Text Mining Project - Web Crawler for Orpheus

Ryan Garland and Sara Edwards

July 2023

1 Problem Formulation

Our project's aim is to help collect news articles for a framework named ORPHEUS. ORPHEUS is a decision support framework that shall support public health practitioners and responders in guiding individuals, groups, communities, and populations out of the chaos brought upon a geographic region by natural or human-made disasters. To mount an effective response, it is imperative that regional public health departments continuously plan mitigation strategies for a variety of hazards that may affect a region. Regional hazards include natural (e.g., flooding, hurricanes, droughts, earthquakes, wildfires) and human-made (e.g., bio-terrorism, industrial accidents, nuclear explosions). Planning authorities need to swiftly prioritize regional hazards and determine the most effective mitigation strategies to address them. While data-driven tools that can aid regional hazard prioritization have been developed in recent years, we witness a distinct lack of such data-driven tools capable of also facilitating the design of effective response plans to address prioritized hazards.

2 Our approach

To assist in this initiative, we want to develop a web crawler capable of gathering and analyzing the content of local news articles depicting hazards impacting a given area. Our project uses the Google search API to gather URLs of storm articles and then applies the k-means clustering algorithm to see if the articles can be organized by type of storm/hazard. The articles are collected by state and the goal is to collect 100 URLs per state to have 5000 articles total. The web crawler application will allow us to analyze articles through the use of text mining and natural language processing techniques to further rank the likelihood of certain hazards occurring in the area.

2.1 Methodology

2.1.1 Data Gathering

First, we had to figure out what custom search API was the best to use given the amount of data we were looking to query and which would allow us to get the data for free. The official Google custom search API has a data limit of 100 requests per day to stay in the free tier. This would have required us to pull data every day for 50 days to get the desired amount, so we went with an unofficial source that utilized the same engine but allowed us to make more requests. The broker is named apilayer and they allow for 1000 requests per month with each request containing 10 search results.

After finding a suitable API, we created a Jupyter notebook that iterates through each state name of the U.S. states and queries the "state name + 'storm article'" to ensure enough new articles are gathered. We stored all the search result URLs in a dictionary with the state name as the key and a list of URLs as the value. Then, we used the Python library Beautiful Soup to download web pages as HTMLs, and stored the title, author, publication, body text, and URL for each page in a Pandas library dataframe object.

2.1.2 Data Cleaning

After creating the dataframe containing information in text format for each extracted and downloaded article, we proceeded to clean our data to ensure Part of Speech tagging can easily identify and separate words into tokens. All steps were performed within the '*part_two.ipynb*' file within the given folder and also found in our public repository. First, we performed checks on our dataframe to remove any articles that had provided no content by searching for cells containing null values for the title, null values for the body of the HTML text, and any cells considered an empty string. Second, we removed any rows representing articles that were not successfully downloaded due to copyright/domain regulations. These rows were usually identified by a "403" code or a 'Forbidden' keyword and the lack of an URL within the dataframe's URL column. Next, we re-formatted all the texts contained within the '*body_text*' column of our dataframe by removing unnecessary new lines, extra spaces, emojis, and sentences containing less than 5 words.

2.1.3 Tokenization and Part of Speech

Tokenization and Part of Speech tagging were performed simultaneously after cleaning each article's text within the Dataframe. An iteration through each of the rows in our dataframe was established to perform the tokenization and tagging of text that matched specific criteria. Language detection was performed through the use of Python's Fasttext library. The language model "$lid.176.ftz$" was downloaded to predict a specific number of languages used in a given text. The Fasttext model was selected due to the fact that it can recognize 176 languages and provides a more generalized insight of the English language for having been trained on data from Wikipedia, Tatoeba, and SETimes. Fasttext language predictor based on the selected model was applied to each text. The output consisted of two possible languages and their respective scores. If the highest score was two times greater than the second language score, and its label corresponded to the English language, tokenization and tagging proceeded. Next, Spacy's Python library was utilized with the assistance of the large English language model "*en_core_web_lg*." The model was loaded into our Python Jupyter notebook in order to tokenize the text data. Lists containing the lemmatized nouns, adjectives, verbs, lemmas, and combinations as '*nav*' were produced by sorting the tokens per article into each list based on their Part of Speech tag assigned by the Spacy model. Lemmatization and tagging were only performed on articles that totalled or exceeded 30 tokens.

