Jacob Facemire

COSC 645

Professor Huang

December 11th, 2022

Project Report

My initial project was to use web scraping and natural language processing to pull news articles and evaluate the tone the authors took using sentiment analysis. Using this, I set out to create a database of articles which could be analyzed to evaluate the bias present in news reporting. In addition, I initially wanted to collect comments left by users and analyze those to see if there were any correlations between the two. After scraping the data, the script would dynamically create SQL statements to insert the data into a SQL database. A separate script would pull the necessary data from the database, perform the sentiment analysis, and import the findings back into the same table for analysis. Once all of the data is complete, an analyst could utilize premade software such as MySQL workbench in order to analyze the data.

With this database a user could create dashboards and other visualizations to perform high level data analysis on media trends. This sort of project is valuable for understanding the relationship between media, political figures, controversies, and voters, and is something researchers have done before (Morris, 2005). One of the reasons I chose this type of project is because prior to my degree in computer science, I was a political science major who worked as an intern for various political action organizations. In these roles, I analyzed trends in populations and media as a data science intern. For this reason, using new technologies I studied in this program and applying them to similar topics I’ve studied in the past felt like a natural fit.

For this project I chose Python, not only because it is known as a powerful language for data analysis, but because I actually have less experience using it compared to other languages, as such I wanted to use this project to gain experience with it. In my research I found a number of libraries to accomplish this goal, beautifulsoup4 and spaCy. Beautifulsoup4, or bs4 is a simple, popular web scraping library that I read was easy for newcomers. "spaCy" is a natural language processing library I learned in this class and used in multiple assignments. For the database, the data is stored using MySQL that is hosted locally on my personal machine. For testing and analysis purposes, MySQL Workbench is the tool used for manually accessing with the database.

My current implementation uses five scripts, sixteen csv files for importing URLs, organized in a simple file structure. Four of the five scripts are meant for scraping data from specific websites. This is done because each website has a different HTML structure, and each script needs to be adjusted to work properly. I originally chose four news sources, however for reasons that will be discussed later, those original sources were changed to include The Washington Post, CNN, NPR, and FOX.

Before running any of these scripts, the user will need to collect a list of URLs which can be in the same “category”. The category they belong to should be the name of the file. For example, a grouping of articles about climate change could be called “climate.csv”. Once this is done, the URLs need to be stored as a CSV file in the directory “[root]\sources\[Website]”. Upon running any of these scripts, the user in presented with instructions, first the user needs to input the names of any CSV files to use for scraping. The user can input multiple file names and denotes when they are done by entering a “0” into the script.

The scripts first take all of these names and stores them in an array which it iterates through using a for loop. For each file, the script calls a function called “import\_url”. This function sets up two variables, first the relative path to the CSV file being used using the provided table name and second, a global array used for storing the text of the articles that are scraped. Then the script sets up the database connection. The name of the file imported, which is the category the contents belong in, becomes the table it is stored in using a “CREATE IF NOT EXISTS” statement. This means for any new table, any of the web scraping scripts can be ran in any order and the first one will set up the new table properly. In order to do this the script uses an empty table in the database called “stock\_table”. The new table copies the structure of this table using the “LIKE stock\_table” phrase instead of specifying column names and values. This makes the code simpler and easier to read. Then the contents are stored into an array using the split() function in python. After the scripts iterates through every URL and calls a separate web scraping function.

The web scraping function first uses the requests library to connect to the provided URL. Then it creates a bs4 object named “soup” which stores the HTML source for the entire webpage. The text of the article is stored in an array using the find\_all() method in bs4. The text is found using the name of the HTML class the in for is stored in. For example, the body of articles for most Washington Post stories is stored in the HTML class “article-body”. As such, the code to pull only text in that class from the webpage is: “text = soup.find\_all(class\_='article-body')”. The headline is stored as well by searching for the “h1” tag in the HTML code.

The body of the article is initially stored as an array. Using a for loop, the entire body is concatenated into one string stored as “articleBody”. Because articles often use apostrophes, which can break the SQL code, escape characters are added using built in python functions. A dictionary is made that relates the “ ‘ “ character (space added for clarity) to the new text “ \’ ”. This dictionary is defined using the string function “maketrans()” and is called on to clean the body of the article using the following code: “clean = str.maketrans({" ' ": r" \' "}); articleBody = articleBody.translate(clean)”. If the scraping is succesfull, the url, headline, and body are stored are appended as an array to the global two-dimensional array initialized in the import\_url function. If the scraping is not successful (tested by checking if the length of the article body variable is zero) a message is displayed to the user and the article is not added to the array. This process is repeated for every URL in the file, then the information is imported to the database using the SQL statement:

"INSERT into " + table + " (Headline, Source, Link, Body) VALUES ('" + textAr[i][1]+"', 'WaPo', '" + textAr[i][0] + "', '" + textAr[i][2] + "')"

Between scripts, the “Source” string is changed to whatever source the script is built to scrape. The “table” is the name of the CSV file imported. If the user included more than one file to use to scrape, this process is repeated for every file.

