International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307 OPEN ACCESS

Available Online at : www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT2390646

Real-time Sentiment Analysis from Data Streaming

Samit Shivadekar[1], Ketan Shahapure[2], Shivam Vibhute[3], Milton Halem[4]

[12]Department of CSEE, University of Maryland Baltimore County, USA

[3]San Jose State University, San Jose, California, United States

[4]University of Maryland, Baltimore County

ARTICLEINFO

Article History:

Accepted: 02 Jan 2024

Published: 10 Jan 2024

Publication Issue

Volume 10, Issue 1

January-February-2024

Page Number

60-70

ABSTRACT

Public sentiment is a potent indicator of how people perceive and receive a topic. It has the power to make or break companies and people. Twitter is one of the best platforms in today's generation to gauge public sentiment. [10] Utilizing the power and influence Twitter has we decided to create a service that would enable us to know how a trending topic is being viewed by the masses in real-time. The user gives the topic as input to the front-end graphical user interface that topic is then taken and fed to the Twitter streaming API. Tweets containing the hashtag of the topic mentioned by the user are returned and the sentiment of those tweets is predicted and sent to the front end where analysis prediction of the sentiment of the tweets is done dynamically as the tweets come in. By using our service for a few minutes the user will get to know what the overall outlook of a topic is and use that information as a guiding beacon for any future decisions regarding that topic.

Index Terms : Twitter, Sentiment Analysis, Real-Time, Stream-ing Data, Topic Analysis, Model training, Prediction

I. INTRODUCTION

The Goal we set out to achieve was to create a service that would enable the user to get a real-time sentiment analysis of the tweets that were coming from the Twitter streaming API based on the topic the user gave as input. Using the Service-oriented computing principles we came up with a service that provided a simple user interface that took input from the user as well as also displayed the tweets and charts that came from the back end. We made use of a large data set to train our model which was later employed to predict tweets as they came in real-time. The working prototype we were able to produce was quick and accurate in its predictions. We feel that this service would come in handy for anyone who is curious to know how a trending topic is performing amongst the masses. Emotions and feelings are two very strong influencing powers that dictate how people behave and they also have a huge impact on actions people take in the real world. We strongly believe humans learn from their mistakes. Our service helps the user understand how some topics are perceived and based on that information the user can make changes so that the next time around the topic has an overwhelmingly positive sentiment around it.

II. SERVICE OBJECTIVES

The Objectives of our Service are as follows:

- Obtain Sentiment of tweets produced by Twitter users on a specified topic in real-time.
- Predict whether the tweet is positive negative or neutral in nature.
- Perform analytics on the tweets that have been obtained and show those results in the form of pictorial charts to the user.

III. IMPLEMENTATION PLAN

The following steps are what we followed to implement the service:

- Obtain topic input from the user through the front-end user input page.
- Obtaining real-time tweets from Twitter using Twitter streaming API for the topic given by the user.
- Stream the topic using Apache Kafka.
- Train a sentiment analysis model on static data.
- Using the trained model, we predict the sentiments of the tweets that come in real-time.
- Take predictions from the model to create pictorial charts that aid in the overall analysis of the topic.

USERS AND CLIENTS

Marketing and Executive teams of Companies that release products. Marketing teams in a company can use our service to see how their marketing strategies for products are playing out if they see that the product is being viewed negatively they can take necessary marketing steps and make necessary recommendations for the product teams for changes. Executive teams can see how their product performs when they release it and based on that they can make decisions on whether to fund or scrape the product because if a product brings the company's reputation down there is no point in putting it out to the market.

Stock, Crypto and NFT investors. Twitter is one first places that gets any news on Cryptos and NFTs and based on that its users react. If the sentiment about a stock is negative it's better to sell it as that is an indication that the stock or Crypto or NFT is going to lose its value and market share. On the other hand, if the sentiment on a stock or Crypto or NFT is positive then it would be a good idea to invest in those things as people are more likely going to buy it more thus increasing its value.

Famous People. When famous people tweet about things. They can use our service to see how what they said is being viewed by Twitters users. Based on that they can decide to continue on a said topic or avoid it. There are many instances where famous people have either gained or lost fans because of what they have tweeted.

IV. SERVICE ARCHITECTURE

The service architecture diagram of our service is shown in Figure 1. which shows all the components involved in the system.