2.1.4 Word and Document Embedding

Once Part of Speech tagging was performed and its outputs stored in our dataframe, we removed articles that did not pass our language and tagging criteria. These were identified by finding elements in our dataframe which had null or empty values in their 'nouns' cell. Word vectorization via the statistical method of Term Frequency - Inverse Document Frequency (TF-IDF) was executed by creating a TfidfVectorizer from the Python library Sk-learn and applying it to the produced nouns of each article. Prior to performing the vectorization, a body of stop words was selected for our TfidfVectorizer function to be capable of filtering particular tokens which do not resemble significance in our clusters and our topic within the query. The stop words selected were retrieved from Spacy's language sub-library and applied to our TfidfVectorizer model. The TfidfVectorizer ensured all nouns from each article were transformed into lowercase tokens, which were then filtered out based on given stop words. The remaining nouns were given each a TF-IDF score representing their level of significance in relation to the content of the given text. The tfidf vectors correspond to the scores for all tokens per text stored in an instance variable for further use.

Gensim's library Word2Vec model was also applied to the nouns for each text to capture conceptual similarities and store them as word2vec vectors. Nouns were tokenized using Python regex and applying string lower-casing and filtering with the previously extracted Spacy library's stop words. The Word2Vec model was trained and its resulting word vectors were comprised of tokens that appear at least 5 times. Each vector contains 100 dimensions as per the model's preset number. Document embeddings were created by combining the tfidf vectors and the word2vec vectors previously created. If a document feature within the tfidf vectors was also found in the word2vec vectors, then its TF-IDF value was multiplied by each of its 100 dimension values on the word2vec vector. The newly produced dimension values were then normalized and stored as a document vector.

2.1.5 Document Clustering with K-Means algorithm

Sk-learn KMeans function was used to create a total of 25 clusters by using the created document vectors in the previous step. Afterwards, each cluster id together with their total number of contained documents and their titles were displayed within the computer console. The clusters were stored in a dictionary variable and a temporary duplicate of our dataframe was edited to store the cluster number assigned within a column '*cluster_num*.'

After performing the clustering algorithm, the mean similarity for each cluster was calculated. All cosine similarity scores for each pair of documents within a given cluster were added and its total sum was divided by the number of total pairs for that cluster. The mean similarity score of a cluster represents how similar the documents inside a specific cluster are to each other and give us insights into the efficacy of our web crawler application. The scores were displayed in the computer console together with their respective total documents within each cluster. Further, Matplotlib and WordCloud libraries were used for easy visualization of the clusters produced by our algorithm. Specific word cloud images can be seen under the results in section 3.

Last, the text within the dataframe was stored in two ways. Unique pickle files were created after scraping, cleaning, and part of speech tagging to ensure ease of use of data and fast access for purposes of clustering testing.

3 Results

Each time the crawler program was performed, documents were clustered to different cluster ids but maintained topic similarities. Therefore, the results to be discussed were obtained during one of the particular runs of our program. After creating the clusters and generating word clouds of the nouns in each cluster, we were able to easily identify their most significant and common words. Clusters 0 [fig. 1], 3 [fig. 2] and 20 [fig. 3] are good examples of the documents found to have identifiable storm information because they all have distinct types of hazards included in them. Cluster 0 is centered around documents containing information about winter storms, the national response, and losing power. In cluster 3, the articles contained are centered around context related to tornado hazards. Cluster 20 is similar to cluster 0 in being about winter storms, but it looks to be centered around North and South Dakota specifically.

On the contrary, Clusters 2 [fig. 4], 22 [fig. 6] and 24 [fig. 5] show that some of the documents had little to no storm/hazard information which shows the clustering algorithm was successful in properly clustering the documents. Cluster 2 looks to be about Wall Street and stock information, Cluster 24 only has date information and some mentions of 'storm'. Lastly, Cluster 22 looks to be about credit cards and personal loans. These results are great because the irrelevant information is clustered together with other irrelevant documents, making it easier to sort the information later.

