Twitter Sentiment Analysis on 2024 US Presidential Election Candidates

Yingqi Ma ym2926, Zhan Shu zs2584, Yuan Dou yd2676
Columbia University

Abstract

In this report, we capture the public sentiment towards 6 target candidates in the 2024 US Presidential Elections, by fetching and analyzing streaming tweets sent out in one hour. We apply NLP sentiment analysis to calculate the average sentiment score, and count the word which appears most frequently for each candidate over every 6-minute tumbling window. We compare the sentiment details between all candidates in one tumbling window, as well as the change of sentiment scores as the change of tumbling windows for every candidate, and provide insights into the differences.

1. Introduction

Twitter is undoubtedly one of the most popular social media platforms for users to share quick opinions or ideas through short text posts, with 436 million monthly active users and 500 million tweets sent daily [1]. A recent survey by the Pew Research Center shows that around 60% of U.S. social media users discuss political issues on sites like Twitter, making it a popular platform people rely on for communication and discussion surrounding political issues [2].

Sentiment Analysis is an area of natural language processing (NLP) measuring a sentiment (positive, negative, or neutral) of text by analyzing the words used in it [3]. In our project, we apply sentiment analysis to analyze the favorability of every presidential candidate.

Current forums are not precise on comparing candidates real-time rating. We use Twitter API to fetch the streaming data and filter out tweets relevant to the target US presidential candidates. Our analysis spans one hour, starting from 02:57:30 until 03:57:30 on May 1st, 2023 UTC, over which we got 37 thousand tweets that mentioned the top 6 popular candidates. After applying sentiment analysis on each tweet, we can then ascertain the average sentiment for every candidate during each 6-minute tumbling window and further compare the change of the sentiment over time. We also count the word which appears most frequently for each candidate over each 6-minute tumbling window. Such kind of key words help understand the sentiment of the candidates in that tumbling window. We finally implemented a web application to visualize the sentiment results.

In the following sections, we will first provide a brief overview of challenges faced currently. Next, we will introduce the technical details used for the project, followed by a detailed explanation of the results. Lastly, we will discuss the implications of the results and will state our conclusion.

2. Challenges Faced

Our main goal is to build a streaming system and web application that is able to reflect people's attitude on the 6 2024 US presidential election candidates using streaming data from Twitter. To achieve this goal, there are several technical difficulties we need to overcome:

1. Figure out how to get streaming data from Twitter and how to ensure that only tweets that are relevant to the 6 candidates are kept.
2. Figure out how to deal with latency when pulling data from Twitter. Since there are a large number of new tweet messages coming in every second, without some kind of optimization or strategies to resolve latency, processing these data can be slow, and we will not be able to update the result in time.
3. Develop a streaming algorithm that is able to calculate the average sentiment score, as well as the most mentioned word by the end of each window.

3. Technical Details

3.1 System Overview

The system of our streaming application is implemented on Google Cloud Platform with the usage of the Tweepy library and sockets to stream live tweets from Twitter based on a set of predefined keywords and send them to a client for further processing. Initially, the connector establishes a connection to fetch data from Twitter API, create a socket for the data and then wait for TCP connection. The TCP client connection is achieved with pyspark streaming and retrieves the data from the socket in DStream type.

Once we get the Dstream with twitter data, all the RDDs will first go through a filter function which only keeps those tweets that are relevant to the 6 candidates. Then the Dstream is split into sentiment analysis task and word count task, which these two tasks will be done in parallel.

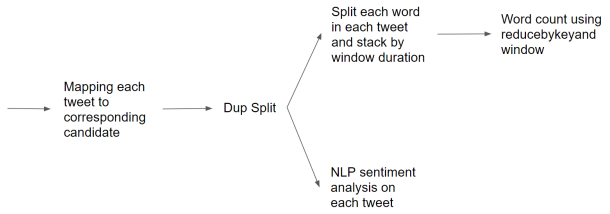

Figure 1: System Overview

Since both of the tasks need to first map each tweet message to its corresponding candidate, we decide to perform this transformation before the stream is split to the two tasks as shown in the above figure. Since this mapping operation needs to process every single tuple and output all of the tuples, the cost of this operation is expensive. Thus, redundancy elimination is profitable in this case, which greatly increases the throughput of the system. Theoretically, since the fraction of cost of this operation in the whole process is roughly one third, the overall throughput will increase by approximately 25% compared to that without redundancy elimination.