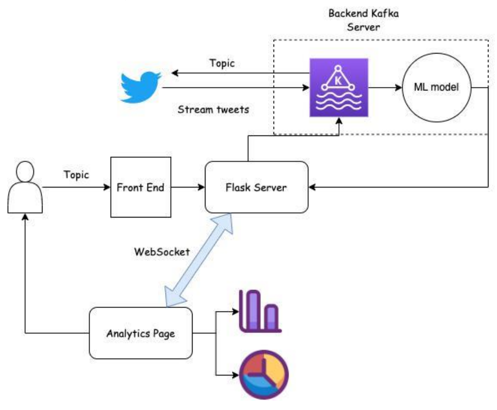


Fig. 1. Service architecture of Real-time Sentiment analysis

As shown in Figure 1, there are mainly 3 components in the system. The front end, Machine Learning model and the backend. Each of these components and their working is explained in much detail in the coming sections. All the components are connected to each other and all of them will be communicating with the help of REST APIs, sockets and Web Sockets.

It is a kind of Microservice architecture [9] where all the components work independently of each other. In general monolithic architectures, all the components of software lie in the same machine and if one of the services breaks, it results in breaking the entire codebase and none of the components will be usable. This similar strategy is extended by a microservices architecture to loosely connected services that may be created, deployed, and maintained separately. Each of these services is in charge of a certain task and can interact with other services via straightforward APIs to resolve more complicated business problems. Instead of being forced to adopt a more uniform, one-size-fits-all approach, the microservices architecture makes it simpler to select the technological stack (programming languages, communication protocols, etc.) that is most suited for the desired functionality (service).

Our system is also an application of micro-service architec-ture. Because we have separated the Machine Learning com-ponent and streaming component from the front-end serving component. The streaming and model prediction service works independently and even if something in the front end fails, we still can get a stream of messages with the prediction for a specific topic if the prediction or the streaming service fails, the front end service which is powered by Flask will still be working although there will be no use of it since we need to get streams of text from the backend for real-time updates. The main work and flow of the project are explained in this section as follows.

When the user enters our website, he is prompted with a Front-end UI which contains a text box where he can enter the topic. The topic can be anything for which the user needs to get real-time tweets and get sentiment analysis from it. Once the topic is entered, the topic is sent to the Flask backend. Flask receives this request and it is sent to the backend web server which hosts Kafka server and the Machine Learning model with the help of sockets. Once the backend server receives the topic, the Twitter streaming API is called and we start getting tweets from the Twitter server. The tweets are obtained from the Twitter server and are fed to the Deep Learning model which is residing in the backend and the sentiment predictions are obtained. This Deep learning model is a transformer model which is already trained on a static twitter dataset where the trained model is saved and is used for prediction. Since we are using a separate server for the Kafka and Machine Learning model, we can customize this server by adding GPU capability or scaling it up. This further proves that Machine Learning and Kafka components are isolated from the Flask which serves the front end and hence Microservice architecture is achieved.

The prediction and the tweets are sent to the Flask which acts as a consumer of Kafka. With the help of WebSockets, a bi-direction connection is made between the front end and the back end and the tweets are sent to the front end in real time.

These tweets are organized using front-end frameworks and are displayed. The number and percentage of positive and negative sentiments are displayed using a Bar graph and Pie chart respectively in the front end which updates dynamically.

V. DATASET OVERVIEW

In order to predict the sentiments of the tweets in real-time, we need a Machine Learning or Deep Learning model in predicting the sentiments. So, to get a sentiment analysis model, we need to train it before, save it, and use it later. We can also use models which are already pretrained and can be used as an API. We used a Twitter dataset which is available in Kaggle. The dataset contains 1.6M rows of data. The dataset consists of features such as, 'ids' where each entry in the dataset has a unique ID, 'date' which has the date when the tweet is posted, 'user' the user that tweeted, 'text' has the text of the tweet. Out of all these attributes in the dataset, our main focus is on using text field. This text field is processed and then used in the model for training. The sentiment of the text is the target variable. The dataset contains 3 sentiments which are Positive, Negative and Neutral sentences. Positive sentences are indicated by 0, 1 for Neutral sentences and 2 for Negative sentences.

VI. MACHINE LEARNING MODEL

Machine Learning and Deep Learning models are trained to predict the sentiments of the text in the dataset. Out of all these attributes in the dataset, our main focus is on using text fields. This text field is processed and then used in the model for training.

To train the sentiment analysis model, we created a baseline model and a final model for completing the model training cycle. We need a baseline Machine Learning model using which we can determine the minimum accuracy which can be obtained. This baseline model must be very simple without performing hyperparameter tunings and also basic default configurations. The baseline model which we decided to use is the Logistic Regression model. The model was trained on the text data in the dataset and the labels.

