Rahul Tholakapalli

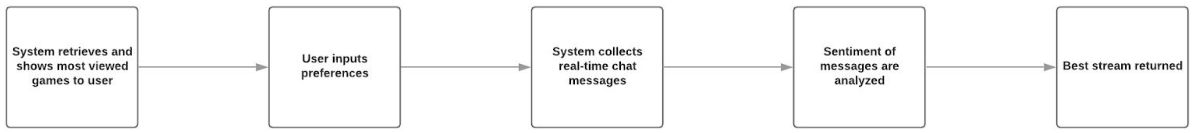
Twitch Channel Recommender

High Level Overview

My web app recommends Twitch channels based on user inputted preferences and real-time analysis of chat messages.

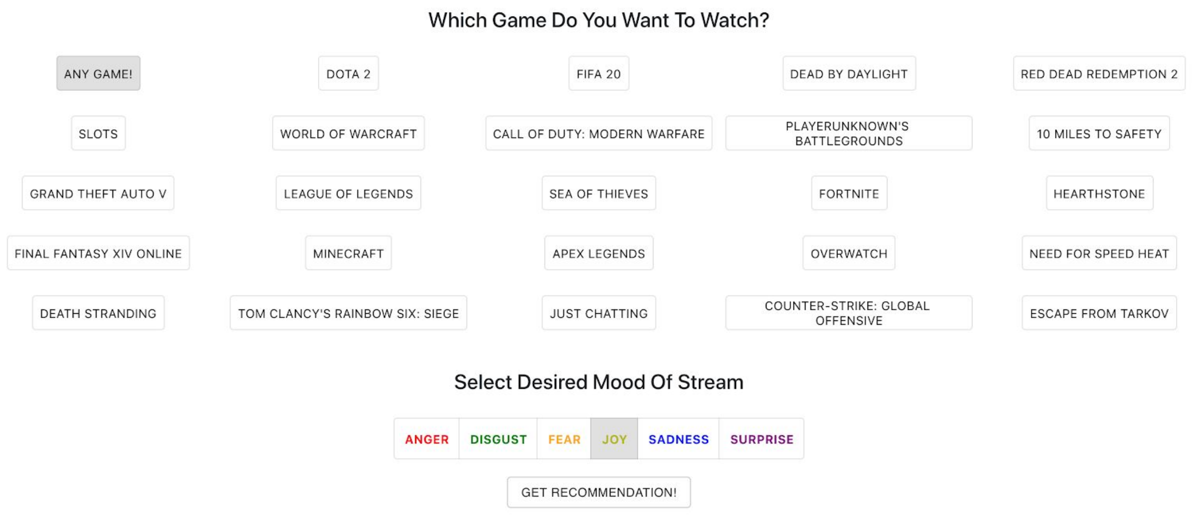
Video demo of functionality: https://www.youtube.com/watch?v=R-YS_NFtGGw

This diagram illustrates the basic functionality at a high level.


System retrieves and shows most viewed games to user
User inputs preferences
System collects real-time chat messages
Sentiment of messages are analyzed
Best stream returned

Live Game and Message Collection

The system starts by using a Twitch API endpoint to get the currently most viewed games on the site. The reasoning behind this is that the more viewers there are, the more chatters there are, and thus the faster and better the recommendation will be. Once this data is retrieved, the user is presented with the following UI:



The UI was created using JavaScript with the ReactJS framework.

Once the user inputs their preferences, the system gathers 50 messages each from the top five Twitch channels according to the user's input. If the user selects a specific game, it will retrieve the top five channels that are playing that game. Otherwise, it will retrieve the top five channels across all categories. In order to maximize efficiency, I used multithreading to collect messages from all channels simultaneously.

Message Analysis

The next step is to analyze the collected messages in order to get the most prominent emotion of the channel at the moment. Unfortunately, it is not sufficient to use a traditional mood analyzer. This is because Twitch has emotes (similar to emojis) that are almost always the most significant indicator of mood in Twitch chat. Traditional analyzers are designed for regular text such as tweets, and are not trained on Twitch emotes themselves.

In order to solve this issue, I mapped emote-word pairs to emotions. To do this, I first analyzed messages that have both emotes and text e.g. "LUL that was awesome". Then, I pass the message without emotes to a traditional mood analyzer (I'm using this open-source one from GitHub: https://github.com/nikicc/twitter-emotion-recognition) in order to get the emotion e.g. "that was awesome" -> Joy. Then, the emote-less message is cleaned (remove punctuation, stopwords, etc.) and the resulting words are used to create emote-word pairs associated with the outputted emotion e.g. (LUL, awesome) -> Joy.

Emote-word pairs may map to different emotions, so when conflicting mappings arise, I took the most common mapping. For example, we may have the following case where multiple messages result in conflicting mappings:

- "LUL that was awesome!" | (LUL, awesome) -> Joy
- "LUL that was not awesome, you suck!" | (LUL, awesome) -> Angry

To resolve this issue, I simply took the most common mapping for the pair. Since the pairing between "LUL" and "awesome" is used more commonly in a joyous context, the first mapping of (LUL, awesome) -> Joy is used.

