Twitter Cabinet

What if Twitter was your presidential cabinet?

Final Project Report

CS 4464 - Computational Journalism | Georgia Tech

April 29, 2016

Team Members

Hamilton Greene
Stephen Song

Table of Contents

Project Description

In January, Presidential Candidate Donald Trump and Fox News had a falling out around Trump's decision not to appear at a Presidential Debate [*Fox News*]. The candidate had polled his Twitter followers prior to the debate to determine whether he should attend. Feeling snubbed, a Fox spokeswoman said the following: "a nefarious source tells us that Trump has his own secret plan to replace the Cabinet with his Twitter followers to see if he should even go to those meetings."

Goal

We set out to survey the Twitter followers of each of the top four candidates to answer the question "What would the policies look like if each candidate replaced his/her presidential cabinet with his/her Twitter followers?"

Approach

We began by determining the basic components we needed to provide this survey of Twitter Followers:

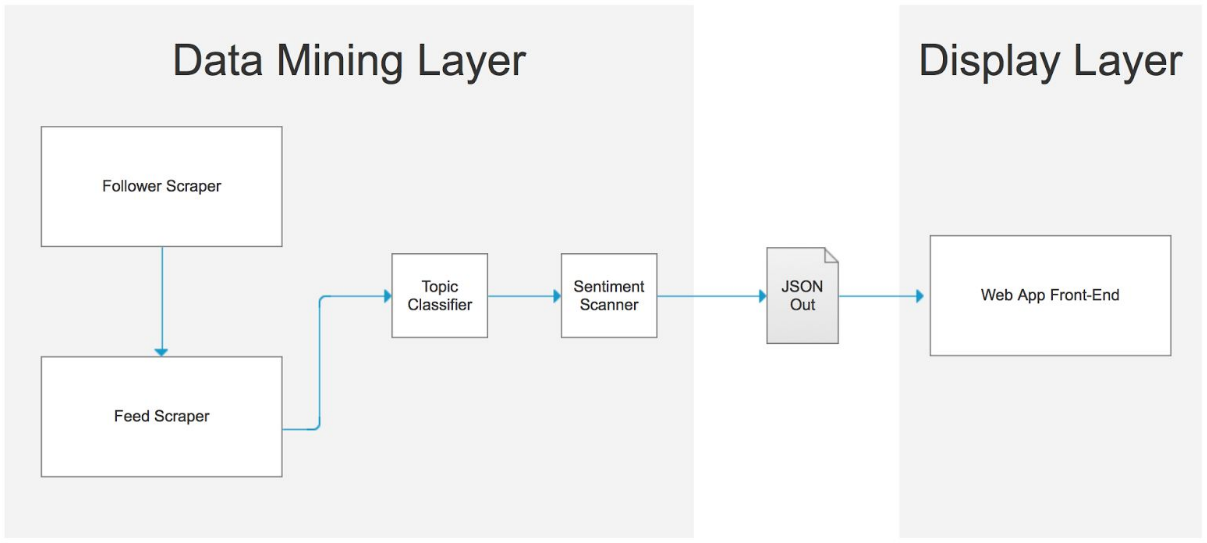
- Twitter followers of each candidate
- Feeds of each Twitter follower
- Classification of each Tweet by topic
- Sentiment analysis of each Tweet
- Aggregation of Tweet sentiment by topic and candidate
- Front end to display this data

We then split up the responsibilities so we could work in parallel and achieve a deeper understanding of the components within our domain. Hamilton did the backend data-gathering while Stephen did the frontend data aggregation.

The Twitter followers of each candidate were mined using the Twitter API. The feeds of each of these follower IDs were then mined through the API. The accumulated feeds were then run through a classification algorithm to tag each tweet with the most relevant topic. We then ran each tweet through a sentiment analysis function and saved the subsequent score with it. Finally, this data was loaded into the frontend for display to the end user.

System Architecture

Pipeline Overview



Our project runs as an asynchronous pipeline. The data gathering mechanisms aren't directly attached to the front end. Rather, a process is run which outputs data files and we then manually insert those files into the front end system.