To train the model on the text data, the data must be processed and cleaned before feeding to the model. For data cleaning, we removed all the rows which do not have any text but have sentiments. The text data is cleaned by removing all the HTML tags, and punctuations and converting all the sentences to lowercase. Also, all the words are lemmatized so that all the differences between similar words are eliminated. Stemming is a method by which a portion of the word is simply sliced off at the end. While several algorithms are employed to determine how many letters must be removed, none of them genuinely understand the meaning of the word in the language to which it belongs. On the other hand, the algorithms in lemmatization are aware of this. In fact, you could argue that these algorithms use dictionaries to comprehend the meaning of the word before distilling it to its lemma, or fundamental term. Once we clean the text data, we have to process the data in order to convert the text into numerical format since the model cannot understand characters in the text. We used Tf-IDF vectorizer to convert the text data into its numerical format. TF-IDF can be broken down into term frequency and inverse document frequency. Term frequency is the frequency of a word with respect to a document. Inverse document helps in getting a number for how frequent the word is in the overall corpus. This helps in reducing the final weight on common words like articles, etc. The results of the logistic regression model are shown in the results section.

Since we have a big dataset having we can use Deep Learning models to train the sentiment analysis task. For the same reason, we have used BERT [5]model which is a transformer model. Transformer is an attention mechanism that learns the contextual relationships between words (or subwords) in a text and is used by BERT. Transformer's basic design consists of two independent mechanisms: an encoder that reads the text input and a decoder that generates a job prediction. Only the encoder mechanism is required because BERT's aim is to produce a language model. The architecture of BERT is shown in Figure 2. We have taken the architecture diagram from one of the websites which has been cited in the reference section. [5]

Fig. 2. BERT Architecture

In the BERT training phase, the model learns to predict whether the second sentence in a pair will come after another in the original document by receiving pairs of sentences as input. During training, 50% of the inputs are pairs in which the second sentence is the next one in the original text, and in the remaining 50%, the second sentence is a randomly selected sentence from the corpus. The underlying presumption is that the second phrase will not be connected to the first. This working is used by BERT to predict the next word in a sentence. It can be modified for a classification task as well. To train the BERT model, the sentences first have to be converted into a format that the model can understand. Here, we cannot use the TF-IDF vectorizer since the BERT model has its own way of encoding the texts. [5]BERT first uses some tokens to differentiate between the sentences. The first sentence has a [CLS] token at the start, and each subsequent sentence has a [SEP] token at the end. The unknown words are replaced by [UNK] tokens since they are not in the dictionary of the words. The rest of the words which are already known are replaced by a numeric number which is already defined by BERT where it has a dictionary of words which has a numeric number for most of the words.

The entire dataset is divided into training, testing and validation dataset where 70% of the data is used for training, 20% of data is used for validation and 10% of the data is used for testing. The training dataset is trained and model obtained is used as an API for predicting the sentiments of the tweets in real-time and prediction is sent to the front-end.

VII. RESULTS OF MODEL TRAINING

The dataset is trained on both the Baseline model and the BERT model and the accuracies are obtained. For baseline model training, the dataset is divided into 80% training and 20% testing. We calculated only the accuracy where we obtained an accuracy of 78.4% on training and 78.8% on the testing dataset. The logistic regression model was trained very fast because we did not do any hyperparameter tuning on that.

For training the BERT model, 70% of the dataset was used for training. The total size of the dataset was 1.6M tweets. Since the dataset was balanced, the accuracy, precision and recall obtained were almost the same. The accuracy on the test dataset which is around 10% of the entire dataset size obtained had an accuracy of 85.23%, precision was 85.86% and recall was 84.15%. The BERT model was trained for 4 epochs in Google Colab with a GPU instance. It took around 8hrs where each epoch took around 2hrs to train. This trained model was saved and is used in the prediction.

VIII. FRONT END

For the front end we kept it very simple we only made two pages. One page takes the user's input and the second page displays the tweets along with the prediction accompanied by the prediction analysis charts. To show the continuous stream of tweets we WebSockets and for the charts, we used the javascript library chart.js, this library helped us plot dynamic charts that kept changing as new tweets were sent to the front end.

IX. BACKEND FRAMEWORK COMPONENTS

The backend framework is built on python programming language. Flask, web framework, has been used as a backbone for the backend. we chose the flask because of its WSGI nature. [8]A WSGI is nothing but a server which implements the web server side of the python applications.

