Software Design Description

Version 1.0
10.18.2012

Political Mood Gauge Based on Social Media

Team Vikings:
Gresham Schlect
Serendel Macphereson
Ryan Sacksteder

Prepared for CS383
University of Idaho - Dr. Paul Oman
Fall 2012

This document was written using a template provided by the Jacksonville State University MCIS department.

Table of Contents

This document was written using a template provided by the Jacksonville State University MCIS department.

List of Figures

This document was written using a template provided by the Jacksonville State University MCIS department.

1.0 Introduction

1.1 Purpose

This Software Design Document provides the design details for the social media based political mood gauge project. The expected audience of this document is the University of Idaho, specifically Software Engineering 383 (instructed by Dr. Oman), developers, and any users wishing to view the project's design specifications.

1.2 Scope

This document contains a complete description of the design of the Political Gauge system.
The system is being developed using the Spiral methodology. The architecture of the output for the system is a basic HTML Javascript web page which is updated by a program that requests information from external systems and processes data. The external systems interfaced by this system are Twitter, a social media site, and a sentiment analysis tool either Sentiment140 or Alchemy depending on user configuration. Other external tools include:

- jQuery
- JustGage - used to display a gauge as output on the web site
- libcurl - curl requests
- jsoncpp - json parsing

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 Social Media - technologies that create online communities/networking capabilities, e.g. Twitter, Facebook, MySpace and/or internet blogs
1.3.2 Twitter - a social media website that allows users to publicly share short sentences, or Tweets
1.3.3 Tweet - a text string posted by a Twitter user (140 characters or fewer)
1.3.4 Retweet - the process of reposting someone else's tweet on one's own profile
1.3.5 Follower - one user on Twitter can 'follow' another in order to directly receive their Tweets
1.3.6 Sentiment Analysis - the process of analyzing a statement and determining its degree of positivity or negativity (also known as sentiment)
1.3.7 HTML - HyperText Markup Language, a basic web page language
1.3.8 JavaScript - a language used in web sites
1.3.9 JSON - JavaScript Object Notation, the blocks of information returned from Twitter
1.3.10 jQuery - an extensive JavaScript library that simplifies Ajax interactions for rapid web development
1.3.11 Ajax - Asynchronous JavaScript and XML, part of the jQuery API, used to create asynchronous web applications which can send data to, or receive data from, a server asynchronously

1.4 References

Twitter: https://twitter.com/about
Alchemy: http://www.alchemyapi.com
Sentiment140: http://www.sentiment140.com/
Ajax: http://en.wikipedia.org/wiki/Ajax_(programming)

Average Number of Followers per Twitter User: http://www.beevolve.com/twitter-statistics/
JSON: http://www.json.org/
SRS: Political Mood Gauge Based on Social Media
SDD Template: www.jsu.edu/mcis/docs/SDDTemplate.doc (Jacksonville State University)

1.5 Overview

The remaining sections of this document are as follows:

- Section 2 Deployment Diagram provides a physical look at the system and provides additional information which describes the possible implementation of the system.
- Section 3 Architectural Design gives a detailed look at the individual modules within the system and how they all work together.
- Section 4 Data Structure Design looks at the individual classes within the application and briefly overviews their functionality.
- Section 5 Use Case Realizations describes the implementation of the use case scenario(s) from the SRS.
- Section 6 Real-Time Design discusses how the software will process data and display output real-time.
- Section 7 User Interface Design looks at the expected output for the software, briefly describing the intended content of the output web page.

2.0 Deployment Diagram

The software is intended to run on each user's machine. That is, it will be deployed after the user runs the executable and will display output locally per user. The software would however, be capable of running on a server in which the application would be executed on a host machine which would send all output to a project web page, accessible to any potential users. The latter deployment approach will be attempted if time constraints allow for it.

