Birds Nest

Senior Project

Jared Stern

Spring 2017

Contents

1 Topic 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1.1 | Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |  |  |
| 1.2 | Motivation (optional) . . . . . . . . . . . . . . . . . . . . . . . . | 2 |  |  |
| 1.3 | Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |  |  |
| 1.4 | Scope . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |  |  |

|  |  |  |
| --- | --- | --- |
| 2 | Pro ject Breakdown  2.1 Architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  2.2 DataAnalysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  2.3 Supporting Images . . . . . . . . . . . . . . . . . . . . . . . . . . | 3  5  5  7 |
| 3 | Challenges & Takeaway | 8 |
| 4 | Future | 9 |
| 5 | Additional Notes | 10 |

6 References

1 Topic

Discussion of Data Mining, Text Scraping and other research tools used to implement a programmatic search of text Tweets on the Twitter Social Media Platform.

*Database Appications: Data Mining; Information Search and Retrieval; Search Process*

1.1 Problem

This project was intended to provide a historical record of a given tweet, looking for its likely origin by tracking down the earliest posted version of its text, image or video. In its final form, the project implements a simple weighting algorithm to provide a list of most similar tweets organized by time.

1.2 Motivation

The primary motivation was to curtail the phenomenon known among twitter users as ‘stolen tweets,’ which is the act of reposting the text/photo content of another person’s tweet, without providing attribution or in other ways indicating original authorship.

In addition, the idea of having a tool like this for historical or entertainment purposes seemed immensely useful, to track a trend or ‘meme’ which may have originated on Twitter.

1.3 Description

The program takes as input, any publicly visible tweet on the Twitter platform (Linked to by simple HTML.) It then uses open source text-scraping projects as a tool to gather all similar tweets, working from the beginning of Twitter to the present. An exact match to the original query stops the process (representing either the first version of the tweet, or the version being queried), and all collected Tweets are sorted by age in a local database.

1.4 Scope

The project was a deliberate decision to work entirely within the constraints of available systems and tools, using them only as a means to implement my search logic programmatically. For reasons that will be discussed later, it will not search for photos and videos, and does not, as of this moment implement any text weight beyond direct matches (w/ wildcards).