Importance of WSGI, the traditional web servers we have doesn't know how to run these python applications. So we are in need of a web server which handles this region and runs these arbitrary python codes. WSGI is the most acclaimed and recognized approach all over the community.

[7] Twitter developer account, this account plays a key role in our service. With the help of this account, we are able to use the Twitter APIs. So it allows you to manage and set up your applications and project. But the projects within this scope can be enabled to connect to Twitter and access its API. Here project in the sense which allows or gives you to set up your environment or your work based on your goals, and use cases, with the Twitter developer platform's APIs, which requires making asynchronous or synchronous requests to the Twitter developer APIs. But here the APIs calls are limited, so it basically restricts to certain calls of retrieving tweets per month from specific endpoints at the project level.

From the Twitter developer account, you get access to a lot of endpoints. But for the project goal, we are mostly dependent on streaming API, because of the scope to get real-time data, in our case series of tweets.

There are several advantages to using a REST streaming

API in real-time use cases [7]:

Low latency: REST streaming APIs allow for low latency communication between the client and the server, as data can be streamed continuously without the need for the client to send requests and wait for a response. This is important for applications that require real-time data transfer, such as online games and collaborative tools.

High throughput: REST streaming APIs can handle high volumes of data with minimal overhead, making them well-suited for applications that require high throughputs, such as financial trading systems and real-time analytics.

Efficient use of resources: REST streaming APIs can be more efficient than traditional REST APIs in terms of resource utilization, as they allow for continuous communication rather than the request-response cycle of traditional REST APIs. This can be important for applications that handle large amounts of data or have a high volume of requests.

Easy to implement: REST streaming APIs can be implemented using standard HTTP techniques, such as chunked transfer encoding, making them easy to integrate with existing systems and infrastructure.

Flexibility: REST streaming APIs can be used in a variety of real-time use cases, such as real-time data processing, event-driven architectures, and real-time messaging. They can be implemented using a variety of technologies, such as WebSockets, Server-Sent Events, and long polling.

[6] Kafka is a publish-subscribe messaging system that allows for the transfer of large volumes of data between differ-ent systems, enabling real-time data processing and analysis. Kafka is designed to be fault-tolerant, scalable, and fast, and it can handle high volumes of data with low latency. It is often used in conjunction with other technologies such as Apache Hadoop, Spark, and Flink for big data processing and analysis. Kafka is widely adopted in a variety of industries, including finance, healthcare, and e-commerce, for a variety of purposes such as real-time analytics, event sourcing, and data integration.

There are several ways in which Kafka can be used with

Python [6]:

As a producer: You can use Kafka's Python client library to publish data to a Kafka cluster. This is useful for streaming data from a Python application to other systems or storing it in a distributed manner.

As a consumer: You can use Kafka’s Python client library to consume data from a Kafka cluster. This is useful for processing data in real-time as it is produced, or for creating a data pipeline between different systems.

With a Kafka Connector: You can use a Kafka Connector, such as the one provided by Confluent, to integrate Kafka with other systems that have a connector available. For example, you can use a connector to stream data from a database like MySQL or a file system like HDFS into Kafka, or stream data from Kafka into a database or file system.

With a stream processing library: You can use a stream processing library like Apache Flink or Apache Spark Stream-ing to process data from Kafka in real-time. These libraries provide higher-level abstractions for working with streaming data and allow you to perform transformations, aggregations, and other operations on the data.

With a messaging library: You can use a messaging library like PyKafka or confluent-kafka-python to interact with Kafka in a more low-level way. These libraries provide APIs for producing and consuming messages, as well as for managing topics and consumer groups.

Flask is a lightweight Python web framework that allows you to build web applications quickly. It is easy to set up and requires minimal boilerplate code. [6]Kafka is a publish-subscribe messaging system that allows for the transfer of large volumes of data between different systems. You can use Kafka with Flask by using a Kafka library or connector to publish data from your Flask application to a Kafka cluster or to consume data from a Kafka cluster and use it in your Flask application. This can be useful for creating real-time data pipelines, streaming data to and from web applications, and performing real-time processing and analysis of data.