4 Difficulties

We had a few problems surface while trying to do this project. First was trying to find an API that would get us the results we wanted. The official Google API and the Bing API both were hard to use and would only give us one search result at a time, so we would have gone through all our requests really fast if we continued to use them. Thankfully there was a third-party option available that provided more data and had a higher data cap. Secondly, downloading all the web pages took a long time because some of the websites were unreachable but the crawler would not timeout. To fix this we added a timeout cap of five seconds and that allowed everything to be downloaded in about 10 minutes. Additionally, during pre-processing of the body text, it was difficult to get rid of menu options and other useless text that was in the body tag. Additionally showing how successful the clustering algorithm did, was a difficult task because we had no way to test the model since the dataset was not labeled. So viewing the results through word clouds was the best option since we could still see if there were reoccurring words among the documents in the clusters.

5 Appendix

See next page.

Figure 1: This is the Word Cloud for the nouns in cluster 0.

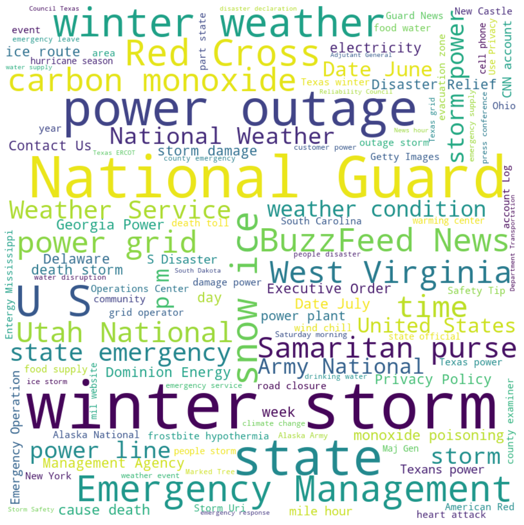


Figure 2: This is the Word Cloud for the nouns in cluster 3.

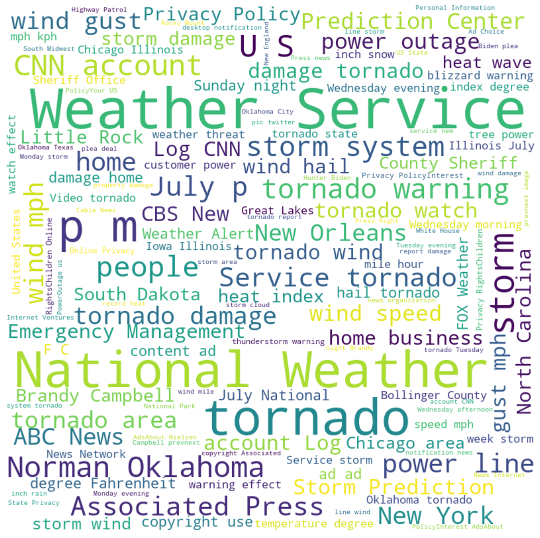


Figure 3: This is the Word Cloud for the nouns in cluster 20.

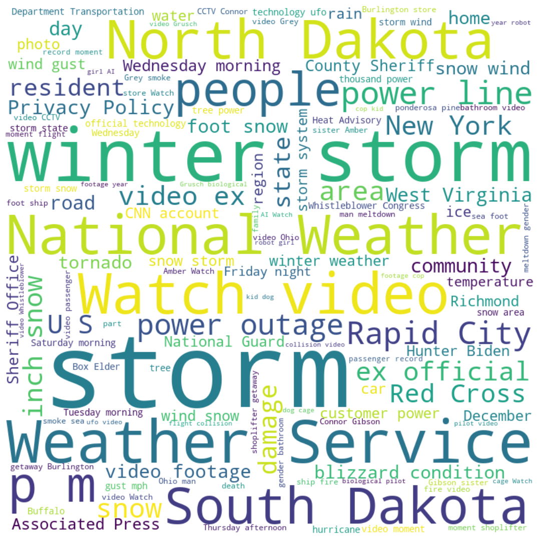


Figure 4: This is the Word Cloud for the nouns in cluster 2.