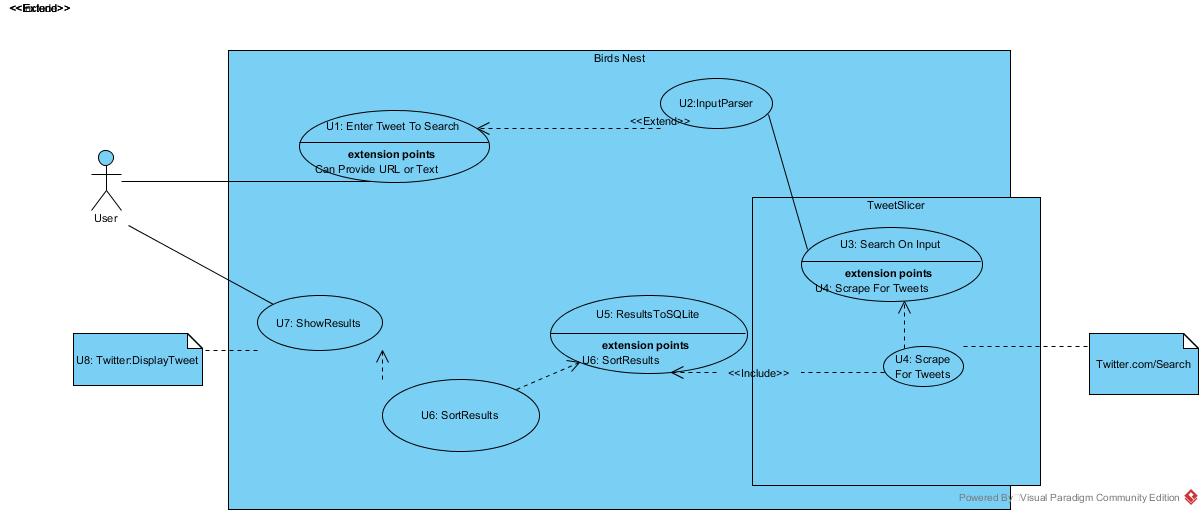
2 Pro ject Breakdown

2.1 Architecture

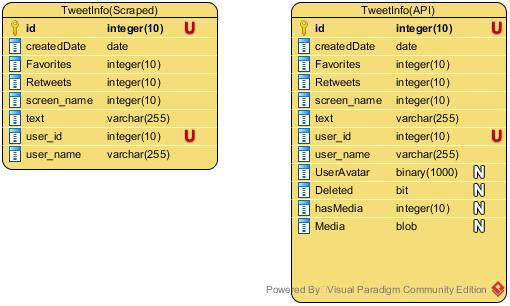
The program is a combination of the following systems and subsystems.

* Twitter is the basis for all input: Tweets are retrieved in one of two ways
  + Multiple Tweets: Python-Based Text-Scraping: See Below.
  + Single Tweet for Display: Twitter API/Embedding
* TweetScraper, authored by Tom Dickenson[1]. Uses HTML-GET queries to produce an HTML readable page with the results of a simple text search using Twitter’s user-visible search page. Returns a JSON Array, and implements a simple search, or a date-restricted search.
* tmhOAuth, <https://github.com/themattharris/tmhOAuth>, a PHP library to handle calls to Twitter’s API.
* PHP Handler Pages to accept a search term/tweet as input.
* Apache2 Webserver to allow for local testing of PHP Pages.
* Ubuntu Virtual Machine, run using VMware.
* SQLite Database, for results storage.