Flask-Kafka is a Flask extension that provides easy inte-gration with Apache Kafka. It allows you to use Kafka in your Flask application by providing a simple Flask-Kafka client, as well as decorators for producing and consuming messages. Flask-Kafka is built on top of the Kafka Python client, and it provides a simple, high-level API for working with Kafka in a Flask application. With Flask-Kafka, you can publish messages to Kafka topics and consume messages from Kafka topics, as well as manage consumer groups and topics. Flask-Kafka is useful for building real-time data pipelines and streaming applications, and it can be used in conjunction with other technologies like Apache Flink and Apache Spark for big data processing and analysis. An event-driven approach to building REST APIs involves designing the API in such a way that it sends out notifications, or ”events,” when certain actions occur. For example, when a new user is created in the system, an event could be sent out to notify other parts of the system that a new user has been created. This allows for real-time, asynchronous communication between different parts of the system, rather than relying on polling to check for updates.

An event-driven approach can be implemented using web-hooks, which are HTTP callbacks that are triggered by an event. When an event occurs, the system sends an HTTP POST request to the URL specified in the webhook, along with a payload of data related to the event. The receiving system can then process the event and take any necessary actions.

Event-driven approaches are useful for building highly scalable and responsive systems, as they allow for decoupled, asynchronous communication between different parts of the system. They are particularly useful for building microservice architectures, where different services need to communicate with each other in real-time.

[4] In our use case we followed WebSockets. WebSockets is a protocol that allows for the creation of a full-duplex communication channel between a client and a server. It allows for real-time communication between the two parties, allowing for the exchange of data without the need for the client to con-tinuously poll the server for updates. This makes WebSockets well-suited for applications that require real-time data transfer, such as chat apps, online games, and collaborative tools.

[4] To establish a WebSocket connection, the client sends a request to the server with an ”Upgrade” header, asking the server to switch to the WebSocket protocol. If the server agrees, it sends back a response with a ”101 Switching Protocols” status code, and the two parties can then commu-nicate over the WebSocket connection. Once established, the WebSocket connection remains open until one of the parties closes it.

WebSockets are implemented using a combination of JavaScript and HTML on the client side, and a WebSocket server on the server side. The WebSocket protocol is supported by most modern web browsers, and there are libraries and frameworks available for building WebSocket-based applica-tions in a variety of programming languages.

But there are challenges in integrating these technologies for our cases and making use of good data flow design matters a lot. System integration refers to the process of combining multiple systems and technologies in order to achieve a desired goal. It involves the integration of hardware, software, and data from different systems, as well as the integration of business processes and organizational structures. System integration can be achieved through a variety of methods, such as using APIs, message brokers, and integration platforms. The goal of system integration is to enable the seamless exchange of data and functionality between different systems, leading to improved efficiency, productivity, and agility. System integration is often used in a variety of contexts, including IT infrastructure, business processes, and supply chain management.

But after integrating these various technologies according to our use case. we need to test it regressively. So below are the touchdown points to be noted.

Here is a more detailed test plan for a streaming API with Kafka and Python:

Set up the testing environment: Install and configure Kafka, Python, and any required libraries or frameworks. Set up a test Kafka cluster and create test topics. Set up a test database or other system for storing test data. Define the test cases: Test the ability to produce and consume messages to and from Kafka. Test error handling scenarios, such as when the Kafka cluster is unavailable or when there are issues with the test data. Test the ability to scale the system under load, such as by increasing the volume of messages being produced or consumed. Test the ability to process and transform data in real-time, such as by using a stream processing library like Apache Flink or Apache Spark Streaming. Prepare the test data: Set up the test data that will be used in the test cases. This may involve creating test messages to publish to Kafka or setting up test data in a database or other system. Run the test cases: Execute the test cases and verify that the streaming API is functioning correctly. Monitor the performance of the system, including latency and throughput, to ensure that it is meeting the desired requirements. Check the output of the test cases to ensure that the expected results are being produced. Analyze the test results: Review the test results to identify any issues or areas for improvement. Document any issues that were found and the steps taken to resolve them. Repeat the testing process: After making any necessary changes, repeat the testing process to ensure that the streaming API is functioning correctly. Document the test plan: Document the test plan and the results of the testing process, including any issues that were identified and the steps taken to resolve them. Include details on the test environment, the test cases that were run, and the results of the tests. Include any performance metrics that were measured, such as latency and throughput.

X. RELATED SERVICES

[1] MonkeyLearn, this service provides Customized sen-timent analysis models. They train the models with the user's own data. [2] Repustate, this service supports text in 24 different languages and also has the capability to Detect the emotions expressed in emojis, slang, abbreviations, and hashtags. [3] Lexalytics, this service enables the creation of dashboards and visualizations from the user's data and this helps in gaining deeper insights into the customer emotions.

