Large Scale Twitch Chat Sentiment Analysis

Altan Senel
altans@kth.se

Fehmi Ayberk Uckun
uckun@kth.se

Oliver Möller
olivermo@kth.se

Code Repository

https://github.com/MoellerO/twitch_chat_sentiment_analysis

Abstract

With a total of 2.78 million concurrent viewers in 2021 and a combined watch time of 5.65 billion hours in the second quarter of 2022, Twitch is the leading gaming live-stream platform in the World. One reason for Twitch's success is the built-in Live Chat, which lets users directly communicate with the Streamer. Depending on the content of the stream, the mood in the chat can vary. For example, if the Streamer (the community favorite) loses a game, the mood in the chat may decrease, while people may be happy if they win a game. Therefore, we found it interesting to build a Stream-Processing application that analyzes sentiment on Twitch Chats in real-time. To determine the sentiment, we scrape the data directly from the Twitch Chat and use roBERTa-base Model, a sentiment analysis Model specifically attuned to sentiments expressed in social media (Twitter). Furthermore, utilizing Kafka [1] and Spark Structured Streaming [2], we provide a horizontally-scaling system architecture capable of analyzing multiple Twitch Chats simultaneously, supplying the ideal base for extensive data analysis on Twitch.

1 System Architecture

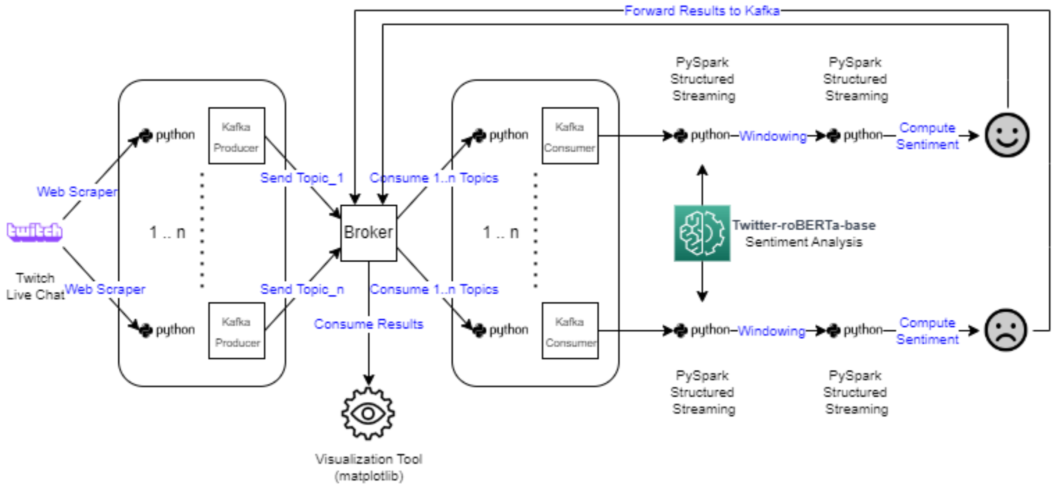

Figure 1: Architecture Diagram

Overview Figure 1 shows the architecture diagram of our application. The application is split into four main parts. The first part is the producer. The producer is responsible for accessing messages from the chat of a specified Twitch Channel and sending them to a distributed message Broker (Kafka). The broker is the second part of our application. It is accountable for storing the different Data Streams (Topics) and provides the functionality that consumers can subscribe to one or multiple data streams. Consumers build the final part of our application. We implemented the consumers using PySpark and Structured Streaming. Each consumer is responsible for receiving chat messages from one or multiple twitch chats and performing real-time sentiment analysis on it. Using sliding windows, the consumer provides a live score for the overall sentiment of a given Twitch chat (or when subscribing to multiple channels) to a category of twitch chats and sends the results back to the Kafka Broker. Now multiple analyzing applications can subscribe and further process the results. In our case, we connected a simple python program that live-plots the results as a Graph (Figure 2).

Data source Our data sources are messages from the chats of an up-to-preference chosen Twitch Channels. We wrote a web scraper to access the messages from a twitch chat. The web scraper uses ChromeDriver, to emulate a Chrome Session. We can scrape the messages by sequentially selecting the HTML Element corresponding to the last message sent.

Sentiment Analysis The Twitch Chat uses very colloquially and informal language. Therefore, we decided to use the Twitter-roBERTa-base Model for Sentiment Analysis. The model is trained on 58M tweets and, therefore, familiar with the slang and language in social media. For simplicity, the sentiment is predicted in three classes: positive, neutral, and negative. Neutral messages are removed before calculating the average in order to prevent all averages to collapse to zero. The model is downloaded from *https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest*.

Message	**Score**
voter intimidation laws	-0.64
bubby62 just re-subbed for 16 months! POGGIES	0.92
count the votes! but also stop the count! but also dont count these	-0.55
Dude the video of this is insane	0.66
here in Australia candidate supporters pester you while you line up to vote, and try to bribe you with hotdogs @hasanabi	-0.74

Table 1: Example Sentiment scores on Twitch messages. Positive scores represent positive sentiment, negative scores represent negative sentiment.

Kafka Producer The web scraper provides the application with all unseen messages once per second. The application then acts as a Kafka Producer, creates a topic (named after the twitch channel), and forwards the messages to the broker. The design decision to use Kafka provides several advantages. First, it decouples the web scraper from the data processing unit, allowing for asynchronous processing. Second, by creating more web scraper instances, we can scrape from multiple streams synchronously, with each scraper acting as an independent producer sending to a different topic. The broker can easily handle the increased message load, as Kafka scales well horizontally. The thrid advantage is, that with this architecture a single consumer can simply subscribe to multiple topics (twitch chats) and perform analysis over a category of twitch channels. Hence, this system is suitable for extensive twitch data analysis.