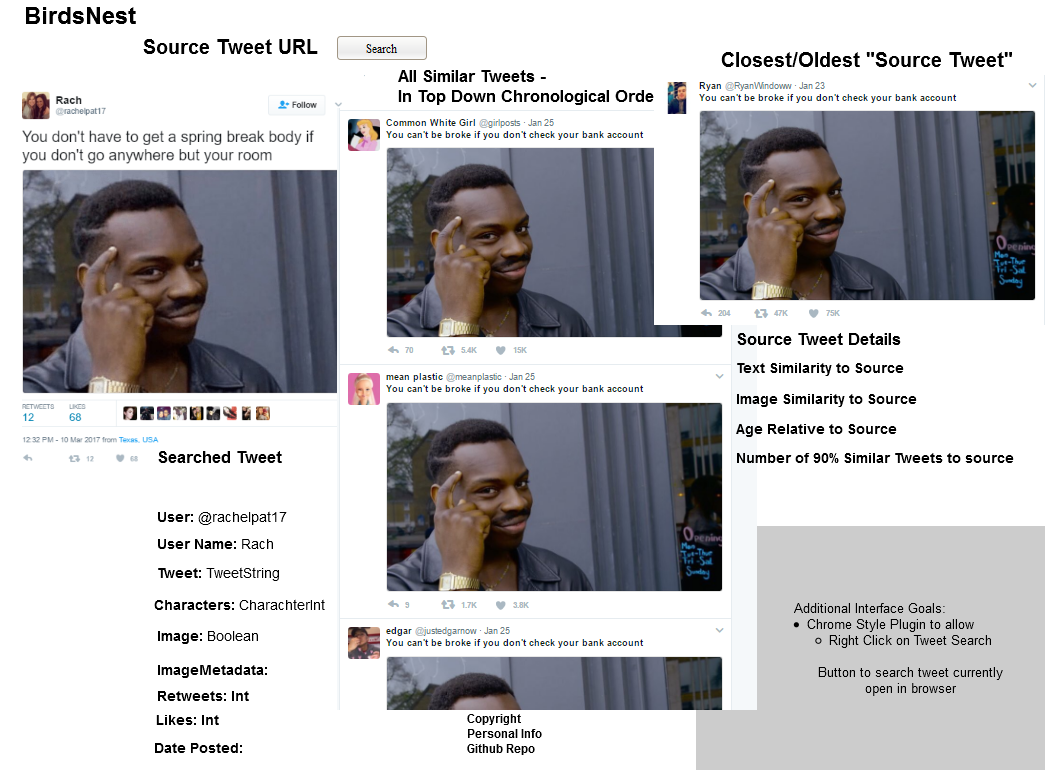
2.2 Data Analysis

Use Case Diagram

Tweet Data Fields By Source

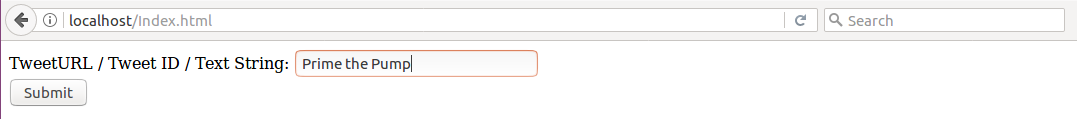
.

2.3 Supporting Images

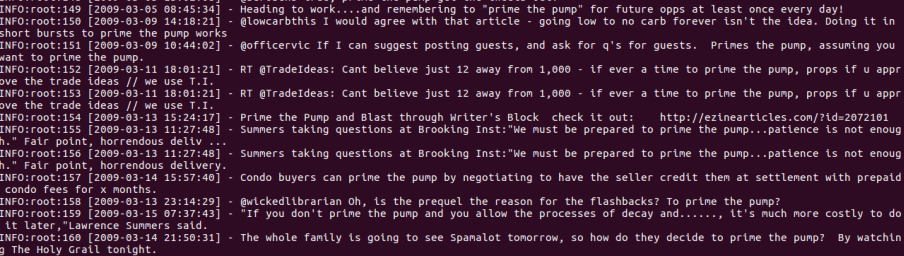
Early Prototype

Sample Use Case:

President Trump claims he invented the phrase “Priming the Pump.” Ignoring the fact that the phrase was coined in the 1800s, we use Bird’s Nest to find the first use of the phrase on Twitter.

First, landing page. The phrase isn’t associated with a given tweet, so this is passed to the Python Script on the command line, which uploads all of the results from Twitter from the first year of Twitter’s existence into an SQLite table.

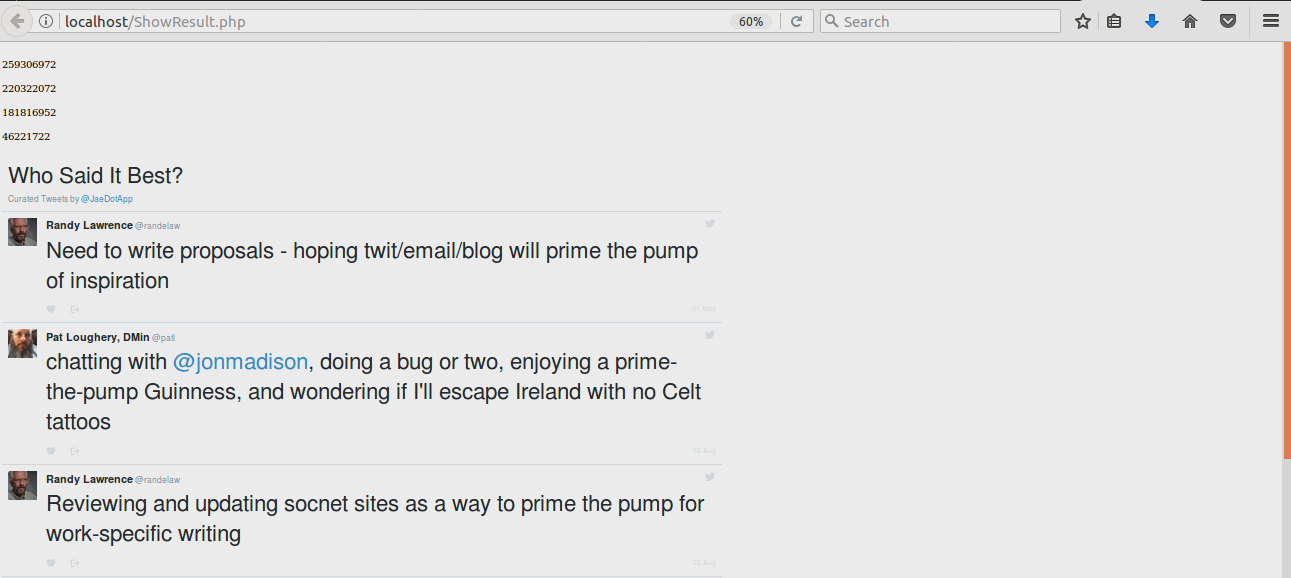
SQL Commands tell us to find non-case sensitive [\*prime the pump\*]. We can also do this in Twitter as well, but since future iterations will want to be more complex, we’ll eat the time-cost and perform the operation on our end. Also helps catch garbage returned by twitter: duplicate responses by the same user with the same timecode are hard to eliminate during the scraping, easy once they’re in a DB.