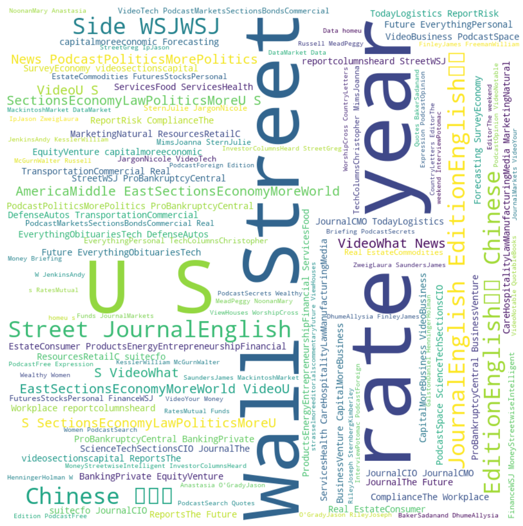


Figure 5: This is the Word Cloud for the nouns in cluster 24.

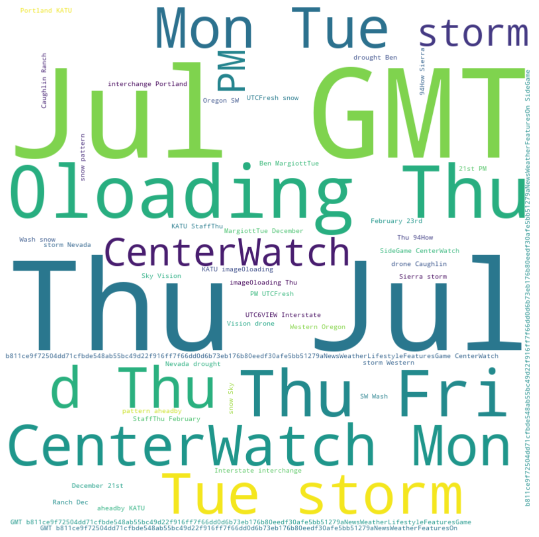
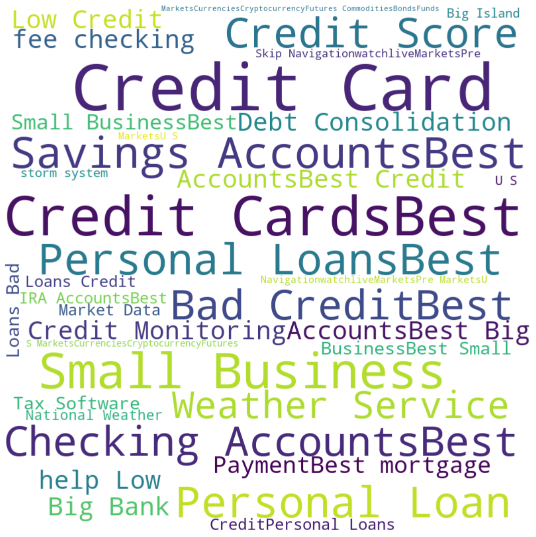


Figure 6: This is the Word Cloud for the nouns in cluster 22.


MarketsCurrenciesCryptocurrencyFutures CommoditiesBondsFunds
Big Island
Low Credit
fee checking
Credit Score
Skip NavigationwatchliveMarketsPre
Credit Card
Small BusinessBest
Debt Consolidation
MarketsU S
Savings AccountsBest
storm system
AccountsBest Credit
U S
Credit CardsBest
Personal LoansBest
Loans Bad
Loans Credit
NavigationwatchliveMarketsPre MarketsU
IRA AccountsBest
Market Data
Bad CreditBest
Credit MonitoringAccountsBest Big
S MarketsCurrenciesCryptocurrencyFutures
BusinessBest Small
Small Business
Tax Software
National Weather
Weather Service
Checking AccountsBest
PaymentBest mortgage
help Low
Big Bank
Personal Loan
CreditPersonal Loans