After all of the scraping scripts are ran, the analysis is performed by a separate script “sentiment.py”. This script first prompts the user to input then names of as many tables they want to analyze which is stored in an array. This array is iterated through, and each table name is passed into a function named analyze(). This function first creates a connection to the database and pulls the body of each article stored using the SQL statement “ ‘SELECT Body FROM ’ + table”. Then the script sets up the natural language processing by initializing the language processing model using the “en\_coure\_web\_sm” pipeline that’s already part of the spaCy library. In addition, another pipeline component “spacytextblob” is added as well. This allows the model to perform the sentiment analysis on the text. The script goes through every item it pulled form the database, performs the analysis and updates the record to include the polarity rating in the “Body\_Polarity” column. The table structures include an “article\_num” column that is an autoincrementing unique primary key. This means that the first article the script pulls will always have an “article\_num” of one, the second of two, and so on due to the way SQL returns values from the tables. Thus, the updates are able to easily insert the data in a for loop using the query:

"UPDATE " + table + " SET Body\_Polarity = " + str(doc.\_.blob.polarity) + " WHERE article\_num = " + str(i+1)

The polarity value is cast as a string in python to allow for easier concatenation, however SQL reads it as a float and thus has no issues importing the data. This process is repeated for every table name that is given.

Throughout this project there was a few issues. First, collecting articles took a bit longer than expected, as many mainstream news websites do not have comment sections. Even still, some sites had comments enabled on only some articles, and some articles simply had a comments section, but no comments at all, or so few comments analyzing them would not provide meaningful number. Of the sites that still have comments enabled and do have active comments section paywall their content, leading to me purchasing multiple subscriptions to complete the curation process.

Initially I chose the Washington Post, the New York Times, Fox, and Mirror, all of which had comments sections and represented a wide range of political biases as measured by the ad fontes media group (Interactive Media Bias Chart, 2022). Of these websites, two had issues which lead to them being replaced. The script for the Mirror initially worked, however after a few test runs of scraping articles the script was blocked, returning an error 403 with every attempt to connect. I was unable to find away around this problem. The New York Times as well implements a system that only allows users to read so many free articles before requiring a subscription. At first I was able to scrape articles, but I was eventually prevented from doing so, with requests only returning the HTML for a JavaScript pop up and not the article itself. For these reasons both of these sites were replaced with CNN and NPR.

In addition, scraping comments on sites I could connect to was challenging. Comments sections are dynamic and were, in every case I observed for this project, handled with JavaScript. After researching ways to pull the JavaScript, including connecting directly to the URL of where the comments data is stored, I learned that beautifulsoup4 works by pulling the initial HTML response to the quest and pulling data from that, making it unable to parse any JavaScript elements. Further research led me to believe a different library, Selenium could be used instead. However, at this point in the project, most of the scraping code was written and the sentiment analysis code as well. Unable to find a solution I could deploy in time I unfortunately left out the comments scraping part of the project. Although I failed to account for the issues of handling JavaScript in web pages, and thus could not pull comments, the bulk of my project, which involved setting up a database and scripts that interact with it to process large text files in bulk, was successful.

Developing this project taught me a lot about the data mining space, and the various tools which are used. In particular, I learned about multiple web scraping python libraries, how they work, and what each can and cannot do. In addition, I was able to build on my knowledge of natural language processing which was developed over this course and further implement those technologies with databases technologies. In addition, prior to this class the bulk of my programming experience was in web languages such as PHP, and Java for local programming, as such choosing to work in Python was in and of itself a valuable experience, and I learned about the syntax and uses for another language.

This project builds upon a large basis of work being done in understanding news media. Understanding the information people read is paramount to understanding how they think, feel, and vote about issues. Many authors have studied the relationships between media bias and current issues such as political polarization, (Bernhardt, Krasa, and Polborn 2008) electoral outcomes (Druckman and Parkin, 2005), and views on political issues (Baum and Gussin, 2005). With a more robust implementation, this project can expediate the data collection process, allowing machine learning models to process significantly more articles than any human researcher, and create statistics which can be used to generate insightful analysis. This tool can quickly capture general trends within news coverage, and a framework for a web scraping and data importing script which can be quickly retooled to work with other websites.

These techniques are applicable to many fields of both business and academics. Market research on a large scale requires sentiment analysis of social media posts, news coverage, and more. Understanding how people feel about a subject is paramount to trying to influence them, whether those people are customers or voters. A search for these tools such as “Selenium” on job sites such as LinkedIn results in various postings for engineers with this kind of experience. In the future, I would like to build on what I have, develop a new scraping tool with Selenium to gather more information locked behind JavaScript.

References

Morris, Jonathan S. *Slanted Objectivity? Perceived Media Bias, Cable News Exposure, and Political Attitudes*. 5 July 2007, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6237.2007.00479.x.

“Interactive Media Bias Chart.” *Ad Fontes Media*, 8 Nov. 2022, https://adfontesmedia.com/interactive-media-bias-chart/.

Dan Bernhardt, Stefan Krasa, Mattias Polborn, Political polarization and the electoral effects of

media bias, Journal of Public Economics, Volume 92, Issues 5–6, 2008, Pages 1092-1104, ISSN 0047-2727, <https://doi.org/10.1016/j.jpubeco.2008.01.006>.

Druckman, James N., and Michael Parkin. “The Impact of Media Bias: How Editorial Slant Affects Voters: The Journal of Politics: Vol 67, No 4.” *The University of Chicago Press Journals*, The Journal of Politics, 1 Nov. 2005, https://www.journals.uchicago.edu/doi/abs/10.1111/j.1468-2508.2005.00349.x.

Baum, Matthew A, and Phil Gussin. *Issue Bias: How Issue Coverage and Media Bias Affect Voter Perceptions of Elections*. https://citeseerx.ist.psu.edu/advanced\_search.