In word count task, each word in the tweet message is splitted and stacked by window duration, and then aggregated using reducebykeyandwindow. In the sentiment analysis task, each tweet will be transformed into a precise number, which reflects the sentiment score of that message. The details of these two tasks will be explained in the following sections.

3.2 Spark Streaming

For the streaming process, we decided to use Tweepy library to stream live tweets from Twitter API Platform based on a set of predefined keywords. Tweepy is a popular Python library for interacting with the Twitter API. It provides a convenient way to authenticate and make requests to the API, as well as to handle streaming data in real-time. With Tweepy, developers can build Twitter applications that access and analyze Twitter data. The library provides a simple way to authenticate to the Twitter API by setting up an OAuthHandler. We created a new application on the Twitter Developer platform, generating access tokens and consumer keys that can be used to authenticate requests. In this application, we only use a BEARER_TOKEN for authentication and it saves a lot of time configuring the authentication process. We also used sockets to send the tweets to a client for further processing. A socket is a networking component that allows two different processes on two different machines to communicate with each other. In our project, the socket is used to receive streaming data from the Twitter API and send it to the client application.

To utilize the tweepy package, we first define a class called MyStream, which extends the tweepy.StreamingClient class and works as a subclass. This class overrides the on_tweet() function, which is called whenever a new tweet is received. The function retrieves the text of the tweet and sends it to the client using the client_socket.send method in tweepy class. The method returns True to indicate that the tweet was successfully processed. If an error occurs while processing the tweet, the on_error() method is called, which prints the error status message and returns False to indicate that the tweet was not successfully processed.

Then we implement a function sendData which is responsible for creating a new instance of the MyStream class, adding the predefined tags to the stream, and filtering the stream based on those tags. This is basically a function that takes in sockets and tags and sends data to the socket by calling MyStream class and authenticates with the BEARER_TOKEN of Twitter API.

Another class twitter_client is built to create a new TCP socket and binds it to a specific host and port. It then listens for incoming connections, accepts the connection, and passes the connection and the tags list to the sendData method. The class is first initialized with a socket object created using socket.socket(), which creates a new socket with the given address family (socket.AF_INET for Internet Protocol) and socket type (socket.SOCK_STREAM for TCP). Then the .bind() function is used to bind the socket to a specific host (TCP_IP) and port number (TCP_PORT). This means that any incoming data sent to the specified host and port will be received by this socket object. Then the run_client() function waits for TCP connection and calls the sendData() function once the connection is established.

Now we finished our Twitter API connector, and we want to establish a connection from another notebook or machine with the IP and Port. We use pyspark streaming to establish the connection and get data from the connector server.

First we configure the SparkConf() object and set it up to run locally with 2 cores by using setMaster() function in spark streaming. Then we create a SparkContext() object with the above configuration. The streaming process is established by creating a StreamingContext() object that takes in the sparkcontext and setting the batch interval of 5 seconds. Therefore, the TCP connection will receive the data from the socket and stream the data in real-time in the spark streaming application. The DStream data is fetched from socket by using socketTextStream() and specifying the IP and Port. socketTextStream is a method provided by the Spark StreamingContext API that allows us to create an input stream that reads data from a TCP socket. This creates a DStream (discretized stream) of data, which represents a continuous stream of data that is divided into 5 second batches as stated in SparkContext().

To avoid data latency in a streaming application, we tried several approaches to minimize the time of latency. First approach is reducing the batch interval of

the streamContect(). Reducing the batch interval can help reduce data latency in streaming applications. This means that the application processes data more frequently, reducing the delay between the time data is received and the time it is processed. However, there are also some problems if the batch interval is set at a too small value. For example, the data may be incomplete, and the sentiment analysis may be affected if there is too little data. Eventually we select the batch interval of 5 seconds to run the data. Another approach we tried to reduce latency is increasing the cluster node. Increasing the cluster nodes allows the application to process data faster. This is an effective way, but the disadvantage is also apparent: the cost of the cluster on Google Cloud is increasing. Therefore, we are consistent with our 2-node cluster. Some potential solutions about data latency are discussed in Section 6 Future Works.

