Janine Heiser
Info257
Dec 27, 2013

Tubes Trends- Trending the Tubes

Description of Database

A current online trend is that websites and social media sites display "trending" data about users who are using these the site or social network; its possible to go online and see what users across the wolrd are talking and thinking about. Website "trending" data typically can be described as a high number of user searches or mentions of particular person, place, thing, or idea on a given search engine or social network. Most recently, on August 30, 2013, Facebook released a new feature called, "Trending" that appeared in the homepage news feed of its website. This new feature, appears to mimic Twitter's "trends box", which highlights popular hashtag and topics on the twitter platform.[1] The concept of a keeping track of user searches and user mentions of particular people, places, things, or ideas was started over 5 years ago by Google in August 2008, which eventually evolved into Google Trend. Although widely used websites have "trending" data, there is no website or entity on the internet that is aggregating this data across multiple websites[2] For my database, I created a database that stores trending" data from Google Hot Trends, Twitter, Instagram and Youtube. Trends do not to not exist in a vacuum; the internet and mass media help move ideas and products to many countries and geographical places. Taken alone, individual trends could come and go and might not seem significant, , however, if one could map trends onto a map by geographical areas, larger patterns world patterns (human behavior and otherwise) and regional patterns might emerge where if one was watching for them.

Purpose of Database

My database about internet "trending" data from major web platforms and social networks hopes to the following purposes:

- to create an database model of world-trends that give gives a way to both see trends occur in

1 http://blogs.wsj.com/digits/2013/08/30/facebook-tests-trending-section-in-news-feed/
2

real time (instantaneous) and think about trends in the long-term (over a course of a week, month, year, etc)

- map that trending data to the geographical coordinates of latitude and longitude how the data pieces geographically relate to each other;
- provide a way for users to compare and contrast what's "trending" on different search platforms and social networks to see if there are common themes
- show the duration of time that items/topics/people/ideas/places/stories are popular/trending on the Internet
- to illuminate a potential "zeitgeist" that may or may not exist on the internet or in the world.

Sources of Data

For this project, my ultimate ambition was to visualize the data; I had hoped to map the trends to places on Geo-cordinates. Having this end goal in mind, I focused my energy on obtaining trending data from sources that provided it for at least several countries. To obtain the data, I gathered it from a web API platform or I scraped it off the website directly. Below lists the APIs and websites from where I gathered data from:

Twitter → http://www.twitter.com → social networking platform

- **API resource, GET trends/available**
- Returns the locations that Twitter has trending topic information for.
- The location is returned in a woeid (yahoo where on earth id)
- When I did this project, the API returned at total of 415 locations.
- EX of API resource, GET https://api.twitter.com/1.1/trends/available.json
- http://dev.twitter.com/docs/api/1.1/get/trends/available

- **API resource, GET trends/place**
- Each API call returns the top 10 trending topics for a a specific yahoo WOEID that trending data is available for.
- trends/place data is refreshed by twitter every 5 minutes, but users can only make 15 calls to this API in a 15 minute period
- Since this API is rate limited, users have to authenticate when they make API calls.
- **Required parameters**:

    - WOEID (The Yahoo! Where On Earth ID of a location)
- EX of API resource call: GET https://api.twitter.com/1.1/trends/place.json?id=1
- http://dev.twitter.com/docs/api/1.1/get/trends/place

Google Hot Trends → http://www.google.com/trends/hottrends **→ search engine site**
- Google releases the number of people that searching for different topics using Google search engines
- On the google hot trend website, google publishes the hot trend information for the following 8 countries: USA, UK, Australia, Canada, Germany, Hong Kong, India, Israel, Japan, Russia, Singapore, Taiwan
- Google only makes the US hottrends data available for via an RSS feed at http://www.google.com/trends/hottrends/atom/hourly; no other countries are availible there
- I was able to get the trend data for the other 7 countries by writing a ruby script that scraped the Google hot trend web site.
- Google updates their data about what's trending in differnet countries about every 1-5 hours, depending on the country.