All the above services predict the sentiment using data that has already been collected and pre-processed. That's where our service stands out we provide the sentiment and analysis of tweets/data that come in real-time and we do it fairly quickly and accurately.

XI. RESULTS

Once the model training is done and the model is saved, we integrate it with the Backend where when we get the tweets from the Twitter API, the BERT model is used for prediction where we predict positive, negative and neutral sentiments. All the components are further integrated with the front-end where we can provide a user with a text box to enter the topic which he wants to get sentiment analysis. We have considered TESLA as an example where we get tweets related to Tesla and we predict the sentiment of the tweets in real-time. Figure 3 shows the front end. Once we enter the topic, we will start to get the tweets in the backend. Figure 4 shows the stream of tweets which we obtain from Twitter API when we hit the API.

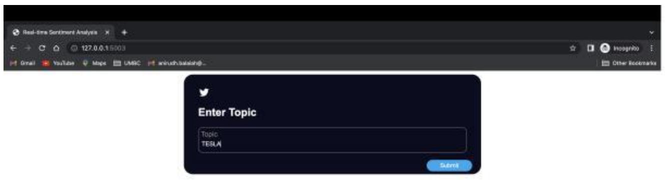
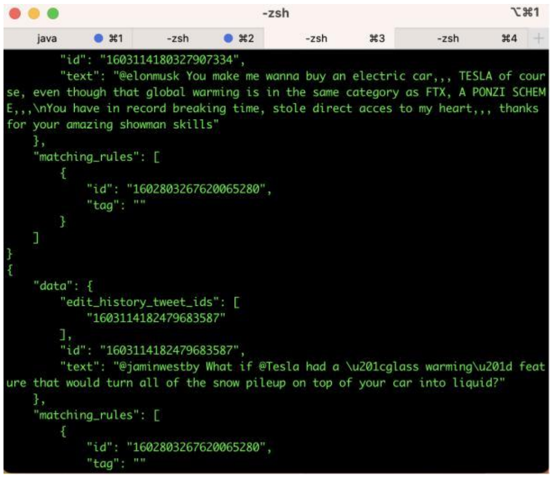


Fig. 3. Front-end to enter topic

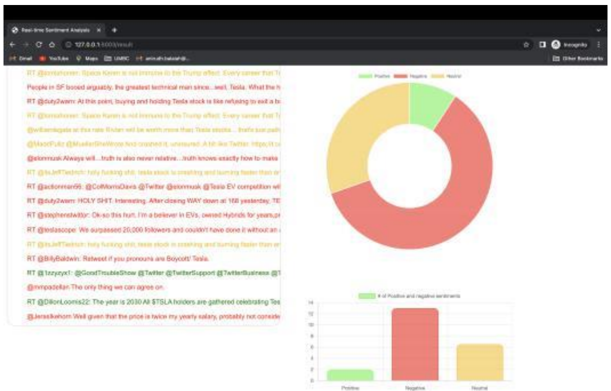
As shown in figure 4, the format of tweets is JSON and there will be a lot of unwanted text. The pre-processing of it is performed in the model cleaning section which is discussed in the above sections. Then the BERT model processes the text and converts it into a format that it can understand.



```
        "id": "1603114180327907334",
        "text": "@elonmusk You make me wanna buy an electric car,,, TESLA of cour
se, even though that global warming is in the same category as FTX, A PONZI SCHEM
E,,,\nYou have in record breaking time, stole direct acces to my heart,,, thanks
for your amazing showman skills"
    },
    "matching_rules": [
        {
            "id": "1602803267620065280",
            "tag": ""
        }
    ]
}
{
    "data": {
        "edit_history_tweet_ids": [
            "1603114182479683587"
        ],
        "id": "1603114182479683587",
        "text": "@jaminwestby What if @Tesla had a \u201cglass warming\u201d feat
ure that would turn all of the snow pileup on top of your car into liquid?"
    },
    "matching_rules": [
        {
            "id": "1602803267620065280",
            "tag": ""
```