3.3 Word Count

The topWordCount function we defined is a transformation function that processes a DStream of tweets correlated to a candidate, and it returns a DStream of the top word count data for each candidate in a sliding window. The input DStream is in the format (candidate, related_tweet).

The first step of the function is to group all the tweets related to a candidate together in a single line string, by reducing the DStream by the key candidate. The resulting grouped_tweets DStream will have tuples in the format (candidate, tweets_string).

The next step is to tokenize the tweets string by splitting on whitespace and filtering out NLTK library stop words and common words. This is done by mapping the grouped_tweets DStream to a new DStream candi_word, where each tuple is in the format (candidate, [list of filtered words]).

The candi_word DStream is then flatmapped to a new DStream candi_allwords_flat with tuples in the format ((candidate, word), 1). By combining the candidate-word pair, we can keep track of the count of each word for each candidate.

A reduceByKeyAndWindow operation is then performed on candi_allwords_flat with a window length of 10 seconds and a sliding interval of 10 seconds, to get the total count of each word for each candidate in the window. The time windows here could be adjusted to desired and suitable time depending on the total stream time.

The resulting candi_allwords_flat DStream will have tuples in the format ((candidate, word), count). The DStream is then transformed into a new DStream top_words, which groups the data by candidate and keeps only the word with the highest count for each candidate.

The top_words DStream will have tuples in the format (candidate, (word, count)). The DStream is then mapped to a new DStream most_count_words, which extracts the top word and its count for each candidate, and returns tuples in the format (candidate, word, count).

The last step of the function is to map the most_count_words DStream with the current time, by using the transform function. The transform function applies a function to each RDD in the DStream, which in this case maps each tuple to include the time at which the top word count was recorded. The resulting DStream is then returned. After that the DStream data was saved to google cloud storage and saved into Big Query Schema.

Overall, the topWordCount function is a crucial component of our streaming application, as it provides insights into the most frequent words related to each candidate in real-time. This information can be used to understand the public sentiment and opinions about the candidates and can help in making data-driven decisions.

3.4 Sentiment Analysis

Valence Aware Dictionary for Sentiment Reasoning (VADER) is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. VADER sentiment analysis relies on a dictionary that maps lexical features to emotion intensities known as sentiment scores. The sentiment score of a text can be obtained by summing up the intensity of each word in the text. VADER's SentimentIntensityAnalyzer() function takes in a string and returns a dictionary of scores in each of four categories: negative, neutral, positive, and compound. [4] The negative, neutral, and positive scores represent the proportion of the text that falls in these categories, and the compound score is calculated by normalizing the three scores. If the compound score is less or equal to -0.05, then the text is considered negative; if the compound score is greater or equal to 0.05, then the text is considered positive; otherwise, the text is considered to be neutral.

Since the texts we are analyzing are tweet messages, where lots of emojis and slangs are involved, and these elements are crucial to determine the emotions in each tweet message. While many other NLP sentiment analysis models simply ignore these elements they cannot identify, VADER performs very well with emojis, slangs, and acronyms in sentences. [5] What's more, VADER identifies usernames and links as neutral, so we don't have to do any complicated preprocessing to the tweet message. VADER also processes text fast enough to be used online with streaming data.

As mentioned in section 3.1, new tweets will flow through the system in the format of Dstreams. For each tweet in the Dstream, it will be filtered so that only all tweets that are irrelevant to the 6 candidates will be ignored. Then all the tweets will be mapped to their

corresponding candidates following the condition that if their names are mentioned in the tweet. Then each tweet message is treated as a string and analyzed using VADER's SentimentIntensityAnalyzer(). Since we are looking for accurate and simple standard that could evaluate all the tweets and comments, we used the compound category only, which best represents the attitude of each tweet in a precise number. Finally, each tweet, along with the corresponding sentiment, will be assigned with a timestamp, which represents the exact time the tweet message is collected.