SQlite Table, Containing Results Subset. In addition to the other values extracted by the program, additional info can be obtained by querying the Twitter API with the Tweet ID. This is rate-limited, and all data it provides is shown in the results screen with the Tweet Preview Widget.

Order SQL by date, where the Tweet contains the exact phrase within its contents.

Using the TweetID, PHP returns all valid results from oldest to newest. Adding all valid tweets to a collection, and then displaying it. To show these next to more complex data results, simply iterating over the collection works. The top lines show the valid tweet-ids in result set, sorted by oldest to newest, and the bottom underneath is the default display of results.



3 Challenges & Takeaway

Due to the various changes and challenges in my project, the final program, in my view serves as more of a functional prototype of work I might do in the future, to be iterated, extended and improved upon. A brief synopsis of my work, and the challenges I faced during it, follow.

My first few weeks’ worth of work on the project were primarily based on initial research papers. None of them were written from anything approaching a lay perspective, some were designed entirely around the solution of more complex problems than I needed to address (DNA mapping,) and others were implemented in ways more complex than I could ever imagine doing on my own or in the time required. While they made enjoyable reading, and I plan to refer to them for future projects, there was little directly applicable. I’ve often struggled with narrowing the focus of my research and projects, and unfortunately this seemed to have been no exception.

Realizing the necessity of some means to narrow my focus, I spent the next few weeks looking into Twitter’s storage of images and videos. As the primary use-case for my project was artists and other creatives looking to find unsourced and unauthorized use of their work, this was the next step. The first portion was a dive into Twitter’s front end: all photos uploaded to the service are stripped of all their metadata, and assigned a new arbitrary title name. This meant I had to go to an outside service.

The next two weeks involved generating test-searches based on images I would find and repost to a fake account, and then test 3rd party image search services against them. Twitter itself offered no such service, even to pro-users, Google proved to be too imprecise, and TinEye, which proved successful on finding images over 5000 RTs, locked programmatic access to their search behind a hefty paywall. Videos and images proved another dead end.

Resolving myself to text search only, my early research indicated that the complex ‘weighting’ algorithm I was thinking of using was a project’s worth of work in and of itself, and was not strictly achievable in the time alone. I would grab the earliest exact match: or, if time and implementation allowed, a simple scoring mechanism based on the number of distinct words contained in my results.

The next attempt was building a program entirely in PHP using Twitter’s search API. Sample queries were found to work effectively, text results were returned in easily parse able forms, and even the challenge of understanding and implementing Twitter’s OAuth authentication for developers was well in-hand. This, of course, is when I discovered that Twitter both rate-limits API requests, and keeps its historical tweet data behind a paywall. Historical meaning, older than seven days.

With little more than six weeks left, and all of my early solutions coming up short, I decided to implement someone else’s open source project: Twitter’s web-search after all, contained all the results I needed, in an order I could use: I simply didn’t know how to get them off the page at once. So, I reviewed a number of different open-source projects hosted on personal sites or online repositories like GitHub. Many were simple implementations of the Twitter API, making them useless to me. Other were dumb re-hosting of Twitter web-content, which showed me how Tweet information was available directly on the HTML source, but not how to manipulate the search page so that all of those results were visible.

Then, the challenge became reading someone else’s program in order to manipulate it. I’d used Python before, but never with a program of such complexity, and never directly from a webpage. I decided to use encapsulation: I would throw the Python script the text string it needed to run from the PHP handler, have the Python program execute its internal logic (changed to fit the patterns of behavior I was looking for) and blindly load those results into an SQLite Database.