Component Details

Each component within the Data Mining layer is written in Python 2.7.

Follower Scraper

Function: The Follower Scraper scrapes the Followers from a given Twitter Handle (see *Definitions*) using the followers/ids Twitter API call (see *Definitions*). Follower Scraper utilizes TweetPony (see *References*) to interface with the Twitter API.

Input: Takes in environment variables made available through a file named environment.py (an example file is available on the Twitter Miner GitHub repository).

Required inputs:
- *TWITTER_HANDLE*: The handle of the Twitter profile you are scraping followers from
- *TWITTER_CONSUMER_KEY*: The Twitter Consumer Key (see *Definitions*) you are using to access the API.

- *TWITTER_CONSUMER_SECRET*: The Twitter Consumer Secret (see *Definitions*) you are using to access the API.

Optional inputs:
- *TWITTER_ACCESS_TOKEN_KEY*: The Twitter Access Token Key (see *Definitions*) you are using for access to non-standard API calls.
- *TWITTER_ACCESS_TOKEN_SECRET*: The Twitter Access Token Secret (see *Definitions*) you are using to access non-standard API calls.
- *Existing Mining Operation***:** At startup, Follower Scraper will search for an existing file named TWITTER_HANDLE + 'Followers.json'. If one is found, it will attempt to continue the mining operation where the previous one left off.

Output*:* File with a JSON object stored on each line (so entire file is not valid JSON).

The fields in the output file are:
- *nextCursor*: The ID of the next page of follower Twitter IDs (see *Definitions*) as returned by the Twitter API (see *Definitions: Cursors*). This is useful in the event that a failure occurs and we need to restart where we left off. It also serves as a sort of unique ID for each row and is used as such in the Feed Scraper.
- *twitterIDs*: Holds all the Twitter IDs gathered in that particular API call as an array .

Running: To run FollowerScraper.py, run command:

`python FollowerScraper.py`

Comments*:*
- *Formatting*: The odd formatting is to make it easier to read in and push out growing files. With file sizes this large, storing it all in memory then writing out with proper JSON would be problematic. With this setup, we can easily restart a mining operation after a failure or begin reading in a mining operation while it is currently running.

Feed Scraper

Function: The Feed Scraper parses through the output of Follower Scraper (a file named TWITTER_HANDLE + 'Followers.json') and hits Twitter APIs user_timeline endpoint (see *Definitions: Twitter API*). Follower Scraper utilizes TweetPony to interface with the Twitter API.

Input: Uses same environment.py file as Follower Scraper.
Required Inputs:

- *TWITTER_HANDLE*: See *Follower Scraper*.

- *TWITTER_CONSUMER_KEY*: See *Follower Scraper*.
- *TWITTER_CONSUMER_SECRET*: See *Follower Scraper*.

Optional inputs:
- *TWITTER_ACCESS_TOKEN_KEY*: See *Follower Scraper*.
- *TWITTER_ACCESS_TOKEN_SECRET*: See *Follower Scraper*.
- *Existing Mining Operation***:** At startup, Feed Scraper will search for an existing file named TWITTER_HANDLE + Feed.json'. If one is found, it will attempt to continue the mining operation where the previous one left off. The *Comment* section has more details.

Output*:* File with new JSON object on each line.

The keys in the JSON object are:
- *feed*: The entire returned feed with each tweet stored as an array of form [Tweet text, TweetID].
- *nextCursor*: The nextCursor ID present on the Follower Scraper output file. This is stored to aid in operation restart.
- *twitterID*: The unique Twitter ID that the timeline was pulled from

Running: *`python feedScraper.py`*

Comments*:*
- *Noise Filtering:* As a preliminary noise filter, flags are set to exclude retweets and replies to help ensure we only get tweets created by the user. We also drop any user timelines that come back with 0 tweets.
- *Restarting paused operation:* On startup, Feed Scraper will first check for the existence of a *Followers* file, then for the existence of an existing *Feeds* file of form TWITTER_HANDLE + 'Feeds.json'. If one is found, Feed Scraper will attempt to resume mining where the last one left off by matching the nextCursor ID with the one in the *Followers* file then by skipping each ID in the *feeds* field until a matching ID is found. Upon successfully finding the last ID, mining resumes as usual from the next ID in the array.