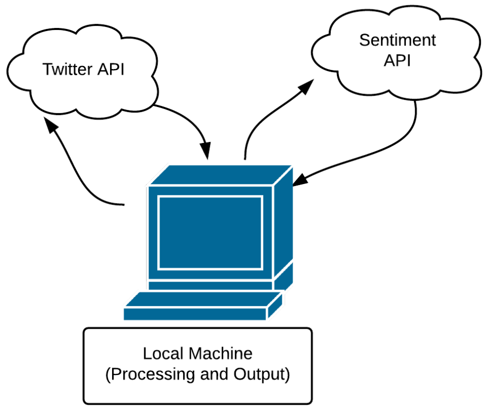

Figure 1: Option 1 - Single-User Deployment

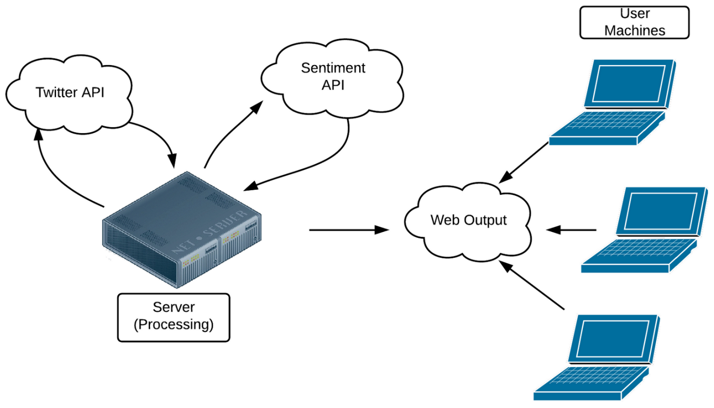

Figure 2: Option 2 - Multi-User (Server) Deployment

This document was written using a template provided by the Jacksonville State University MCIS department.

3.0 Architectural Design

3.1 Overview

The application has two parts. The first is a sequence of modules that does all the processing work, fetching tweets and calculating sentiment values. The second part is the output, primarily a gauge that displays the output from the module sequence.

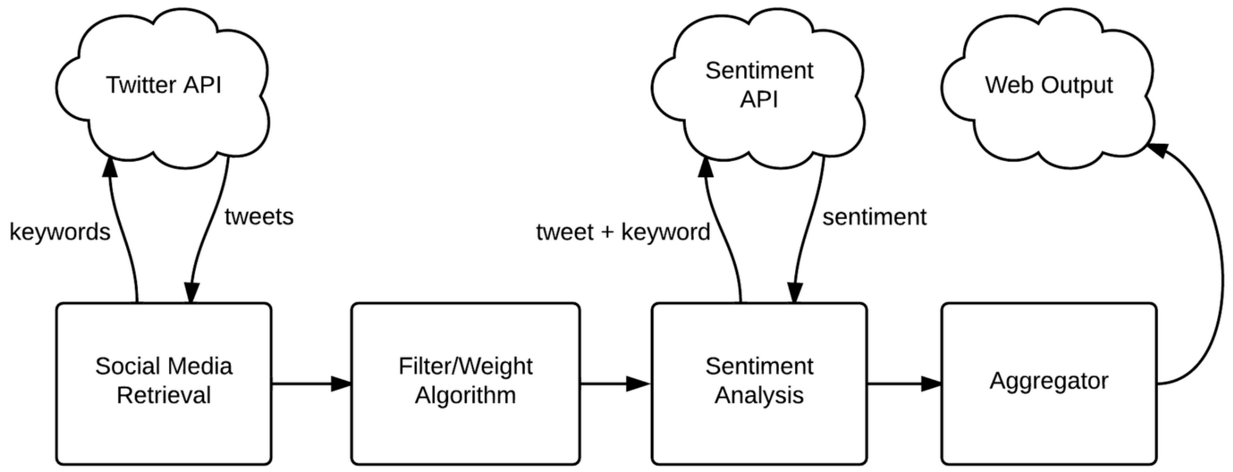

Figure 3: General Architecture Design

3.2 Modules

Each module is compiled from C++ code, and runs as a separate process, reading from stdin and printing to stdout. To run the entire module sequence, each module is piped to the next until the output is ready for the output. There are four distinct modules: social media retrieval (fetches tweets) the filtering/ weighting module (throws out useless tweets, determines the weight of the rest), the sentiment analysis module (gets sentiment for each tweet and combines with weight), and the aggregator module (sums the results for the latest tweets). As seen above, Figure 3 describes the dataflow of the software.