3.5 Streaming Result Processing

Table 1: Sentiment Analysis Result

time	tweet	candidate
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	-0.7845	biden
2023-05-01 03:24:20.000000 UTC	0.8221	biden
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	0.2263	biden
2023-05-01 03:24:20.000000 UTC	-0.0772	biden
2023-05-01 03:24:20.000000 UTC	-0.5574	biden
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	-0.5574	biden
2023-05-01 03:24:20.000000 UTC	-0.3818	biden
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	-0.5423	biden
2023-05-01 03:24:20.000000 UTC	-0.3818	biden
2023-05-01 03:24:20.000000 UTC	0	biden
2023-05-01 03:24:20.000000 UTC	-0.6705	biden
2023-05-01 03:24:20.000000 UTC	0.3818	biden
2023-05-01 03:24:20.000000 UTC	0	biden

Table 2: Word Count result

time	count	word	candidate
2023-05-01 02:57:30.000000 UTC	174	hunter	biden
2023-05-01 03:03:30.000000 UTC	181	hunter	biden
2023-05-01 03:15:30.000000 UTC	194	hunter	biden
2023-05-01 03:27:30.000000 UTC	199	hunter	biden
2023-05-01 03:21:30.000000 UTC	201	hunter	biden
2023-05-01 03:39:30.000000 UTC	207	hunter	biden
2023-05-01 03:45:30.000000 UTC	209	hunter	biden
2023-05-01 03:09:30.000000 UTC	219	hunter	biden
2023-05-01 03:51:30.000000 UTC	203	hunter	biden
2023-05-01 03:33:30.000000 UTC	243	hunter	biden
2023-05-01 02:57:30.000000 UTC	92	&	trump
2023-05-01 03:39:30.000000 UTC	97	&	trump
2023-05-01 03:27:30.000000 UTC	104	the_trump_train	trump
2023-05-01 03:45:30.000000 UTC	118	the_trump_train	trump
2023-05-01 03:21:30.000000 UTC	119	the_trump_train	trump
2023-05-01 03:33:30.000000 UTC	120	the_trump_train	trump
2023-05-01 03:09:30.000000 UTC	120	the_trump_train	trump
2023-05-01 03:03:30.000000 UTC	120	&	trump

The above two tables show the sentiment analysis result and word count result after running the stream for an hour. The sentiment analysis result consists of three columns: timestamp, which is the exact time each tweet message is collected; sentiment score, which is the normalized sentiment score for each tweet message as described in the previous section; candidate, which is the name of the candidate each tweet is corresponding to.

The word count result is calculated at the end of every tumbling window, which in our case, is calculated every 6 minutes. The word count result consists of 4 columns: timestamp, which is the exact time of the end of each tumbling window; word, which is the word that appears most frequently in each window for each candidate; count, which represents how many times the corresponding word appears in that window; candidate, which is the name of the candidate.

Once we get these results, we use spark sql window function to create 6 minute windows using the time column, which results in 10 distinct windows. For each candidate in each time window, the sentiment scores are aggregated to calculate the average sentiment score by grouping candidate and window. The word count result is then added to the spark dataframe if for each row in the word count, the timestamp is within the range of any of the 10 windows and the candidate matches. Finally, the results are broken into smaller dataframes, with each data frame representing the streaming results in a specific 6 minute window.

4. Result

Our analysis spans one hour (6 minutes per time window), starting from 20:08 until 21:08 on April 24th, 2023, over which we got 37 thousand tweets that mentioned the top 6 popular candidates. And we get 10 csv files of sentiment details for 10 time windows.

Table 3: Sentiment details of latest time window

	A	B	C	D	E
1	candidate	sentiment	tweet_count	word	word_count
2	asa hutchinson	-0.37293	4	republican	2
3	biden	-0.05168	1460	hunter	209
4	marianne williamson	0.4404	9	trudygonzales:	12
5	nikki haley	0.2294	2	disney;	1
6	ramaswamy	0.103707	61	vivekgramaswamy	23
7	trump	-0.01721	1836	the_trump_train	118

Each csv file includes the average sentiment score, total number of tweets, most mentioned word, and number of most mentioned word for every candidate. With these csv files, we implement a web application to visualize the results.

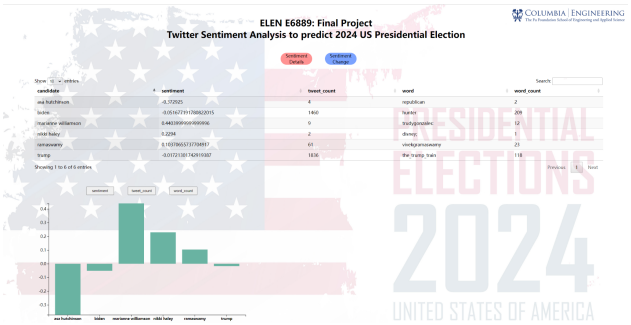

Figure 2: Overview of Web

It has two sections, the first section shows the table of sentiment details of the latest time window, as well as the bar chart of average sentiment scores.

