wierman and Lingwen Gan for helpful advice and guidance throughout the project.

# 9. REFERENCES

[1] A. Go, R. Bhayani, and L. Huang. Twitter Sentiment Classification Using Distant Supervision. *CS224N Project Report, Stanford*, 1-12.

[2] E. Kouloumpis, T. Wilson, and J. Moore. Twitter Sentiment Analysis: The Good the Bad and the OMG!. *ICWSM*, 11:538-541, 2011.

[3] A. Bifet and E. Frank. Sentiment Knowledge Discovery in Twitter Streaming Data. *Discovery Science*, 2010.

[4] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes Twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, 2010.

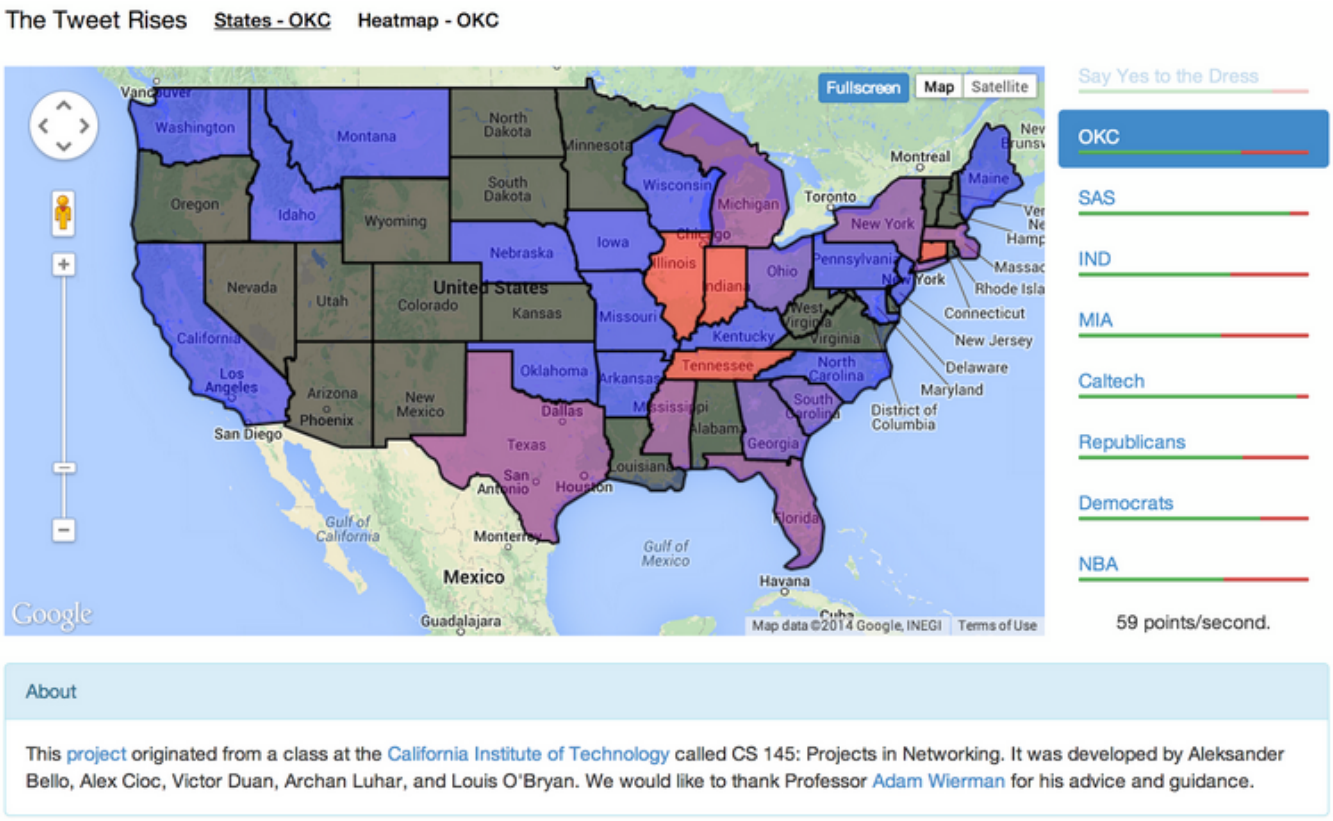


**Figure 7: Final state map product, focusing on the OKC topic. At the time, the Oklahoma City Thunder were playing the San Antonio Spurs in the Western Conference Finals of the NBA in 2014.**