The PHP handler page would then resume logical control, connect to the same SQLite file, (when time allowed, perform all internal logic sorting and considerations,) and display results in another PHP page.

Other challenges involved technical implementations: early prototypes on an AWS server vanished after an issue with my free student account, and PHP configuration options, including simple matters like ‘finding out if Curl is enabled’ proved frustrating as well. In future projects, I will be sure to thoroughly consider both my test environment, and regular backups of my project, using version control services like Git, and online repositories like GitHub.

4 Future

There is plenty I want to and can do with this in the future. The first step would be to make an official fork of TweetScraper to support Parallel scraping by default, and implement an actual to-file class. With a large database of sample tweets to work from, I’d also like to get some experience with data classification, and implement a precise text weighing method to allow for more complex cases, which might include variations on tweets with changed proper nouns, or cases where it is the structure, rather than the words themselves that are being changes.

With a means to grab and store Twitter data, another not-too far off possibility is the creation of a tracker for ‘stolen’ art. By implementing the service as a widget, I could easily keep track of frequently requested users, which would suggest a pattern of behavior: users obviously don’t believe that this person is the creator of the items they post.

My experience working with an official API has been invaluable as well: while they can seem all-powerful, looking at constraints, and considering if they are actually capable of suiting your needs is a step I won’t be skipping again.

While it’s harder to apply directly, this was also my first time working with another programmer’s codebase: I’m surprisingly excited to do something similar again.

In terms of developing Birds Nest as a product or project, I want to consider what it brings to the table as a unique product in its own right. A work-around or automation of Twitter’s search function is only the first step, and is really only viable so long as I’m the product’s primary user. To that end, I’m probably going to put Bird’s Nest up on the shelf, and work primarily on a local database of collected tweets.

5 Additional Notes

While, ultimately, I feel I was only able to partially meet the goals of my project, I feel that ‘failure’ was largely a result in an incomplete understanding of what would be needed to accomplish the goals of my project, and an unwillingness to discard more complicated portions of the project until they had already incurred a time cost too drastic to recover from.

However, I do not feel that this project was in any way a waste. While the fruits of my efforts were unable to manifest in the project directly, I made great strides in both my understanding and practice of various complex programming tasks I would be expected to perform in the future. I had to read and understand a complex system, and spent significant time using and testing APIs, and can confidently do that again in the future. My time spent guiding students in the Code Samurai tutoring program paid dividends when I was forced to read through several inconsistently documented projects to discover which would be of use to me, and then implement (and in some cases debug) those programs.

**6 References**

**[1]** **Tomkdickinson. 2017. tomkdickinson/Twitter-Search-API-Python. (March 2017). Retrieved May 12, 2017 from** <https://github.com/tomkdickinson/Twitter-Search-API-Python>  
  
**Themattharris. 2014. themattharris/tmhOAuth. (August 2014). Retrieved May 12, 2017 from https://github.com/themattharris/tmhOAuth**

**2016. Scraping Tweets Directly from Twitters Search Page – Part 1. (December 2016). Retrieved May 12, 2017 from http://tomkdickinson.co.uk/2015/01/scraping-tweets-directly-from-twitters-search-page-part-1/**

**Amihood Amir. 2010. *Combinatorial pattern matching: 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21 - 23, 2010; proceedings*, Berlin: Springer.**

**Florin Gorunescu. 2011. *Data mining concepts, models and techniques*, Berlin: Springer.**

**Min Song and Yi-Fang Wu. 2009. *Handbook of research on text and Web mining technologies*, Hershey, PA: IGI Global (701 E. Chocolate Avenue, Hershey, Pennsylvania, 17033, USA).**

**Allen Zeng, Joe Germuska, Steve Tarzia, Rich Gordon, Josh Shi, and Alex Duner. 2014. A beginner's guide to collecting Twitter data (and a bit of web scraping). (March 2014). Retrieved May 12, 2017 from https://knightlab.northwestern.edu/2014/03/15/a-beginners-guide-to-collecting-twitter-data-and-a-bit-of-web-scraping/**