Youtube → http://www.youtube.com **→ video search engine**
- **API resource, GET Most Popular**
- API returns the feed of the most popular videos for the past day
- Also can return the most popular videos for a particular geographical region for the past day:
- The most popular videos for each day is available for over over 39 countries.
  EX of API resource call: https://gdata.youtube.com/feeds/api/standardfeeds/most_popular
- EX of API resource call: http://gdata.youtube.com/feeds/api/standardfeeds/*regionID*/most_popular?v=2
- http://developers.google.com/youtube/2.0/developers_guide_protocol_video_feeds

Instagram → Instagram.com → photo sharing social networking platform
- **API resource, GET Most Popular**
- This API gets a list of what media is most popular at the moment. Can return mix of image and video types. API will also return the latitude and longitude coordinates of popular item when availible.

- EX of API resource call: https://api.instagram.com/v1/media/popular?access_token=ACCESS-TOKEN
- http://instagram.com/developer/endpoints/media/
- Instagram updates this API with new data at different rates; when the information is released is more sporadic than google hot trends or twitter.

Yahoo Where On Earth Id → Geographical/Mapping Data Source → http://developer.yahoo.com/geo/geoplanet/guide/concepts.html

- A yahoo Where On Earth ID (*WOEID*). Is a unique identifier that is assigned geographical entities on earth; this id is never changed; it acts as unique identifier for a specific place on earth.
- Using the yahoo WOEID API, you can retrieve the latitude and longitude, as well as other geographical information for a specific place. To call the WOEID API:
- 'http://where.yahooapis.com/v1/place/'woeid?[developerapikey]
- The pieces of trend dating that I collected had to normally related; when taken directly from the data source, that could relate pieces of trending data from different sources to one another. To solve the problem of normally relating the data, I used the yahoo woeid to map the data to a specific geographic place.

Implementation of the Data Collection and Deliverables

General Design:

To gather the data, mentioned above, I wrote ruby scripts that collected the trend data, parsed the trend data, and then inserted the data into my database. From an amazon ec2, server, I deployed these the ruby scripts to run at regular intervals using the cron-tab. Below, I list a few details about how I implemented of my database system. Additionally, all the scripts for this project were turned in and referenced below, in case you'd like to see how they were exactly implemented.

Server

for my project, I didn't think I could use the harbinger server because I need root access to run cron jobs continuously. As a result, I deployed a an amazon ec2 ubuntu 12.10 server instance. Specifically, I set up a dual LAMP stack, as well Nginx and Ruby on Rails. As a deliverable, I created a shell script that does this whole process automatically. The file is called **setup_ruby_stack_aws_ubuntu12.sh.**

Setting up the Geography/Modeling relation:

In a ruby script, I called the twitter, GET trends/available API to get a list of 415 places; these 415 places are distributed through out the world and also are countries. In the same ruby script, after getting the 415 places, I sent them to the Yahoo woeid API to get more information about them, like what there names were, the latitude and longitude coordinate, etc. Then, I parsed the returned yahoo woeid API results, and inserted the location data into the database. For each place, yahoo has a hierarchy of places embedded in the json, it returns, for instance, country, state town. Because of this hierarchy, I had to normalize the places table, and separate out the countries woeids from the other locations. To this I ran some sql queries that found the country woeids and then put them in a separate country table. Normalizing the location data required more effort upfront, because I had to separate out the data, but in the long run, separating locations on country/location allowed me to make specific queries more easily/more efficiently when I called API services that only had country data. When I called these APIs, my query only had to look through the country table, instead looking through a list/table of all the locations.

To parse the data, I found a few ruby gems that were particularly helpful. These gems are:

the **json gem**: → this parsed the jason that API's returned.

Nokogiri → this gem also parses xml and html elements.

The nokogiri gem is probably my favorite gem that I used for this project. It has the ability to search documents via XPath or CSS3 selectors and really minimizes the amount of regular expressions you have to use to find the data you're looking for.