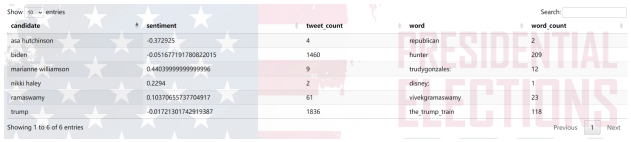

Figure 3: Table of Latest Sentiment Details

The table helps users to get the latest information of candidates' sentiment. Users may be curious about the sentiment score, like why Trump has a neutral sentiment while Biden has a negative one. The most mentioned word could help them to know the keyword that other twitter users are focusing on. They could easily search the candidate's name together with the keyword to get more political updates.

There's a sort button next to each column name. This will help users to sort the table according to statistics they are most interested in. For example, if a user wants to check who has the highest sentiment score, just click on the column of "sentiment". And the table will show the rank of sentiment. In the figure below, we can see that Marianne Williamson has the highest sentiment, while Asa Hutchinson has the lowest.

candidate	sentiment
marianne williamson	0.44039999999999996
nikki haley	0.2294
ramaswamy	0.10370655737704917
trump	-0.01721301742919387
biden	-0.051677191780822015
asa hutchinson	-0.372925

Figure 4: Sorting of Average Sentiment Score

The Bar Chart intuitively shows the comparison of average sentiment score, total number of tweets, and number of most mentioned word between the candidates.

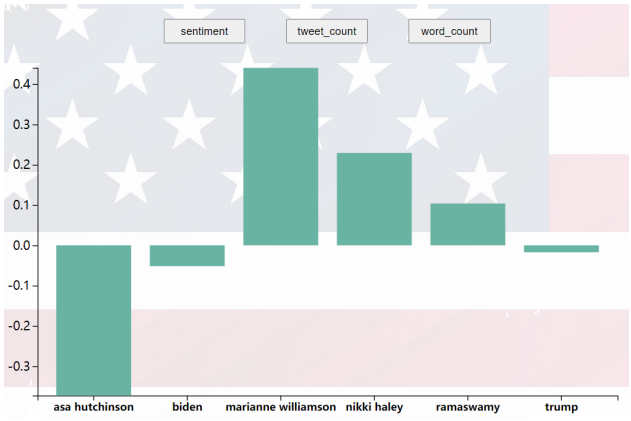

Figure 5: Bar Chart of Average Sentiment Score

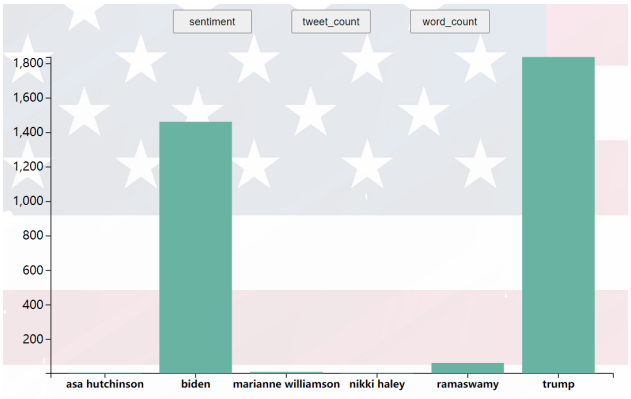

Figure 6: Bar Chart of Count of Relevant Tweets

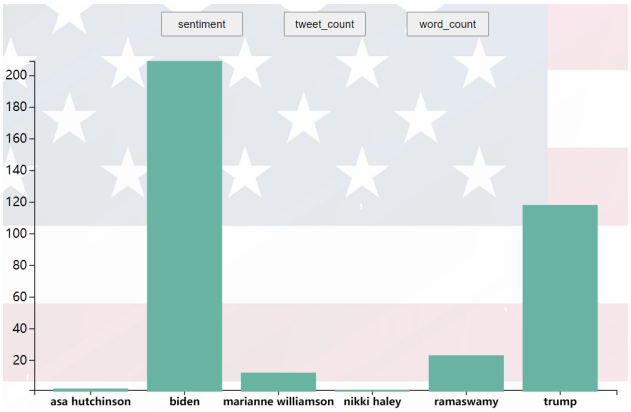

Figure 7: Bar Chart of Count of Most Mentioned Word

From these Bar charts, we can see that Trump and Biden have relatively more focus than other candidates. And the more tweets about one candidate, the less absolute value this candidate's average sentiment score is, and our analysis of their sentiment is more precise. For the candidates with less than 50 tweets, our analysis of their sentiment is not that precise, because without enough tweets, the sentiment results tend to be one-sided.