3.2.1 Social Media Retrieval

This module retrieves the tweets from Twitter. It fetches the tweets from the Twitter Streaming API, implemented with the twitter_stream class. The class provides a simple interface; provide a list of keywords to search for in tweets, and a callback function that takes a tweet object. This callback will be executed for each tweet that twitter sends back.

The twitter_stream class accesses the Twitter Streaming API with an HTTP request performed by the curl library. Once the stream is started, it makes a single long-lived connection, and incrementally receives the JSON data for each tweet. Once a tweet is retrieved, it is slightly reformatted and immediately written to stdout (in JSON format) to be piped to the next module.

3.2.2 Filtering/Weighting Algorithm

The filtering/weighting module filters out tweets that we do not want to process, and weights the remaining tweets based on quantifiable probable impact. The filtering part of this module removes tweets with any the following characteristics:

- written in a language other than English
- not containing requested keywords (occasionally the Twitter Streaming API lets one slip)
- contain multiple keywords (sentiment analysis becomes unreliable)
- have already been processed

The weighting part of this module is designed to weight the values of tweets based on their influence. We decided to measure the 'influence' of a tweet based on the number of followers the tweet reached - a characteristic of each tweet that is objective and quantifiable. However, the number of followers of the author of a tweet does not necessarily represent every potential reader of that tweet - if the tweet has been retweeted by another user (one of the original author's followers), then it has been sent to the second user's followers. Any tweet can be retweeted multiple times, and a retweet itself can be retweeted. Our basic approach to retweets is to estimate the number of additional twitter users who could have seen that tweet.

In the data sent back for each tweet, we can see whether or not a tweet is a retweet, how many followers the author has, and how many times this tweet has been retweeted. We do not receive the number of followers of the authors of each retweet, so we estimate. The average number of followers for a user on Twitter is 208 (source: Beevolve Social Media Analytics), so we multiply the the number of retweets by 208, and add the original number of followers (see equation below).

$$w = f + 208r$$

w: weight
f: number of followers
r: number of retweets
208: average number of followers per user

Unfortunately, if we handle each tweet separately this way, double-counting is possible. For instance: once we process one tweet with 10 retweets, we add 208 to the weight for each of those retweets, guessing at the number of followers of each author of those 10 retweets. But if we later receive one of those retweets and process it, then an extra 208 followers will have been counted. Additionally, we may find that the author of that retweet actually had 10,000 followers - so we extremely underestimated the weight of the first original tweet. To ensure that no double counting like this occurs, and to make sure we correct all incorrect estimations that we can, the following weighting algorithm will be applied:

This document was written using a template provided by the Jacksonville State University MCIS department.