To see the exact process I used to set up get the twitter locations, send them to the yahoo api serve and then insert them into the database, please refer to the scripts:

twitter_aval_places.rb , twitter_aval_places.rb , tubestrends_dbschema_part2.sql , tubestrends_dbschemaWithNormalization.sql

Getting Twitter Trend Data:

Using the list of 415 places listed in the country and places table, I grabbed the twitter trends data using the twitter **GET trends/place** API, and then parsed the json that was returned and inserted it into the database. One thing to note is that twitter limits the number of API calls you can make per hour; specifically you can make 15 API calls every 15 minutes, or a rate of 1 api call per minute. As a result, I had to be strategic about how I automated this job. For 415 woeids, it takes about 7 hours for the script to grab all the trends for all the places. That being said, there were 8 countries that Google published trend for hourly. To have an apples to apples comparison, of google to twitter, I scheduled the the script, get **twitter_current_trends_now.rb** to run at the top of every hour; meanwhile the **twitter_get_tr_restoworld.rb** checks to see what time it is, and if its ten minutes after the hour, then it runs to go out and get trends. Using timing mechanisms like this, in addition to using the cron tab allowed me to automate the process so that I could collect twitter trend data for different locations 24 hours a day/ continuously, at a rate of 1 trend per minute → 1 min. Another difficultly that I will mention is that in addition to rate limits, twitter also requires API users to authenticate using a process called Oauth; basically you have to give the Twitter API your key and then they issue you a token to make the API call. The process of figuring out how to authenticate with twitter was a difficult for me; prior to this project, I had no experience using or making API calls in a script or in a web application, so there was a steep learning curve. Other things to note are: the **twitter_get_tr_topcountries.rb** and **twitter_get_tr_restoworld.rb scripts** reference the following class/script files

twitter_get_trends.rb twitter_current_trends_now.rb twitter_get_tr_restoworld.rb

twitter_get_all_trends.rb

Getting Google Hot Trends Data

As mentioned before, Google only makes the US hottrends data available for via an RSS feed. As a result, I had directly scrape the website to get the data However, the hot trends for each country: UK, Australia, Canada, Germany, Hong Kong, India, Israel, Japan, Russia, Singapore, Taiwan are all on "different pages" pages on the website, and the actual hot trend information is rendered in javascript to the user's browser. Since the country page hot trend data is generated by javascript, you can't just scape the page. To get around this issue, I figured out a way to created an instance of Firefox in a shell instance, and then call the google hot trend website for each country from the browser. Calling the website in the shell browser allowed me to the obtain the HTML for the country hot trend website. Ruby has an amazing gem library called **watir**; watir was actually written for selenium/automated testing. Then, another amazing ruby gem that I used is called **headless.** Headless is an add on to watir. Basically, headless takes an instance of watir, and then runs an instance of Firefox in the shell, hence the name headless, for headless browser window. Because headless can run Firefox in the shell, it tricks websites/web servers into rendering the javascript and outputting HTML for the page. Once the website has rendered the HTML, you can use a gem like nokogiri to grab the HTML from the headless browser window and then parse the information you need out of it. Using headless and watir gems, I was able essentially trick Google into giving me the hot trends pages for UK, Australia, Canada, Germany, Hong Kong, India, Israel, Japan, Russia, Singapore, Taiwan. Additionally, to continuously get the hot trends data, I set up an crontab job. This job ran every hour. Google posts a banner saying the last time that the hot trends have been updated. As a result, within in the get hottrends script, the script checks to see if the hot trends have been updated in the last hour. If has, then script runs, but if not, the script takes not action. To see the exact specifics of all the implementations mentioned above, look at the script, **google_get_hot_trends.rb**

Instagram and Youtube Popular Data

To get instagram popular data from the instagram API, you have to authenticate with the Instagram API with a key, but the process is a more simple than Twitter's authentication process. Upon authenticating with the Instagram API, the API provides the popular data as a json object that you can easily parse. The Youtube API serves the popular data via a RSS feed; so to get the data, you simply call the API are returned with an atom_xml object that can be easily parsed with an xml parser
to see the exact way things were implemented, please refer to the scripts: **instagram_get_popular.rb and get_youtube_popular_feeds.rb**