Topic Classifier

Function: The Topic Classifier parses through the Feed Scraper output and classifies each tweet found therein into one of several categories. It currently uses a simple word-matching algorithm to determine category.

Input: Takes in a the output file from Feed Scraper.

You must also provide the name of the file to read from and the name of the file you want to write to.

Example: python topicClassifier.py realDonaldTrumpFeeds.json -w realDonaldTrumpTopics.json

Output: A file with a JSON object on every line

The keys in each JSON object are:

- *topic*: The topic the Classifier matched the tweet with.
- *text*: The text of the tweet.
- *feedID*: The Twitter ID of the user/feed the tweet was pulled from.
- *tweetID*: The unique Tweet ID (see *Definitions*) of the specific tweet.

Running: python topicClassifier.py FEEDFILENAME.json -w TOPICOUTFILENAME.json

Comments:

- *Topic Curation*: In an attempt to lessen the impact of our personal biases on the topics we decided were important in the political arena, we used ISideWith's Popular Polls section (see *References*) to choose Issues that others thought were popular. We then tried to choose key words that could be found within each Issue's summary blurb.

Sentiment Scanner

Function: Sentiment Scanner runs through each of the tweets output from Topic Classifier and assigns a sentiment value to it using the VADER sentiment analysis library (see *References*).

Input: Takes in the output file from Topic Classifier.

You must also provide the names of the file to read in and write out to.

Example: python sentimentScanner.py realDonaldTrumpTopics.json -w realDonaldTrumpSentiment.json

You may also set a --pretty flag which will write the output in prettified JSON (valid notation). This helps in the ingest stage of many frontend JSON parsers.

Output: Outputs the exact same thing as Topic Classifier, but with an appended *sentiment* key which holds VADERs analysis of the respective tweet text.

The VADER analysis is in the form of:

- neg : [0, 1] value of negativity
- neu: [0,1] value of relative neutrality
- pos : [0,1] value of relative positivity

- Compound : [0,1] value of total sentiment

Running: python sentimentScanner.py TOPICFILENAME.json -w SENTIMENTFILENAME.json

Front-End

The front-end of the app was built with HTML, CSS, JavaScript, jQuery, and Semantic UI. The processing and filtering of the tweets and their sentiments are done client-side in JavaScript, as much of the data scraping was already performed and exported to the JSONs. The app calls $.getJSON() to process each of the JSON objects per candidate and loads it into the page.

There are two pages in the app: index.html (the home page) and result.html (the results). The URI parameter appended to result.html is used to run the analysis and generate the relevant visualization (e.g. result.html?topic=Obamacare).

Several Semantic UI components we used include their icon sets, progress bars, and loaders. While the framework provided us with HTML elements that could easily be modified and set via JavaScript, we customized the components in styles.css and script.js to fit our needs.

Data Description

The data we dealt with consisted of the Twitter followers of given Twitter handles and the tweets present on their timeline.

Project Achievements

We were able to successfully style and implement the front-end app for our project. By using Javascript, jQuery, and Semantic UI, topics and data can be procedurally displayed and expanded upon. We wanted to design an app that could be published as a journalistic artifact like several of the other visualizations and tools we saw during the semester, and from a visual design standpoint, we were able to do so.

Process Reflection

MongoDB

We started out using MongoDB to store all the data on our server. We chose it because it was easy to use (especially with Python) and it would allow us to process JSON relatively easily. Unfortunately, we were unable to get it to run continuously over large periods of time. The maximum time it was up seemed to be around 2 days before encountering a problem of some kind, requiring a manual restart. It started to become a hassle, but we kept up with it in an attempt to save us from refactoring the entire system. Unfortunately, the last time it went down, there wasn’t enough space on disk to repair its data, pretty much forcing drastic measures. At this point, we decided to move over to simple file creation.