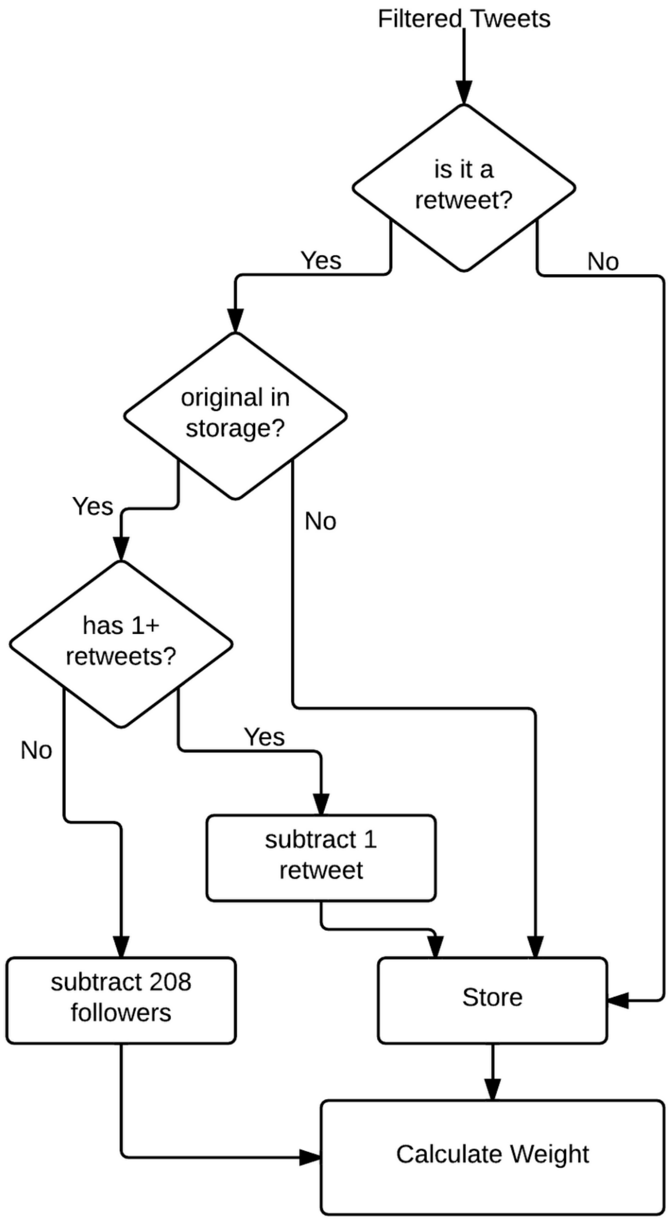

Figure 4: Weighting Algorithm

3.2.3 Sentiment Analysis

The sentiment analysis module sends tweet text and keywords out to the sentiment API. By using asynchronous curl requests, this module does not depend heavily on the sentiment analysis tool's throughput and speed. Rather than having to wait for each sentiment API HTTP request to return the sentiment of the given tweet, the asynchronous curl requests will allow the application to continually push tweets to the sentiment analysis tool, queueing up the tweets which are waiting for a sentiment to be returned from the sentiment API. In essence, the application won't have to wait for one request to finish before sending another. Once the sentiment value (a number between -1 and 1, ranging from

This document was written using a template provided by the Jacksonville State University MCIS department.

very negative to very positive respectively) is returned. If the sentiment of a tweet is negative toward a particular party, it will be changed to count as a positive sentiment toward the opposite party. For example a tweet with a sentiment of -0.5 for liberals, will be converted to a positive sentiment of 0.5 for conservative. This resulting positive sentiment is then multiplied by the weight value which was determined from the filtering/weighting module. This will allow us to make an easy calculation of the nation's mood in the aggregator module. The equation below describes the final value sent to the aggregator.

$$v = ws$$

v: final value
w: weight
s: sentiment (positive)

3.2.4 Aggregator

To drive the output of the gauge, we will aggregate the liberal/conservative values as we receive them from the sentiment module. Specifically, only the values from the last X-number of tweets will go into the gauge (where X is configurable by the user). To get a numerical aggregate/comparison from the last X-number of tweets, we sum the liberal and conservative values from each tweet respectively, then calculate a percentage (rounded to the nearest integer). The gauge takes liberal percentage and automatically calculates the conservative percentage (100 - liberal percentage). The equation for liberal percentage is below:

$$liberal\% = \frac{100 * \sum v_{liberal}}{\sum v_{liberal} + \sum v_{conservative}}$$

The aggregator sends along a JSON object that contains the calculated liberal percentage, and the number of tweets that have been processed. The next step is output.

To save and analyze data over longer periods of time, the current gauge value can be saved to a CSV at a set interval. The aggregator module will save this data (timestamp, liberal and conservative gauge values, and total positive/negative sentiment reflected in the gauge for each party) to the end of a CSV file whose name is the current date. The interval can be modified in the user configuration file, but the default save interval is 30 seconds. These CSV files will allow users to perform more broad and detailed analysis on their own if they desire.

One drawback to the simple gauge (representing the political mood of the latest X-number of tweets) is the fluctuation and seemingly inconsistent results it displays. To combat this, and present the user with more consistent daily trends, daily averages will be displayed in the web output as well. These daily statistics will consist of the average gauge values for the day, as well as the total number of tweets processed during the day. The aggregator module will save the most current daily statistics on the same time interval at which the CSV output is saved. To maintain and build upon the daily averages a recovery feature will need to be implemented.

