CS 410 Final Project
Sushrut Vani (vani)
Shaleen Agrawal (sagrawal)

Documentation

Introduction

Media is all around us, and it shapes the way we view the world as well as our values and desires. We are all influenced by media every single day, without fail, from TV advertisements, to music, movies, TV shows, billboards, the internet, newspapers, magazines and many other mediums through which media is delivered. With media having such a profound impact on many different people from different backgrounds, it's important to recognize the influence media has and explore the ramifications of that influence. .

One of the most influential types of media in the modern day is the internet. Whether it's social media, news articles, or any other kind of internet content, the internet has a lot of information. This information isn't regulated in any way, and thus any ideas can become popular if they gain enough traction. We can utilize our unprecedented access to the way people think today by exploring patterns in internet data to uncover trends. We can even help identify biases among certain communities or websites to help people get a better understanding of where they can receive objective, unbiased opinions.

So, enough about what we gain out of analyzing social media content, how do we approach this problem? Well, using sentiment analysis tools we can make it extremely simple for users with no technical expertise to leverage our technology and better understand the content they are consuming. We built a chrome extension which allows people to query a variety of internet sources and yields results about how positive or negative the comments were, as well as the relative objectivity present in the statements being made.

Implementation Overview

Our app is built on two main sources, the frontend is build as a chrome extension through which the user can send queries and the backend is built as a heroku app using py flask. Basically, the way the application works is that we get what the user wants to query from the chrome extension and then send it to the heroku app. From there, we can return the results of the specified query and display them to the user via the chrome extension. For graphing, we use the chrome extension to send users to the heroku app, where we can graph more detailed results to queries.

Basically, the chrome extension communicates with the app using GET requests and then listening for the response before parsing it and giving it to the user. We send parameters to the heroku app by adding them to the url for the app, similar to a REST system. However, all of this is handled internally, so the user doesn't have to worry about navigating to the correct URL or formatting the URL in a correct fashion. Instead, the user can simply specify a query with the options given, press a button, and wait for the results.

The chrome extension uses html buttons and parses the inputs given by the user and then formats the GET request with a URL that the Heroku app can then use to determine the parameters that are required and utilize them in the requested queries. We use request listeners to wait for a response, and as soon as we receive the response from the heroku server, we can immediately give our users the results.

The heroku app is where the bulk of the implementation details are. After we receive a query, depending on which type of query was chosen we go into a specific function which resolves that query and gives us back the results. The app is built entirely in python and utilizes vaderSentiment, textBlob, and NLTK in order to process and analyze the data. If the user sends a twitter query, TweePy is used to find tweets with the specified search term included before we then analyze the text of the tweet using the aforementioned libraries. Meanwhile, if a user wants to query Reddit instead, the praw library is used to execute the desired query before we analyze the resulting data. After we complete the analysis of the data, plot.ly is used to surface graphs on our server, which we can then access from the chrome extension by redirecting users to the server.

Function Overview

a. **Chrome Extension**

For the chrome extension, very little in the way of actual calculation is done, and as a result, there are very few functions. All of the scripts attached with a particular functionality of the extension have a create request function in order to create the request that needs to be sent to the server. These functions all are fairly similar, only including some error checking based on what query the user is attempting to send.

There is also a reqListener function in most of these auxiliary files which is invoked when the extension receives a response from the GET request that was sent to the app. This function simply takes the data received and presents it to the user.

Finally, there is a popup.js file which has a simple script that shows the user if they are highlighting any text on the page within the chrome extension. This allows the user to verify the text that they are sending to the server when they choose to use the site analysis feature through which they can analyze the sentiment of any site or article they visit just by highlighting the text.

b. **Heroku App**

This is where all the real work is being done, and as a result it contains most of the major functions which do the heavy lifting. There are some helper methods which help with dealing with when certain optional parameters are not specified, but the key methods that are more than one or two lines are described in detail below.

run_reddit(keyword, subreddit, max_comments)

This function takes three parameters, the keyword to search for, the subreddit to analyze, and the number of comments to analyze total. The idea of this function is to go through and find as many posts relating to the keyword as possible. If they are related, then we go through all the top comments, and we keep analyzing comments and collecting data till the max_comments parameter is reached. At this point, we calculate how many comments were positive and negative, calculate our overall subjectivity, and then return the data to the user.

graph_routine(input_param, subreddit, num_comments, num_posts, time_period)

This function is very similar to the run_reddit function, but it provides more versatility and is suitable for more in depth queries where we want to know more than simple polarity numbers. We allow the extra parameters num_posts and time_period, and now num_comments refers to how many comments we want to look at per post rather than an overall count of comments like max_comments was in the previous function. This function also allows users to graph sentiment data over time. For each post that fits the user's criteria, we analyze num_comments amount of posts, and then we record the net ratio. Then for however many posts as was specified, we plot the ratio in relation to the date the post was created. The time_period parameter allows us to take advantage of the reddit API's ability to let us access posts within a certain time frame, so if we want to look at all time posts we can do that, or if we want to limit our search to more recent events, the user has that option as well. This function also utilizes the helper method index, which helps return the graphs and inject them into an html page so that we can display them to the user.

twitter(keyword, max_comments)

Again, this function provides very similar functionality to the reddit functions, but now we are analyzing Twitter data instead of reddit data. As a result, we no longer need a subreddit to specify, and we can simply search for max_comments amount of tweets with the specified keyword included. We use TweePy to find the tweets and then a combination of Vader and TextBlob to analyze the data, just like before. After we gather the specified amount of tweets, we can calculate the total positivity and negativity scores along with the subjectivity score and send it back to the user.

csite_call_api(input_string)

This function is a little bit different because it allows users to specify something to analyze. Maybe they are reading an article and they feel like it is biased or they have some other data that they would like to analyze. Here, we simply take an input string, which the user has provided via highlighting it on their chrome, and analyze it after splitting the string into sentences. We then calculate the metrics as we have done for the other functions and return them to the user.

default_graphs()

This function surfaces a graph of the latest simple query ran by the user. If they received a sentiment score and objectivity score in the chrome extension display, the user has the option of heading over to our default graphing url via a button on the extension to see a visualization of those results, in order to make it easier to parse.

Usage

The main functionality of the app is as follows:
In order to use the app, simply select which query you would like to use, and then fill in the required fields and hit the corresponding button.

Reddit:
- Simple Query
    - Search Term (key to search for, optional)
    - Subreddit (subreddit to search through, required)
    - Sample Size (total comments to analyze, required)
- Graphing Query
    - Search Term (key to search for, required)
    - Subreddit (subreddit to search through, required)
    - Sample Size (comments to analyze per post, required)
    - Posts to look at (total posts to analyze, required)
    - Time period (time_period to analyze, required)
        - Options are day, month, year, all time

Twitter
- Simple Query
    - Search Term (key to search for, required)
    - Sample Size (number of tweets to analyze, required)

Any Site
- Simple Query
    - Highlighted Text (text must be selected when you press the button)
    - Note: if the text is selected it will appear above the analyze button
    - Note: You have to refresh the page once after refreshing the extension for this to work as intended

Bugs/Things To Work On

Large queries or keywords that rarely occur can occasionally cause the server to time out, not much that can be done about this due to limitations of the API. The baseline graphing server can be a little buggy at times, refreshing the page usually fixes this. As mentioned above, highlighting text in order to execute a desired query won't work until a new page is loaded after refreshing the extension. This is due to the script needing to be injected to the page.

Github link for project:

- Chrome Extension: github.com/honestly-funny/410Backend
- Heroku Server: github.com/honestly-funny/bucket_api_heroku