Twitch Recommendation System

Group Members: Nideesh Bharath Kumar (Section 3), Ritwika Das (Section 2), Shriya Shrestha (Section 3)

Google Colab Link | Video Link

Github Repository Link

Introduction:

What problem are we solving?

The project addresses the challenge of aligning Twitch users with streams where they can experience meaningful engagement. Existing recommendation systems prioritize content-based features or popularity metrics but fail to consider the critical role of chat dynamics, such as sentiment, activity levels, and prevalent topics, in fostering user satisfaction. This gap often leads to mismatches between users and stream communities, negatively impacting engagement and retention.

How do we plan to solve it?

To solve this, we will develop a supervised learning-based recommendation system that classifies streams based on their chat environment and matches them to user preferences. By leveraging Twitch's API to collect real-time chat data, we will use natural language processing (NLP) techniques to extract text embeddings and train predictive models. These models will classify streams into categories (e.g., supportive, competitive, casual), while user preferences will be captured through embeddings based on historical and real-time behavior. The system will then employ similarity metrics, such as cosine similarity, to recommend streams where users are likely to engage positively.

Relation to Course

This approach builds directly on concepts from supervised learning and natural language processing covered in class.

1. **Supervised Learning Techniques:** We apply these to train models that classify chat data into predefined categories based on labeled training data.
2. **Text Embeddings:** The use of transformer-based architectures, such as BERT, to generate embeddings ties into our discussions on advanced NLP techniques.
3. **Similarity Metrics:** Techniques like cosine similarity are applied to measure the relevance of recommendations, a concept discussed in the context of feature engineering and vector representation.
4. **Real-Time Analytics:** The challenge of processing dynamic and real-time data reflects topics in scalability and efficient computation from lectures.
   By integrating these elements, the project not only utilizes foundational knowledge but also extends it to address a complex, real-world application.

Motivation:

Why is our project important?

Our project is important because it enhances the way viewers discover Twitch streamers by matching recommendations with the emotional tone of a streamer's community. Traditional recommendation systems often rely on game titles or viewer counts, overlooking the unique vibe of each stream. By using sentiment analysis on live chat data, our system helps users find communities that match their mood or preferences, and active streamers they can watch based on the tone of the current chats. This overall leads to a more personalized, engaging, and satisfying viewing experience, which improves user retention and also supports smaller or niche streamers by connecting them with the right audiences.

Why are we excited about it?

This project is exciting because it combines real-time data analysis, natural language processing, and community dynamics to solve a meaningful problem in a creative way. It goes beyond surface-level metrics to understand the emotional pulse of Twitch communities, making recommendations feel more human and personalized. It also has the potential to elevate viewer experiences and give more visibility to streamers, which makes the impact feel both technically and socially rewarding.

What are some existing questions in the area?

1. **How can recommendation systems incorporate social dynamics effectively?**
   While many platforms focus on content-based or popularity-driven recommendations, integrating social and behavioral factors, such as chat dynamics or community interaction is still an open question.

2. **What role does sentiment analysis play in improving user engagement?**
   Understanding the emotional tone of chats and its effect on user retention is critical, but challenging, especially for real-time and dynamic platforms like Twitch.

3. **How can context-aware systems enhance personalization?**
   Questions remain on how to best integrate contextual factors like time of day, user mood, and past interaction history into recommendation systems.

4. **How do real-time analytics impact recommendation performance?**
   Real-time updates of user preferences and chat environments pose challenges in terms of scalability and computational efficiency.

5. **What are the ethical implications of personalized recommendations?**
   This includes questions around user data privacy, potential biases in recommendations, and promoting inclusivity for niche communities.

Prior Related Works

1. **Traditional Recommendation Systems**
   Many existing platforms, such as Netflix and YouTube, rely on collaborative filtering, content-based filtering, or hybrid models. These approaches prioritize user preferences based on historical data but often lack insights into community or social interaction dynamics.

2. **Sentiment-Aware Recommendations**
   Research has explored sentiment analysis for platforms like YouTube and Discord. Studies show that sentiment-aware systems can enhance user satisfaction by tailoring content to their emotional states. For instance, systems that recommend uplifting content to users expressing negative emotions have increased engagement.

3. **Community Dynamics and Real-Time Chat Analysis**
   Studies on gaming platforms like Discord and chat-heavy services like Slack highlight the importance of community-driven recommendations. However, Twitch's highly interactive and real-time nature presents unique challenges not full addressed by these works.