The aggregator module will be able to recover daily processed-tweets counts and daily averages. If for some reason the user stops the program, or it terminates early through some error, it will continue

This document was written using a template provided by the Jacksonville State University MCIS department.

calculating the daily averages immediately where it left off. The detailed values will continue to be appended to the CSV file, tagged with a timestamp to show any gaps in the data.

3.3 Output

The format of the output is an html web page. There are two components that make up the interface between the web page and the aggregator - an updating script that writes the most recent liberal percentage value and total number of tweets processed to a file, and an AJAX request that runs from the browser to poll that file and update the page.

3.3.1 Update Script

The update script is a simple bash script that overwrites a file with the latest information (liberal percentage and number of tweets as passed from aggregator). This takes the burden of IO processing off of the aggregator. The update script can easily be modified to append each packet of information to a longer log file, so that it can be reviewed later.

3.3.2 AJAX

The AJAX request is run in JavaScript from the user's browser. The AJAX request reads the contents of the file that is being continuously updated and returns it to the JavaScript running on the web page. Given this new information, the web page will update with the latest information. One important note here is that if this is run on a single-user system (i.e. the processing and output are on the same machine) then AJAX requests may not always be accepted. For instance, Google Chrome does not accept AJAX requests for files on the same system as the browser. However, Firefox is happy to complete the requests. On a multi-user system (i.e. the processing happens on a server, and users access the output web page from their own machines) there will be no AJAX limitations.

This document was written using a template provided by the Jacksonville State University MCIS department.

4.0 Data Structure Design

4.1 JSON (JavaScript Object Notation)

JSON is a serialized format; any object and its data members (precluding binary data) can easily be expressed as text. For more explanation, see http://www.json.org/. Data from the Twitter API and the sentiment API HTTP requests is returned to our application via JSON; additionally, each module pipes its data to the next in JSON format. To decode JSON data, we use *libjsoncpp*.

4.2 tweet class

The purpose of the tweet class is to give each module an easy interface to the json data passed to it. As each module receives tweet JSON data, the tweet class packages it as a nice C++ class for interaction.

4.3 twitter_stream class

The twitter_stream class provides a C++ interface for the Twitter Streaming API. On construction, you provide the class with a callback function that takes a tweet object; the class will then initiate the HTTP request and call the callback (passing a tweet) whenever it gets data.

4.4 sentiment class

The sentiment class provides a C++ interface for the sentiment API. On construction, the class sets up an asynchronous http request interface, so that each time you request the sentiment toward a keyword in a tweet, the class fires off an async request. The class spits out the sentiment result for each individual request as soon as it is complete, without waiting for any others.

4.5 aggregation deque

The aggregation module sums the sentiment from the most recent X-number of tweets (where X is configured by the user) to drive the gauge output. The latest X tweets are stored in a deque (double ended queue). If there are fewer than X tweets in the deque, the aggregator adds each tweet it receives to the deque. After the deque holds X tweets, for each tweet the aggregator receives, it also pops off the oldest tweet.

4.6 processed_ids deque

This data structure will contain the id's of tweets that we have already processed. On occasion, the Twitter Streaming API may give us the same tweet twice. To ensure that no double counting will occur even in this circumstance, the deque of tweet id's will be searched for the id of each incoming tweet. The deque will only hold X tweets, the same amount as the aggregation deque - removing the oldest as it adds the newest.

This document was written using a template provided by the Jacksonville State University MCIS department.

5.0 Use Case Realizations (see SRS 3.4)

Given the simple output of the software, the only foreseeable use case for the product is to allow intrigued parties to view that output. There will be limited to no interaction with the output. The only possible interaction with the output will allow the user to click or hover on particular parts of a time-lapse graph of gauge values or map of the states in the nation to see the particular numeric results for that location/time. Bear in mind that the time-lapse graph and map output are both optional output types, that will only be implemented given adequate time.

This document was written using a template provided by the Jacksonville State University MCIS department.