The mapping for an individual emote may look like this:

- (LUL, awesome) -> Joy
- (LUL, great) -> Joy
- (LUL, happy) -> Joy
- (LUL, suck) -> Angry

Once all pairs are created, I set a default mapping in the case that an emote is passed in without any accompanying text. This default mapping corresponds to the most common emotion associated with the emote. In the case of the above example, this would result in the pair (LUL, None) -> Joy since Joy is the most common emotion associated with LUL.

Once all the pairs were collected and mapped to their appropriate emotions, I trained a machine learning model mapping pairs to their respective emotions. I used an SVM with SKLearn as my model. I couldn't directly input the pairs into the machine learning model, however, because ML models need numerical inputs and can't operate directly on raw text. To convert text into numerical representations, I trained a Word2Vec model (using the code here: https://github.com/evanslt/BlogCode/tree/master/NLP) on a large amount of Twitch data to map words and emotes to vectors. Word2Vec maps words that are similar in meaning closer together in vector space. For example, after training the model, the vector mapping may look like the following:

- great -> [1.1, 1, 1]
- good -> [1.2, 1, 1]
- bad -> [-3, -5, -10]

Words that are similar in meaning, such as "great" and "good" are very close together in vector space, while words with different meanings such as "bad" are very far away. The conversion allows text to be input into a machine learning model while still being able to retain relative meanings.

The final step was to actually analyze an individual message. Let's say we have the example "OhMyDog, that dog came in unexpectedly". How can we pick the most meaningful emote and word in a message to pass into our model? To get the most meaningful emote, the highest occurring emote in the message is chosen. In the case where there are multiple emotes that have equally high frequencies, the emote that generally occurs more frequently across Twitch is selected. This is because individual emotes typically have equally high significance in terms of indicating emotion, and an emote that occurs more often will have more data trained on it so we can be more sure of that emote's associated emotions. In order to get the most meaningful word, I used a different approach. First, I trained a TFIDF (term frequency–inverse document frequency) model to get relative frequencies of words. It has been shown that usually the less common a word is, the more significant it is to the meaning of a sentence. To analyze our example message, let's say we have the following frequencies determined from our TFIDF model:

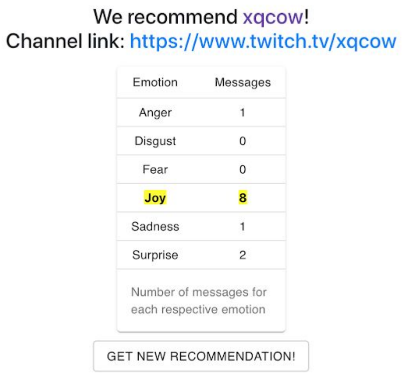
- dog -> high frequency
- came -> high frequency
- unexpected -> medium frequency

The highest occuring emote is OhMyDog, and the lowest frequency word is unexpected. Thus, we would pass in (OhMyDog, unexpected) into our model and probably get “Surprise” as a result. If we were to pass in (OhMyDog, dog) instead, it is likely that we would get “Joy” instead (since dogs are usually positively connotated) and the analysis of the message would be less accurate.

There are a few additional design choices I made. After analyzing some of the results, I added some manual overrides in areas that I thought the model lacked accuracy. This was mainly custom mapping emotes that are less common to their respective emotions because there wasn’t enough training data to get an accurate result. Additionally, I made the choice to only analyze messages that are only emotes or both emotes and text. This is partly because the raw mood analyzer is relatively slow, but more importantly because messages without emotes tend to be more analytical and not as indicative of the mood of the stream.

Retrieving and Returning Best Channel

To choose the best channel after analyzing all the messages, the channel with the most messages that corresponded to the user selected emotion is returned. For example, if the Twitch channel “Tfue” had five messages that were joyful, and the channel “Xqcow” had eight messages that were joyful in the same time period, Xqcow would be returned. Here is what the user would see after receiving the recommendation:


We recommend xqcow!
Channel link: https://www.twitch.tv/xqcow

Emotion	Messages
Anger	1
Disgust	0
Fear	0
Joy	8
Sadness	1
Surprise	2

Number of messages for each respective emotion

Concluding Remarks

In terms of performance, there are several areas for improvement. The system worked better when something very significant happened in the stream because that would trigger a strong response in the form of a flood of emotes. When streams were more calm, the recommendation system wasn’t able to return as good of a result since most of the chat messages were more analytical. I think one of the biggest improvements I could make would be to simply train on more data. For Word2Vec, the recommended training dataset size is equivalent to the entire Wikipedia corpus. I simply didn’t have time to collect and train on that much data. Additionally, there are additional sets of emotes known as “BetterTTV” and “FrankerZ” emotes. These weren’t considered to be emotes in my analysis, but they have become very widely used on Twitch recently, even more so than regular emotes in some cases. Including these extra emotes in my analysis would most likely provide more accurate results. Overall, however, the recommendation system performs well and I’m quite happy with how it turned out.