Challenges During Implementation

This project was extremely challenging for me to implement; I ran into many challenges. The challenges were so great that I didn't have a change to turn in assignment #4, because the complexity of the task I was trying to accomplish. Some challenges that I faced were:
Coming into the project I did not know any ruby, although, I had done a significant amount of PHP development in the past. I chose to implement this project in ruby because I wanted to learn it, and there nothing better than learning by doing. Although it was my choice to do this in project in Ruby, it made the project a lot more difficult for me, because I had to learn a new language as I went. Another challenge that I faced was learning how APIs worked; prior to this project, I had never made calls to an API server in any previous program that I wrote. As a result, I had to learn how APIs work, how one would authenticate with the API service, and then how to parse all the json and XML that gets returned from APIs. Another problem that I ran into was that, at one point int the project, I managed to loose my my private key to my amazon cloud ec2 server instance. Without a private key, you can't access the amazon server instance- as in you're locked out forever. I spent about a day and a half troubleshooting , trying to figure out how to get into my amazon ec2 server without my private key. After all that, I couldn't find anyway to get around not having a private key, so I started up a new server instance. Upon

starting up a new server, I had to run all my set up scripts, my database set up scripts, etc, as in it was huge time sucker. Finally another problem that took long to solve was how to run ruby scripts via the cron tab.

It is recommended that ruby be run through the ruby rvm (ruby version manager). This is because there are lots of ruby libraries that are supported in different versions. The ruby rvm lets you run multiple versions of ruby simultaneously, so you have access to all the gems. Unfortunately, the rvm information/ ruby environment variables (aka the place where ruby gems/libraries are loaded from) for each user are stored in the user's bash shell. This presented a problem when I tried to run ruby scripts as crontabs job. Crontab jobs are run by the root user; the root user did not have the same ruby environmental variables my user did in its bash shell, so the job failed when the crontab ran because the root user could not find the ruby/gems libraries. It took me a long time how to first, figure out why the jobs were failing and then second, solve the problem and finally get it to work so that the crontab can run to the jobs.

To solve this problem you acutally just give the crontab a direct path the ruby gem library that you scripts use, for instance:

```
54 * * * * /usr/local/rvm/bin/ruby-1.9.3-
p484/home/ubuntu/final_project/getdata_scripts/google_get_hot_trends.rb
```

User Interface

The ruby implementation of an MVC is rails. The rails syntax to create a MVC is very different than the PHP implementation of an MVC in that you have to create models for each table/database item that you want to access in the controller. In rails, there is no simple way to make sql calls, you have to use models. (maybe there is a way to do it, but I couldn't figure it out in time) Ultimately, I could not figure out the syntax of how to create and use rails models. I spent many hours trying to figure it out, but was unable to. As a result, I couldn't create a MVC for my database in the allotted time. To compensate for my lack of user interface, I wrote three reports that show the types of things that my MVC would show. These reports show how the database can be use to:

1. Show, in the past 5 days, what and in what countries have Google and twitter had similar trend items

2. Find the 35 highest trending/most talked items in the database

3.Get Twitter Trending Data for specific days/times for specific cities

These excel reports are entitled:

Tubes_trends_report2.xls, tubes_trends_example_report1.xls, Tubes_Trends_Report3.xls

I also included the sql I used to generate these reports: I also included the sql I used to generate these:

queries_for_reports.sql

The reports contain the full results that I have in my database, and hopefully show that I've successively implemented my database. Additionally, if I had more time to work on this project, for the MVC interface, I had also hoped to map the place/country trend I that I had collected onto a map, via the Google maps API, using the latitude and longitude data for each location that I had collected. If I had more time to work on this project, I am sure that I could have figured out how to use rails models, create an MVC and map the trends data onto a map, but in the alloted time, it didn't happen. Despite my lack of a web UI, I did go to great efforts to make a system that automatically collects, transforms, and load data into a database (ETL). I am hoping that my success in creating an ETL tool will make up for my lack of web UI.

Conclusions

My project demonstrates how an ETL tool can be implemented using ruby, and shows how its possible to collect trending data that's publish online and make it organized and usable in a database. Working on this project was challenging, but ultimately rewarding. In fact, I enjoyed working on this project so much that I am planning to trying to get the the Rails MVC working over winter break/into the future.