

# ID2221 Project

## Real-Time Sentiment Analysis of U.S. Election Candidates Using Streaming Data

Íñigo Aréjula  
inigoaa@kth.se

Jacopo Maragna  
jmaragna@kth.se

Júlia Tribó  
juliatic@kth.se

October 29, 2024

## 1 Problem description

We are nearing the crucial election between Donald Trump and Kamala Harris. To gain timely insights into public opinion, we aim to track and update real-time sentiment towards both candidates. This analysis will be based on data gathered from tweets, allowing us to continuously measure and outline the percentage of people expressing favourable views towards each candidate. The sentiment data will be updated dynamically, reflecting shifts in voter sentiment as they happen throughout the election cycle.

## 2 Tools

- **Kafka** will simulate a real-time data stream from an initial dataset of tweets downloaded from the social media platform X (formerly Twitter).
- **Apache Spark** will process the incoming data stream, utilizing the MapReduce paradigm to perform distributed computations.
- For text classification and sentiment analysis, we will use the pre-trained model **cardiffnlp/xlm-twitter-politics-sentiment** from Hugging Face.
- The final results will be stored in **MongoDB** for persistent storage and easy querying.
- The user can watch a chart in real-time using their browser. We created a web server using Go.

## 3 Data

Our data source is X (formerly Twitter). Initially, we considered consuming data directly from the official API; however, it cost hundreds of euros. As a result, we developed our own X scraper, which can extract data in real-time. For demo purposes, however, we have pre-loaded approximately 500 tweets into a file, and we will always consume data from this file.

## 4 Methodology and algorithm

We collected a large volume of tweets into a dataset and send them to Kafka to simulate a real-time data stream that feeds into Spark. Using Spark, we will implement a MapReduce process that includes sentiment analysis within its workflow.

For each tweet, positive sentiment towards a candidate will increase that candidate's favorability percentage, while negative sentiment will decrease it. Our classification algorithm will follow a series of steps designed to accurately classify the sentiment of each tweet and adjust the favorability score for the corresponding candidate accordingly. This approach will allow us to continuously update and reflect the real-time sentiment trends for both candidates as new data flows in.

## 5 Implementation

First, a Twitter scraper was designed to retrieve tweets containing the words "trump" or "kamala." The results for each candidate were stored in separate TSV files. Next, three Kafka brokers with two partitions each, along with a Zookeeper, were deployed. Additionally, a Kafka topic for each candidate was created: "trump\_tweets" and

“kamala\_tweets.” Each topic was initialized with two partitions and a replication factor of three, ensuring that each broker would have a copy of both partitions for each topic. This implementation is illustrated in Figure 1.

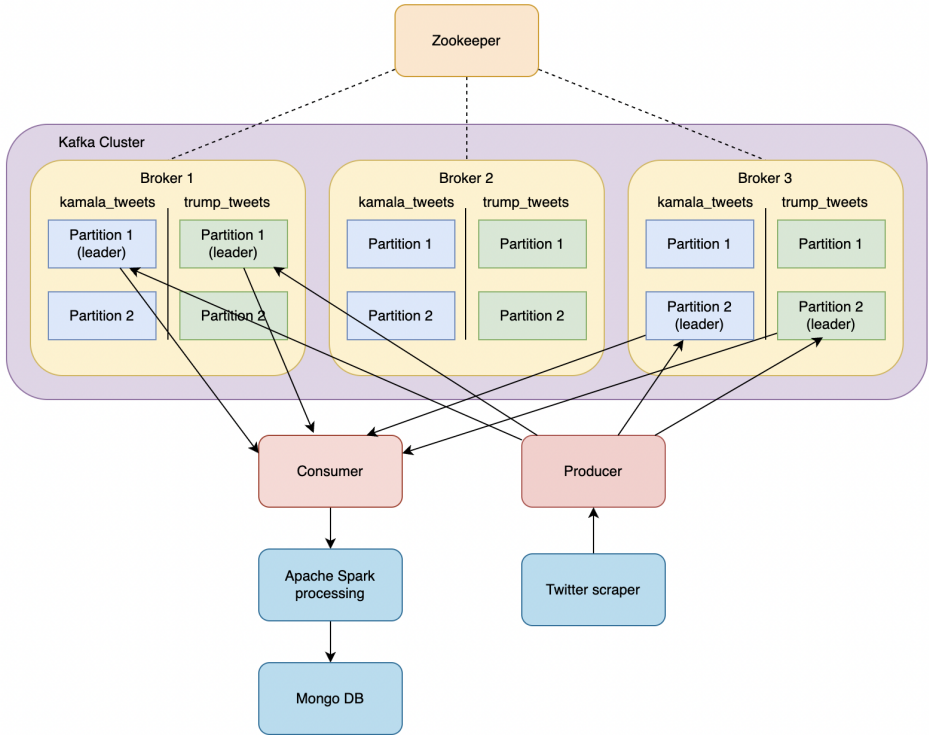



Figure 1: Kafka architecture

In addition, a UI for Kafka was also deployed to monitor the Kafka cluster in real time. As shown in Figure 2(a), all three brokers are up and running with four virtual partitions each (two for each topic). Figure 2(b) illustrates that both topics have been created with two partitions and a replication factor of three.

Uptime			Partitions				
Broker Count	Active Controller	Version	Online	URP	In Sync Replicas	Out Of Sync Replicas	
3	2	3.7-IV4	4 of 4	0	12 of 12	0	
↕ Broker ID	↕ Disk usage	↕ Partitions skew ⓘ	↕ Leaders	↕ Leader skew	↕ Online partitions	↕ Port	↕ Host
1	102.05 KB, 4 segment(s)	-	2	-	4	19092	kafka1
2 	102.05 KB, 4 segment(s)	-	2	-	4	19093	kafka2
3	102.05 KB, 4 segment(s)	-	2	-	4	19094	kafka3

(a) Kafka brokers

Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
kamala_tweets	2	0	3	281	256 KB
trump_tweets	2	0	3	561	373 KB

(b) Kafka topics

Figure 2: Kafka-ui view

After creating the topics, a producer was implemented to read the corresponding TSV files and send a tweet every 0.25 seconds to those topics, simulating a real-time data stream.