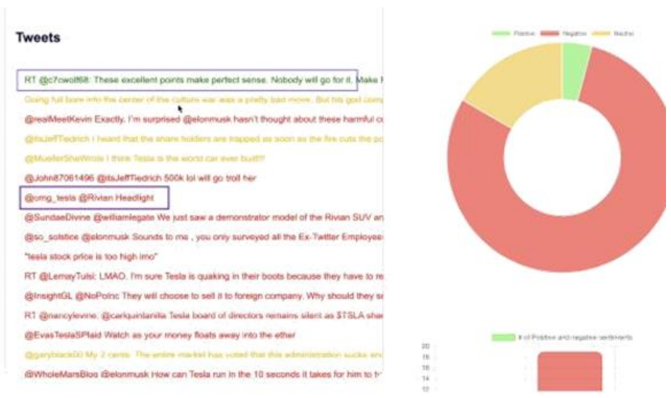
Fig. 4. Stream of tweets in the Backend

Once we get the tweets and predictions, it is sent to the front-end page we used ChartJS to render analytics and the resulting webpage is shown in Figure 5.



RT [illegible]
People in SF booed arguably the greatest technical man since...well, Tesla. What the h
RT @duty2warn: At this point, buying and holding Tesla stock is like refusing to exit a b
RT [illegible]
[illegible]
[illegible]
@elonmusk Always will...truth is also never relative...truth knows exactly how to make
RT [illegible]
RT @actionman95: @ColMorrisDavis @Twitter @elonmusk @Tesla EV competition wi
RT @duty2warn: HOLY SHIT. Interesting. After closing WAY down at 1[illegible] yesterday, TE
RT @stephenstwittor: Ok-so this hurt. I'm a believer in EVs, owned Hybrids for years,p
RT @teslascope: We surpassed 20,000 followers and couldn't have done it without an
RT [illegible]
RT @BillyBaldwin: Retweet if you pronouns are Boycott Tesla.
RT @1zzyzyx1: @GoodTroubleShow @Twitter @TwitterSupport @TwitterBusiness @T
@mmpadellan The only thing we can agree on.
RT @DillonLoomis22: The year is 2030 All $TSLA holders are gathered celebrating Tes
@Jerasikehorn Well given that the price is twice my yearly salary, probably not conside

Fig. 5. Front end result with Tweets and Charts



Tweets

RT @c7cwolf68: These excellent points make perfect sense. Nobody will go for it. Make I
Going full bore into the center of the culture war was a pretty bad move. But his god com
@realMeetKevin Exactly. I'm surprised @elonmusk hasn't thought about these harmful c
@itsJeffTiedrich I heard that the share holders are trapped as soon as the fire cuts the p
@MuelerSheWrote I think Tesla is the worst car ever built!!!
@John87061496 @itsJeffTiedrich 500k lol will go troll her
@omg_tesla @Rivian Headlight
@SundaeDivine @williamlegate We just saw a demonstrator model of the Rivian SUV an
@so_solstice @elonmusk Sounds to me , you only surveyed all the Ex-Twitter Employee
"tesla stock price is too high imo"
RT @LemayTulsi: LMAO. I'm sure Tesla is quaking in their boots because they have to re
@InsightGL @NoPoinc They will choose to sell it to foreign company. Why should they s
RT @nancylevine: @carlquintanilla Tesla board of directors remains silent as $TSLA sha
@EvasTeslaSPlaid Watch as your money floats away into the ether
@garyblack00 My 2 cents: The entire market has voted that this administration sucks an
@WholeMarsBlog @elonmusk How can Tesla run in the 10 seconds it takes for him to t