6.0 Real-Time Design

6.1 Tweet Processing

Since the software is meant to provide a snapshot of the current political mood of the nation, the project must be designed in such a way that it can parse and process tweets real time. Using the Twitter API and curl, we plan to retrieve the most recent tweets that Twitter has to offer. After tweets have been retrieved from Twitter they will go through processing where they will be analyzed sentimentally.

6.2 Sentiment Analysis

Using the sentiment API and asynchronous curl requests, the software will send off the the tweets for sentiment analysis. As the sentiment for each tweet is being calculated by the sentiment API, the asynchronous curl requests will allow the software to send more sentiment requests to the sentiment analysis tool, building up a queue of tweets currently being processed. This will make the sentiment analysis processing very efficient, helping the software maintain a high throughput of tweets. As each sentiment is returned, it will be sent to an aggregation module which will calculate the the political mood of the most recent tweets, writing this information to an output file.

6.3 Output

As each tweet completes processing, the most current ratio of pro-liberal to pro-conservative tweets will be written to an output file. Using jQuery and Ajax (see glossary terms 1.3.10 and 1.3.11 respectively), the host web page will continuously read the latest values from the output file. These values will then be used to update a gauge which displays the results from the most recent X-number of tweets (where X is configured by the user) that have been processed. Another graph on the web page will show output trends over a more extended period of time. Finally, daily averages and tweet counts will also be updated. This will ensure that the output/results from the software are a real-time representation of the political mood of the nation.

A separate JSON file for logging can be overwritten by any module, containing information to be displayed in a notification box to the user on the webpage. This file will be polled using Ajax just like the output file above - but only twice per second.

This document was written using a template provided by the Jacksonville State University MCIS department.

7.0 User Interface Design

The user interface will be implemented through an HTML web page and consist of one or more output gauges. These gauges will display the results gleaned from processing tweets, determining sentiment analysis, computing tweet weight, and aggregating the most current political moods.

7.1 Graphic 1: Mood of the Nation Gauge (see SRS 3.1.3.1)

The mood gauge will be implemented using a third party JavaScript library called JustGage. The gauge is a semicircle with blue (liberal) on the left and red (conservative) on the right. If the liberal sentiment is high on Twitter, the dividing line between colors will be farther to the right - more blue than red. This gauge will have no user interaction possible.

7.2 Graphic 2: Mood of the Nation Time-Lapse (see SRS 3.1.3.2)

The time-lapse output is optional. If it is implemented, it will use a third party JavaScript library called Highcharts. The time-lapse graph will be a line graph, displaying the historic values from the above gauge at one minute intervals. The user will be able to hover or click on a point on the line to see the numeric value at that time.

7.3 Graphic 3: Mood of the Nation by State (see SRS 3.1.3.3)

The mood of the nation by state output is optional. If it is implemented, it will use a third party JavaScript library called Leaflet. It will be a map of the United States, conveying the political leaning of each state through the color of the state. Red will represent conservative, and blue will represent liberal. The saturation of the color will indicate how far in either direction the state leans; gray will indicate neutrality in that state (or a lack of data). The user will be able to hover or click on each state to see the numeric values of the sentiment for tweets originating in that state.

7.4 Graphic 4: Daily Statistics

Though not required, daily statistics will be shown on the webpage to give the user a better view of the general political mood of the day. There will be three numbers: the average liberal and conservative daily gauge percentages, and the total number of tweets processed that day. The number of tweets will constantly increment as the application is running, but the daily averages will only be updated every 30 seconds. This interval can be adjusted by the user in a configuration file.

7.5 Graphic 5: Notifications

Though not required, notifications of interest will be displayed for the user on the webpage. Currently, notifications will only be displayed in two cases: first, when a very popular tweet is processed (see weighting module) or when an error in the application occurs that the user should know about. The notifications will appear in the top left corner of the screen. The infrastructure is in place for these notifications, so more can be added in the future to display more information.

8.0 Help System Design
The software’s lack of user interface eliminates the need for an extensive help system. As a result, installation instructions will be clearly laid out and explained in the software’s README file. Descriptions of the gauges and other aspects of the web page and application configuration will be available in the user manual.

This document was written using a template provided by the Jacksonville State University MCIS department.