Kafka Broker & Zookeeper To facilitate the setup of our application, we use prebuilt docker images for the Kafka broker and zookeeper. Within the YAML file, we can define further configurations, such as the topic replication factor of the broker. Docker-Compose then allows us to start both with a single command. Note that we host both using our local machine. However, this is not a requirement, and any server could host the broker and zookeeper.

Kafka Consumer & PySpark Structured Streaming To realize the Kafka consumers, we make use of Spark Structured Streaming. Spark Structured Streaming can directly read from Kafka and is ideal for processing the data. We map two user-defined functions over the data stream. The first one translates the Kafka incoming byte stream into Strings, and the second infers each message's sentiment. To decrease the variance, we use overlapping sliding windows (window duration 2m, slide duration 1m) and compute the chat's sentiment as the mean of the sentiments of every messages within the window. Each new result is streamed back to the Kafka Broker.

2 Results

In theory, an arbitrary number of analyzing applications can now subscribe to the Kafka topic holding the sentiment inference results and further processing them. We created a simple Python program that makes a live graph covering the chat's sentiment over the last five minutes. For the visualization, we use a moving average. Each bar is computed as the average sentiment from the previous and the current 2-minute window. Note that starting at 18:57, one of the players in the stream was losing, significantly shifting the mood in the chat.

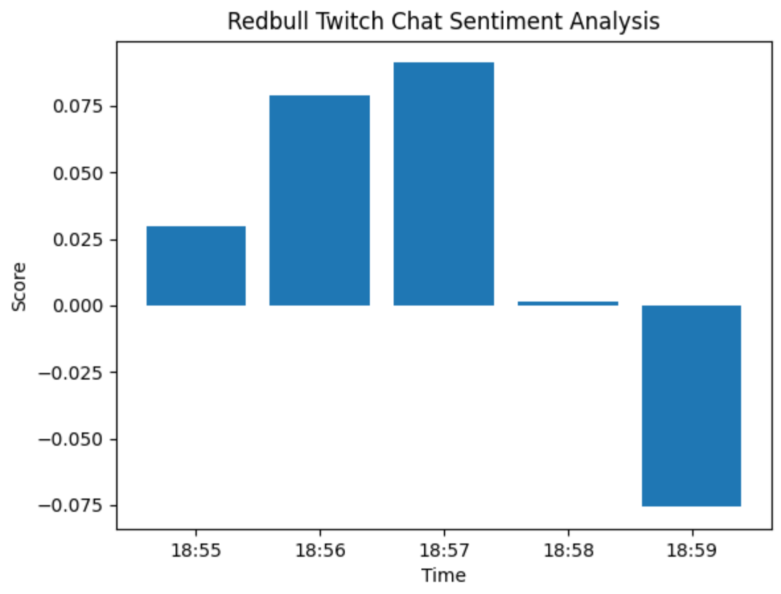

Figure 2: Moving Average over a sliding window (Size = 2 mins, Slide = 1 min). Positive Scores are representing positive sentiment, negative scores are representing negative sentiment. The analyzed chat originates from a Red Bull Livestream showing an Age of Empires IV game.

3 How to run the code

Link to the code https://github.com/MoellerO/twitch_chat_sentiment_analysis

Prerequisites

- Ensure that Java, Hadoop, and Spark are installed, and environment variables are correctly set. Especially on Windows, we found it quite tedious to get Spark Sessions to work.
- Ensure Docker is installed.
- Install the required packages from *requirements.txt*. All packages can be installed using the *pip installer*.
- Sentiment Analyzer uses a PyTorch model, PyTorch can be installed using the commented line in the requirements.txt or you can install the version suitable to your setup.

Start Kafka Broker and Zookeeper

- If preferred, change the configuration of Broker and Zookeeper within the *docker-compose.YAML* file.
- Run *docker compose up -d* to start Broker and Zookeeper
- On success, the application will print "Starting zookeeper ... done" and "Starting broker ... done" to the console.

Start the Producer

- The producer is located in *scraper.py*
- Type *python scraper.py –channel <channel_name>* to start the producer.
- The producer will now open a chrome browser window, scrape the chat, and send the messages to the Kafka Broker

Start the Sentiment Analyzer

- The Sentiment Analyzer is located in *analyzer.py*
- Manually change the *TWITCH_USERNAME_LIST* variable in the code with your channel.
- Type *python analyzer.py* to start the analyzer.
- The analyzer is now going to consume the messages sent to Kafka, process them and send the results of each window back to Kafka.

Start the Visualizer

- The visualizer is located in *visualizer.py*
- Before starting the consumer, verify that the variable TWITCH_USERNAME_LIST contains the same Twitch channel name as the producer.
- Type *python visualizer.py* to start the consumer
- The consumer will now receive messages, infer the sentiment, and print the result to the console.

References

[1] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: a Distributed Messaging System for Log Processing. page 7, 2011.

[2] Michael Armbrust, Tathagata Das, Joseph Torres, Burak Yavuz, Shixiong Zhu, Reynold Xin, Ali Ghodsi, Ion Stoica, and Matei Zaharia. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 601–613, New York, NY, USA, May 2018. Association for Computing Machinery.