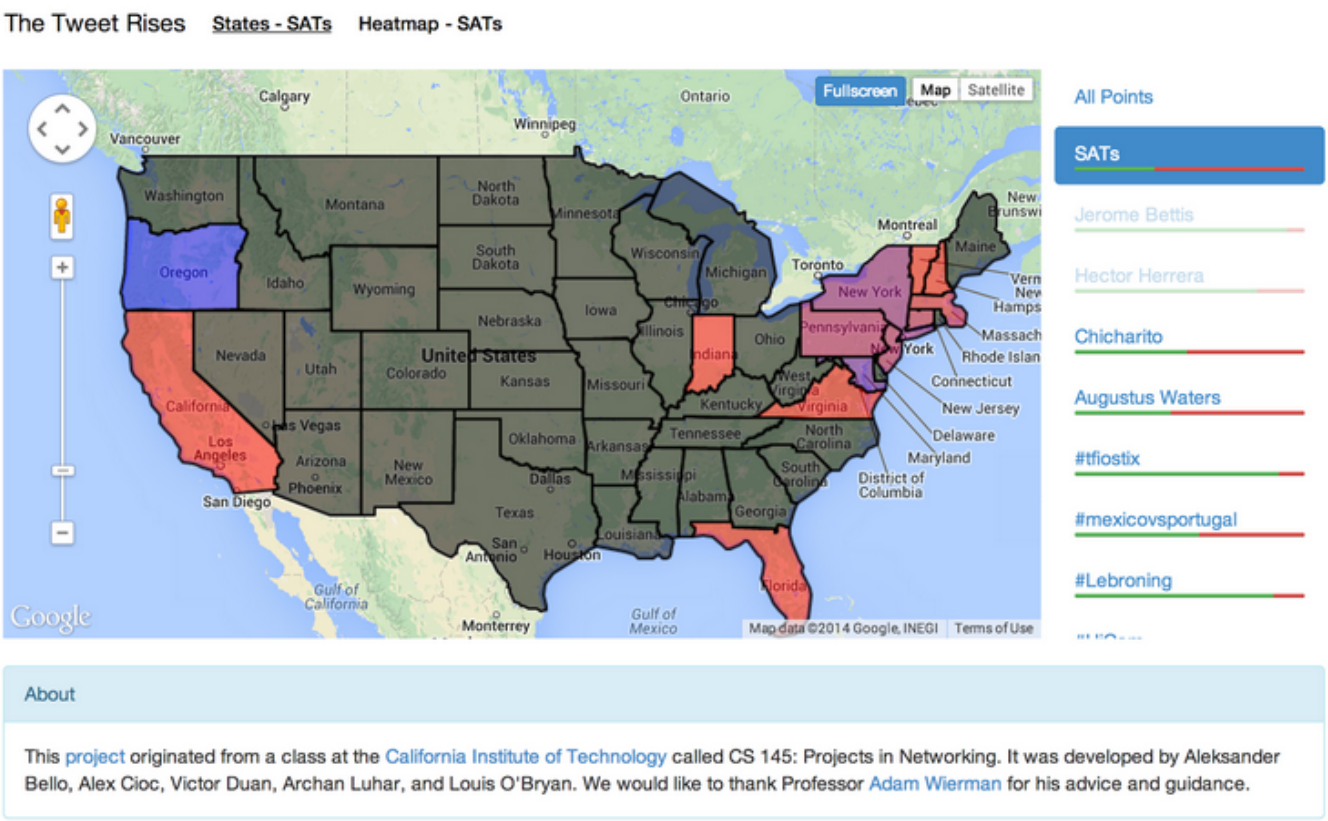


**Figure 8: Final state map product, focusing on the SATs topic.**