JavaScript Mining

We started off mining with a Node.js app. It worked fine for Followers, but as we got deeper into the Feed Scraper logic, we ended up in an endless asynchronous callback stack. This is probably due to poor planning at the outset for asynchronous calls, but it was frustrating enough to make us switch all the mining over to Python. A greater understanding of the language and its features would probably have helped us circumvent such an issue.

Limitations and Future Work

The number of tweets in our data set served as a limitation for the scope of this project. Because there are millions of Twitter followers per presidential candidate, it would take months to gather enough tweets to paint a complete picture of the followers on the site. Additionally, it would take even longer to gather enough tweets per follower to be able to run an accurate sentiment analysis.

Another problem is that there is a problem with the selection of tweets. Many Twitter users have thousands of tweets, and in order to get a good indication of their sentiment towards a certain topic, you would have to go through a majority of those tweets to obtain a complete picture. One could design the system to dynamically parse through the data set prior to running, but the issue of scale can still be encountered.

The filtering of tweets and determining which ones should be used posed an interesting question. When we first ran the experiment, many of the most positive tweets were either one or two word tweets or emoticon-only tweets that scored really high in sentiment. However, a lot of these do not have meaning or any information relevant to the topic at hand, so the issue we encountered was how to identify which tweets should actually contribute to the score.

Lastly, the sentiment analysis only provided one dimension of emotion, as many sentiment analysis engines (including the one we used) rely on a positive, neutral, or negative scale. However, positive and negative scales are not regular enough to gain an overall picture of each follower. In our prototypes, we had additional emotions such as sarcasm, enthusiasm, or criticism. While there are sentiment analysis algorithms that do have multiple dimensions of emotion, many of those are expensive or are available only to research institutions. Perhaps someday in the future, a consumer distribution of these algorithms could be used in this project.

Code

Twitter Cabinet code can be found at: https://github.com/songstephen/twitter-cabinet

Demoing Twitter Cabinet

The part of our project that was created specifically for end users is the front end. If you load up the front end code on a local server with appropriate JSON files in the data directory, the project should work fine.

As for the Python mining code, simply running the python scripts through a terminal then showing off the output files is probably your best bet.

Definitions

Terms

Cursors - Twitter uses cursors as a form of result pagination. Not all results will fit into one request, so you must use cursors to tell Twitter which page you want to get results from next. If you don't, you'll likely receive either the same results every time or just the newest ones - depending on the rate of activity of the given feed. To read more about cursors, see *Using cursors to navigate collections*.
Tweet ID - The unique numerical ID given to each tweet on the platform.
Twitter Handle - The unique plaintext, human-readable username of a Twitter profile. Not to be confused with a Twitter ID. Example Twitter Handle: @realDonaldTrump
Twitter ID - The unique numerical ID given to each Twitter user. Example TwitterID: 3891842652

Twitter API

API Request - A request is a call to the API. It doesn't matter what, if anything, is returned.A request should be assumed to be counted against your usage limits even if a null object is returned.
Rate Limits - Limits imposed by Twitter to regulate API usage. There are separate limits for app authentication and user authentication. Limits are set based on call, so exceeding your limit on statuses/user_timeline won't affect your limit on followers/ids. Your allotment of calls refreshes every unit time, which is currently set to 15 minutes. For more info, take a look at Twitter's Rate Limits Chart.

Request Types

followers/ids - The followers/ids endpoint returns up to 5,000 of the given user's follower IDs. The rate limit is set to 15 requests/unit time.
statuses/user_timeline - The statuses/user_timeline endpoint returns up to 3,200 of the given user's most recent tweets. You may set flags to exclude replies and retweets. Rate limits are set to 300 requests/unit time.

References

Fox News issues incredible response to Donald Trump's Twitter poll about going to the Fox debate. Colin Campbell. Jan 26, 2016. Business Insider. http://www.businessinsider.com/fox-news-donald-trump-debate-megyn-kelly-twitter-2016-1

ISideWith - Popular Political Issues. A list of popular issues on a polling site. https://www.isidewith.com/polls/popular

TweetPony. A Twitter API wrapper for Python .https://github.com/Mezgrman/TweetPony

VADER Sentiment Analysis. A Sentiment Analysis library for Python. https://github.com/cjhutto/vaderSentiment