Fig. 6. Correct and Failed Predictions

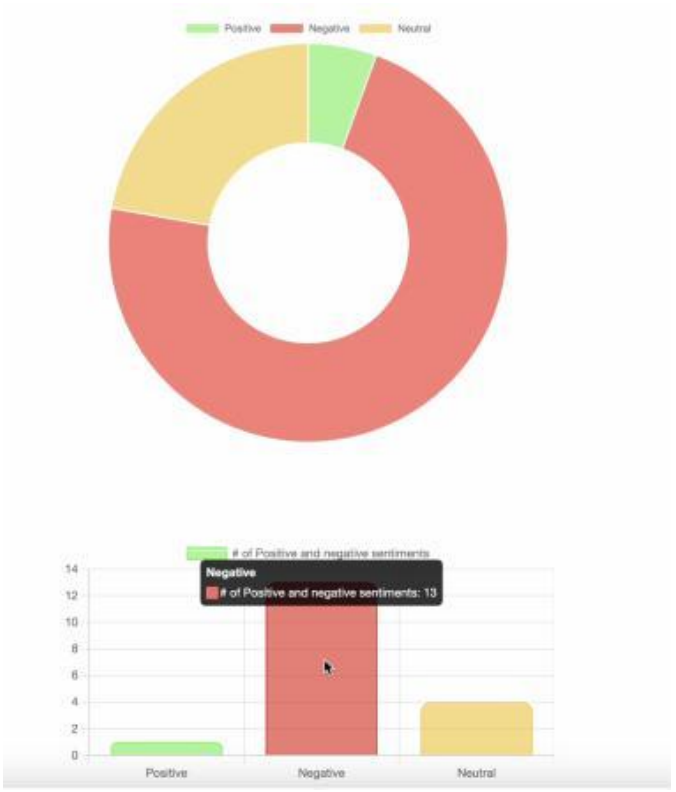


Fig. 7. Analytical Charts

To explain the final output webpage in detail, it displays all the tweets with colours where green indicates positive, red is negative and yellow is a neutral tweet. Figure 6 shows the situations where the model might succeed and fail. The blue box shows a failed scenario where the model predicts a negative when the sentence is neutral. The blue box shows a scenario where the prediction was accurate. In Figure 7, I hovered the mouse over the graph to indicate the number of negative tweets streamed so far. The same applies to other columns in the bar chart. The Pie chart shows the percentage of positive, negative and neutral tweets updated in real-time.

XII. FUTURE SCOPE

There are multiple areas where our project can be improved. The BERT model which is used in the project can be further improved by training on more data and longer. Since we had a limited GPU, we could not train it for more epochs. We can experiment with other transformer models since there are other state-of-the-art models better than BERT. Here, we are using just one stream of data to get the public sentiment. However, we can use multiple streams which can be easily integrated since we are using Kafka. We can add more metadata along with sentiment and display to the front-end rather than only sentiment results.

XIII. CHALLENGES

Training the BERT model was not an easy task since it involves a lot of components along with training. It took us some time to integrate WebSockets with Flask where sending the real-time data from Flask to the front end did not work and we had to spawn separate threads to achieve real-time communication. Normal HTTP communications did not work and had to find ways to tackle the problem of sending data in real-time to the front end. Faced difficulty in reducing latency and we tackled this problem by creating a separate server for hosting Kafka and Deep Learning model.

XIV. CONCLUSION

To conclude our project, we aimed in targeting customers who are looking for getting the sentiment of public opinion on a specific topic in real-time. We have utilized the power of Deep Learning to get accurate predictions and WebSockets technology to stream the topics to the front end in real-time.

The success of the project also leverages the Twitter streaming API to pull and filter the streams in real time. Analytics were generated in the front end to display the numbers and percentages of various sentiments of tweets. We were able to achieve the expected real-time experience for the user with very minimum latency even though a lot of things are happening in the background and a big Machine Learning model.

XV.REFERENCES

[1]. 'Text Analysis,' MonkeyLearn. https://monkeylearn.com

[2]. 'Unlock the hidden emotion with the best sentiment analysis tool,' www.repustate.com. https://www.repustate.com/sentiment-analysis/

[3]. 'Sentiment Analysis - Lexalytics,' www.lexalytics.com, May 16, 2022. https://www.lexalytics.com/technology/sentiment-analysis/

[4]. G. L. Muller, 'HTML5 WebSocket protocol and its application to distributed computing,' arXiv:1409.3367 [cs], Sep. 2014, Accessed: Dec. 22, 2022. [Online]. Available: https://arxiv.org/abs/1409.3367

[5]. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understand-ing,' arXiv.org, Oct. 11, 2018. https://arxiv.org/abs/1810.04805

[6]. 'Apache Kafka,' Apache Kafka. https://kafka.apache.org/documentation/

[7]. 'Filtered stream introduction,' developer.twitter.com. https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction

[8]. 'Welcome to Flask Flask Documentation (2.2.x),' flask.palletsprojects.com. https://flask.palletsprojects.com/en/2.2.x/

[9]. C. Richardson, 'Microservices.io,' microservices.io, 2017. https://microservices.io/patterns/microservices.html

[10]. Sarlan, Aliza & Nadam, Chayanit & Basri, Shuib. (2014). Twitter sentiment analysis. 212-216. 10.1109/ICIMU.2014.7066632.

Cite this article as :

Samit Shivadekar, Ketan Shahapure, Shivam Vibhute, Milton Halem, "Real-time Sentiment Analysis from Data Streaming", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 10, Issue 1, pp.60-70, January-February-2024. Available at doi : https://doi.org/10.32628/CSEIT2390646
Journal URL : https://ijsrcseit.com/CSEIT2390646