In the second section, a line chart is used to show the change of sentiment score over time for a particular candidate. Users can choose the candidate using the select button on the top left.

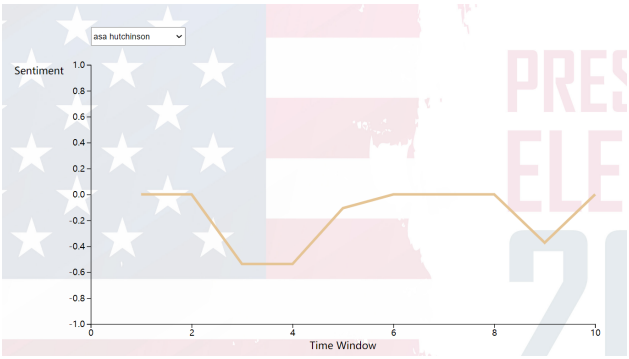

Figure 8: Line chart of Sentiment Change Over Time (Hutchinson)

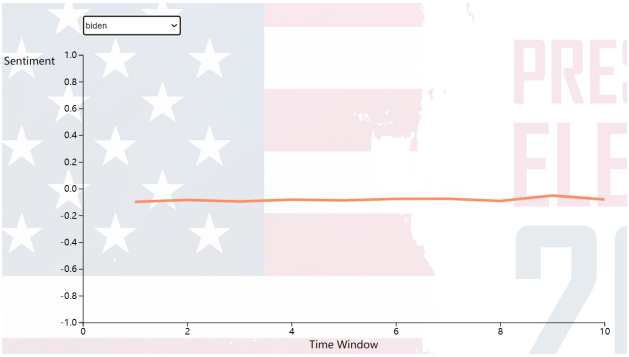

Figure 9: Line chart of Sentiment Change Over Time (Biden)

From the comparison of Hutchinson and Biden's line chart, we find Biden has a more smooth line chart. That's because Biden has more tweets about him, so the average sentiment of him will be more stable.

5. Conclusion

Through the use of sentiment analysis and word count, we provide a numerical and graphical result of sentiment details of these 6 candidates. Also we visualize the comparison of sentiment and popularity between them, as well as how political sentiments, during a time period, are changed on a social platform like Twitter.

Comparing the sentiment analysis results and the word count, we build a connection between the simple numbers and the keyword of the candidates that people are mostly concerned about. Users can better understand the candidates' sentiment scores by referring to the keyword and know what happened to the candidates.

6. Future Work

In the future, one possible improvement to the current implementation of Twitter streaming is to save the wordcount and tweet data directly to a cloud bucket instead of Big Query schema. This would allow for easier access and manipulation of the data by other functions and applications, such as the window aggregation function and our web application. With the data stored in the cloud, the window aggregation function could be scheduled to run periodically and pull the data from the cloud bucket for analysis. This would remove the need for the function to process the data in real-time, reducing the load on the system and potentially increasing the efficiency of the analysis.

Additionally, another potential improvement would be implementing the compression of data. Data compression can reduce the amount of data that needs to be transferred, reducing the time it takes to move data through the system. This method can further reduce the latency of the data therefore make the analysis more accurate.

7. Reference

[1] Salman Aslam, "Twitter by the Numbers: Stats, Demographics & Fun Facts, " https://www.omnicoreagency.com/twitter-statistics, 2023.
[2] M. Duggan and A. Smith, "The Political Environment on Social Media," 2016.
[3] Feldman R. Techniques and applications for sentiment analysis. Communications of ACM. 2013;56(4):82–9.
[4] Beri, A. (2020, May 27). Sentimental analysis using vader. Medium. Retrieved May 4, 2023, from https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664
[5] Pandey, P. (2019, November 8). Simplifying sentiment analysis using Vader in Python (on social media text). Medium. Retrieved May 4, 2023, from https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f