Our Spark service, configured as a Kafka consumer, is set up to read streams from two topics: trump\_tweets and kamala\_tweets. Each topic is processed by two separate streaming jobs, where data is handled in batches. For each candidate, we manage the corresponding data stream and, within each mini-batch, we predict the sentiment (positive or negative) of each tweet.

After processing each batch, the results are aggregated, counting the positive and negative sentiments. These aggregated results are then sent to MongoDB, with updates made in real-time for each candidate to keep the dashboard up-to-date with fresh insights.

Sentiment classification is powered by a transformer model fine-tuned on political tweets. This model is an extension of the multilingual twitter-xlm-roberta-base-sentiment model, focusing specifically on sentiment analysis for politicians’ tweets.

Given the size of the model, special consideration was required to efficiently integrate it with our Spark infrastructure. Within our Docker-Compose deployment, we treated the model as a dependency of the consumer service. The question arose: how can each Spark worker leverage this model to make predictions?

Here’s the solution: the model is initially loaded into memory by the main thread of the consumer service. It is then broadcasted to all worker nodes using sparkContext, enabling each worker to access the model for inference. This approach avoids the need for each worker to load the model independently, which would be both computationally and memory-intensive, and impractical for real-time streaming.

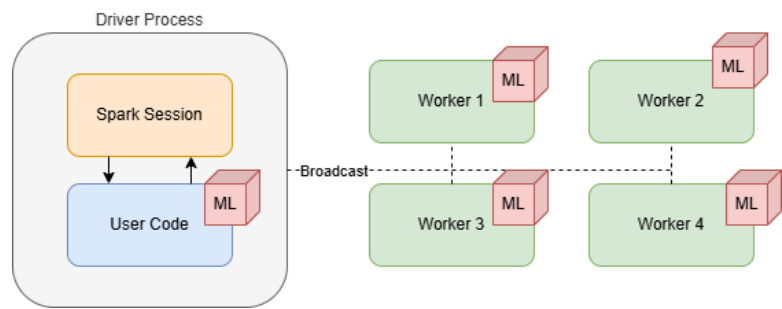


Figure 3: Model Broadcast

Once the tweets in each mini-batch are classified within Spark’s map function, the results are reduced by key (positive or negative) and counted. Finally, the driver collects these results from the worker nodes and sends them to MongoDB for storage.

The database was organized with one append-only collection for each candidate, containing the attributes: id, timestamp, support, and oppose. So that when a tweet is classified as positive, the support counter is incremented; conversely, if the tweet is classified as negative, the oppose counter is incremented.

Finally, a dashboard was implemented to represent the data by showing the winning percentage of each candidate in real-time. To compute the percentage, the current support of a candidate was added to the opposition of the opponent and then divided by the total number of current votes. In Figure 4, a demonstration of the dashboard is presented.

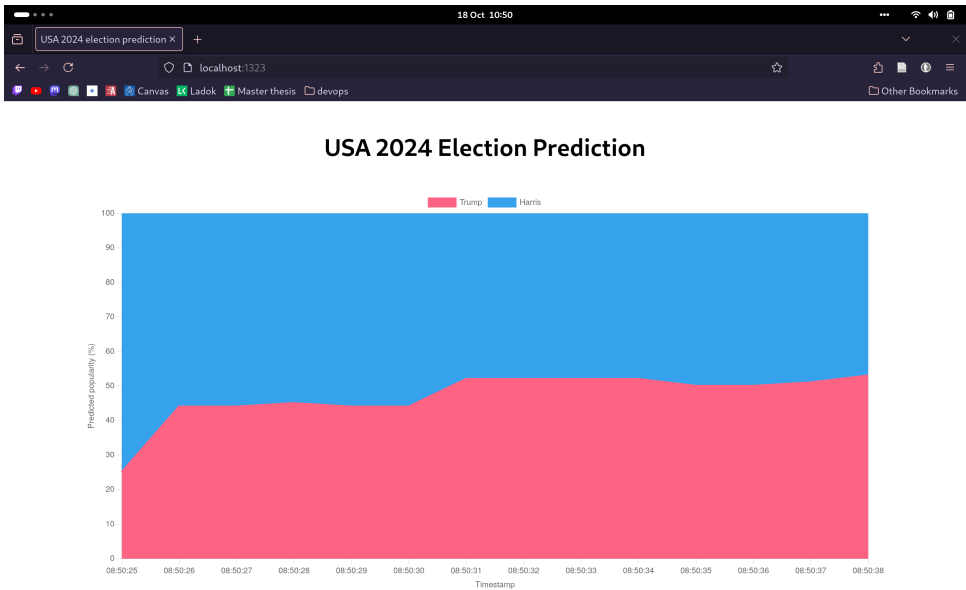


Figure 4: Dashboard view

## 6 Conclusion

We successfully implemented and deployed a distributed data-intensive pipeline. This system processes data from the raw source—Twitter—where we gather the information, all the way to a database and a dashboard that displays the results. Notably, we simulated a real-world scenario by deploying the services across several nodes, enabling our system to demonstrate recovery scenarios.

Using the map-reduce pattern on Spark in our pipeline demonstrates how a complex computation can be easily parallelized. It allows interaction with the data as if it were a local array, while in reality, a more complex process is running behind the scenes.