# Spark Project

Efoe Etienne Blavo

2. Mai 2023

**Universite Paris Saclay**

# 1 Introduction

Nowadays, we are witnessing a surge in user-generated content on live streaming platforms, with Twitch being a prime example. Analyzing chat sentiments in real-time can offer valuable insights into users' perceptions, opinions, and emotions about the content being streamed. In this project, I aim to develop a system to collect chat messages from Twitch, perform sentiment analysis, and apply a clustering algorithm to group messages based on their sentiment values. Additionally, I will create a dynamic dashboard to visualize the sentiment distribution and cluster sizes in real-time.

# 2 Processing API Data as a Stream

### 2.0.1 Streaming API (Twitch API)

Twitch is a streaming platform primarily focused on game streaming, which generates a significant amount of data through its chat system associated with various channels. To access these chat messages via the API, I followed these steps:

After creating my Twitch account, I proceeded to create an app on Twitch's development portal https://dev.twitch.tv/console/apps/create. Once the application was created, a Client ID and Client Secret were automatically generated.

Using the Client ID, I generated an access token essential for authenticating requests to the Twitch API. Generating the access token is fairly simple. I used the HTTPS link provided below, which redirected me to a localhost link, granting access to the access token:

https://id.twitch.tv/oauth2/authorize?response_type=token&client_id=<YOUR_CLIENT_ID>&redirect_uri=http://localhost:3000&scope=channel%3Amanage%3Apolls+channel%3Aread%3Apolls&state=c3ab8aa609ea11e793ae92361f002671

Replace <YOUR_CLIENT_ID> with your actual Client ID.

### 2.0.2 Python server (server.ipynb)

Upon obtaining the access token, I can securely interact with the API. I have developed a Python server that establishes a connection to the Twitch API, specifically to a chosen channel's chat, and relays the chat data to a connected client as a JSON string in real time.

To build the server, I followed these steps:

1. Configure the authentication details and settings for the Twitch IRC server (e.g., specifying the channel name (I choose **pokelawls**) to connect to, the access token, my Twich username) and the local socket settings (host, port, etc.).
2. Create a **parse_message** function to extract the username and message from the Twitch IRC response using regular expressions.
3. Implement the main function to initiate the server.

The main function performs the following tasks:

- Establishes and connects to the Twitch IRC socket.

- Sends authentication and channel membership information.
- Creates and configures the server socket.
- Listens for incoming connections and accepts client connections.
- Continuously processes the Twitch IRC response in a loop and forwards it to the client as a JSON string.

The JSON string sent to the client is in this form:

```
msg_data = {
    "date": time.strftime("%Y-%m-%d␣%H:%M:%S", time.localtime()),
    "username": username,
    "message": message
}
```

```
:zelfarso!zelfarso@zelfarso.tmi.twitch.tv PRIVMSG #pokelawls :gm shy
:toobie_!toobie_@toobie_.tmi.twitch.tv PRIVMSG #pokelawls :gm shy catPAT
:zelfarso!zelfarso@zelfarso.tmi.twitch.tv PRIVMSG #pokelawls :@pokelawls, new house tour now right? Awkward
:swaybread!swaybread@swaybread.tmi.twitch.tv PRIVMSG #pokelawls :catPAT
:swaybread!swaybread@swaybread.tmi.twitch.tv PRIVMSG #pokelawls :shyMad add catHide
:toobie_!toobie_@toobie_.tmi.twitch.tv PRIVMSG #pokelawls :shy add menherainsane back
:toobie_!toobie_@toobie_.tmi.twitch.tv PRIVMSG #pokelawls :ReallyGun
```

fig 1: Response received from Twich IRC.

From (fig 1) , it is evident that the only valuable information we can extract is the username and the message content. Twitch refrains from providing user locations to uphold security and privacy concerns.

## 3 Spark Streaming (spark_streaming.ipynb)

The implementation of sentiment analysis for messages received from the server is achieved using Spark Streaming and the TextBlob library. I developed a **sentiment_analysis** function that employs TextBlob to assess the sentiment of a message, returning a score of -1, 1, or 0 based on the polarity provided by TextBlob after analysing the given message.

Subsequently, I defined the **process_rdd** function to handle each RDD in the DStream, performing the following actions:

- Extracts the message from each data element in the RDD and analyzes its sentiment by calling the **sentiment_analysis** function.
- Applies a K-means clustering model, initialized with three clusters and random centers, and trains it on the sentiment values extracted from the messages to assign a cluster to each message.

- Displays the cluster size in the sliding window.
- Converts the sentiment value to a string (-1 = 'Negative', 0 = 'Neutral', 1 = 'Positive').

```
Cluster sizes in the sliding window:
  Cluster 1: 1 messages
  Cluster 2: 2 messages


--------------------------------------------------
Timestamp:      [2023-04-30 20:09:06]
Username:       zelfarso
Sentiment:      Positive
Message:        better twitch adblock
Cluster:        2
--------------------------------------------------


Cluster sizes in the sliding window:
  Cluster 1: 1 messages
  Cluster 2: 2 messages


--------------------------------------------------
Timestamp:      [2023-04-30 20:09:08]
Username:       zelfarso
Sentiment:      Neutral
Message:        sometimes works
Cluster:        2
--------------------------------------------------


Cluster sizes in the sliding window:
  Cluster 1: 1 messages


--------------------------------------------------
Timestamp:      [2023-04-30 20:09:08]
Username:       zelfarso
Sentiment:      Neutral
Message:        sometimes works
Cluster:        1
--------------------------------------------------
```

fig 2: Cluster size - Sentiment - Message - username - date

As depicted in the (fig 2), we obtain the results when executing the Spark Streaming file.

To enable real-time and dynamic data visualization, I opted to create an SQLite database and store all pertinent informations for visualization, including cluster, message, and sentiment.

## 4 Visualization (dashboard.ipynb)

In this section, I developed a dynamic dashboard to display the sentiment analysis results of messages and their corresponding clustering. To achieve this, I first developed a **fetch_new_messages** function that connects to the SQLite database, retrieves all messages from the **messages** table, and returns the results as a tuple.

Next, I implemented the primary function, **display_dashboard**, which creates a DataFrame from the results returned by **fetch_new_messages** and subsequently generates a bar chart displaying sentiment counts per cluster. The **display_dashboard** function is executed in a loop, allowing it to update information by considering new messages every 10 seconds. See (fig 3)

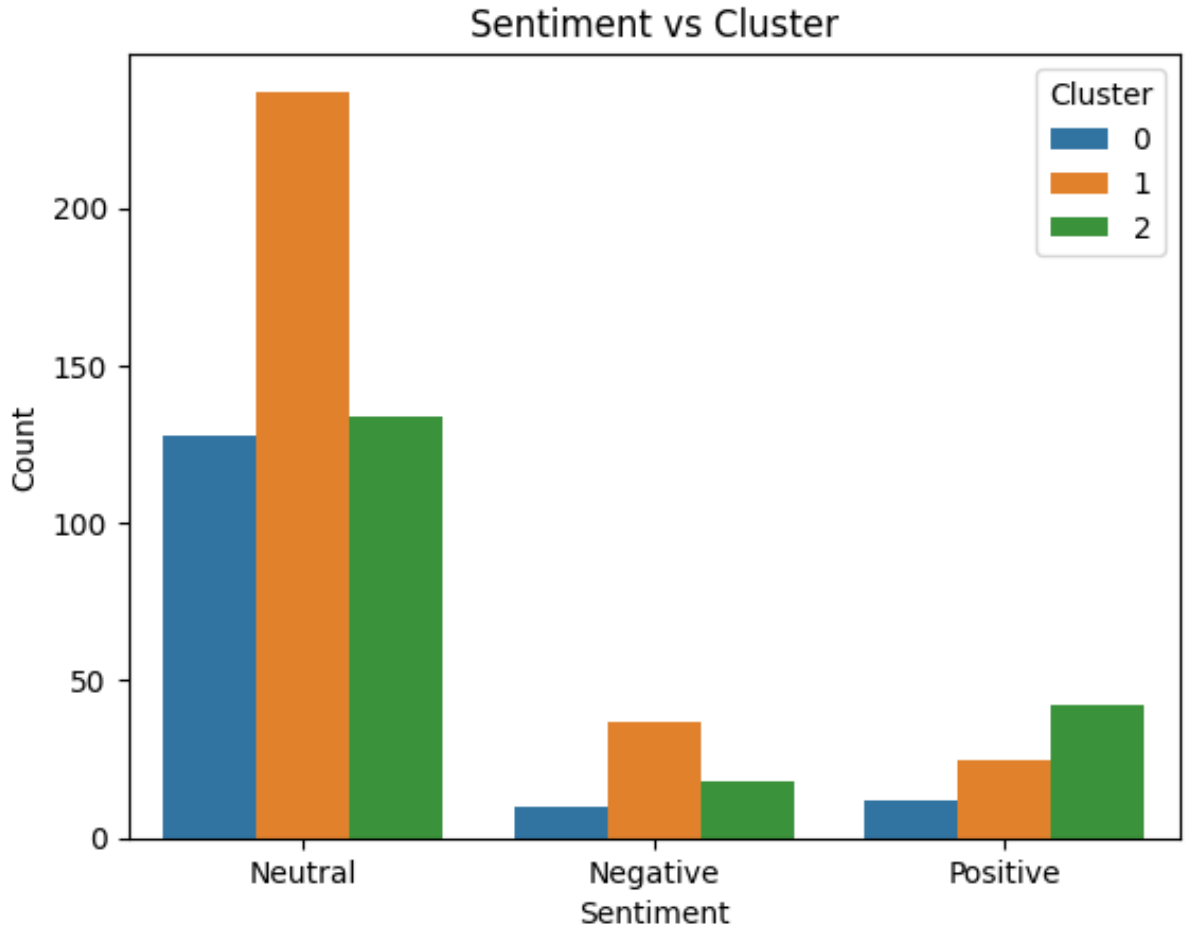


fig 3: Sentiment counts per cluster

## 5 Conclusion

This project demonstrates the potential for extracting meaningful insights from chat messages on live streaming platforms. The information gathered can be used to improve content quality, tailor advertising campaigns, and enhance user experience. Future work could include refining the sentiment analysis process, exploring additional clustering techniques, and expanding the dashboard's capabilities to include more detailed visualizations and analytics.