4. **Twitch-Specific Research**
   Previous works have investigated using metadata (e.g. viewer counts, categories, etc.) for recommendations. However, they largely overlook the role of chat environments. One study discussed how sentiment and chat tone could influence view retention, laying foundational insights for projects such as this one.

5. **Transformer Models for Text Embedding**
   Advances in transformer-based models like BERT and GPT have revolutionized text embeddings for NLP tasks. These models enable deep contextual understanding, critical for analyzing sentiment and extracting nuanced insights from chat data.

Method:

Dataset

The dataset for this project was derived from Twitch's chat data, collected through the Twitch API. The dataset consists of live chat logs from various streamers, representing a broad spectrum of streaming genres and community interactions. These chat logs were curated and preprocessed to ensure high relevance to the problem of recommending streamers based on chat dynamics.

Form of Data

The data is primarily in raw text form, structured in a JSON format. Each record contains chat messages sampled from live Twitch streams. For the model, the data was formatted as input-output pairs, where:

1. **Input:** A concatenation of a user prompt describing their preferences and a series of chat messages from a specific streamer.
2. **Output:** The streamer's name, serving as the target label for classification.

Features

Key features of the dataset include:

1. **User Prompt:** Descriptions provided by users that detail their preferences for a Twitch stream, such as "I enjoy streams with positive supportive communities" or "I prefer high-energy, fast-paced chat discussions."

2. **Chat Messages:** Textual context samples from a streamer's live chat logs, capturing the tone, sentiment, and engagement style of the community.
3. **Streamer Name (Target Label):** The label indicating which streamer's chat messages are being analyzed, serving as the classification target.

Additional preprocessing steps such as tokenization, sentiment analysis, and embedding generalization were applied to these features to make them suitable for input into a machine learning model.

Model

Initially, the project intended to use DistilGPT-2, a lightweight version of GPT-2, fine-tuned for causal language modeling, due to its natural language understanding capabilities. However, due to performance constraints and the classification nature of the task, we transitioned to using all-MiniLM-L6-v2, a more efficient and embedding-focused transformer model from the Sentence Transformers library.

This model excels at producing high-quality sentence embeddings suitable for semantic similarity and classification tasks, aligning better with our goal of matching user prompts to chat dynamics. Embeddings generated by all-MiniLM-L6-v2 were used as inputs to a downstream classifier that predicts the most appropriate streamer label.

Defining the Problem/Feature Space

The problem was framed as a supervised learning classification task. The input space consists of user prompts and chat messages combined into a unified sequence The feature space includes:

1. **Text Embeddings:** Derived from all-MiniLM-L6-v2 to capture semantic meaning in prompts and chat messages.
2. **Stream Context Representation:** Capturing the temporal and contextual flow of chats.

The output space is defined as a categorical variable, representing the streamers to recommend.

Implementation Steps

The implementation followed these steps:

1. **Data Collection:** Use the Twitch API to gather real-time chat logs from a diverse set of streamers.
2. **Data Preprocessing:** Clean the chat data by removing URLs, emojis, special characters and applying tokenizations. Normalize text to lowercase to ensure consistency.
3. **Dataset Creation:** Pair user-generated prompts with sampled chat logs to create labeled input-output data for training.
4. **Embedding Generation**: Use all-MiniLM-L6-v2 to encode both prompts and chat logs into fixed-size vectors.
5. **Model Training**: Train a classification model on top of the embeddings to predict the most suitable streamer.
6. **Evaluation**: Assess the model on validation data, monitoring metrics such as accuracy, loss, and qualitative alignment between recommendations and user preferences.

7. **Integration and Testing**: Deploy the model into a testing environment to simulate real-world recommendation scenarios and fine-tune thresholds for matching user inputs.

Evaluation

The evaluation strategy includes:

1. **Training Metrics:** Monitor training and validation loss to ensure effective learning. High validation accuracy indicates the model's ability to generalize to unseen data.
2. **Recommendation Quality:** Test the model by providing real user prompts and assessing how well the recommended streamers align with expected preferences.
3. **Fuzzy Matching:** Incorporate a secondary evaluation metric to handle cases where a perfect match is unavailable, measuring the cosine similarity between embeddings of user prompts and chat messages.
4. **Qualitative Feedback:** Validate with human testers to measure satisfaction with recommendations based on different user preferences.

Testing/Measuring of Success

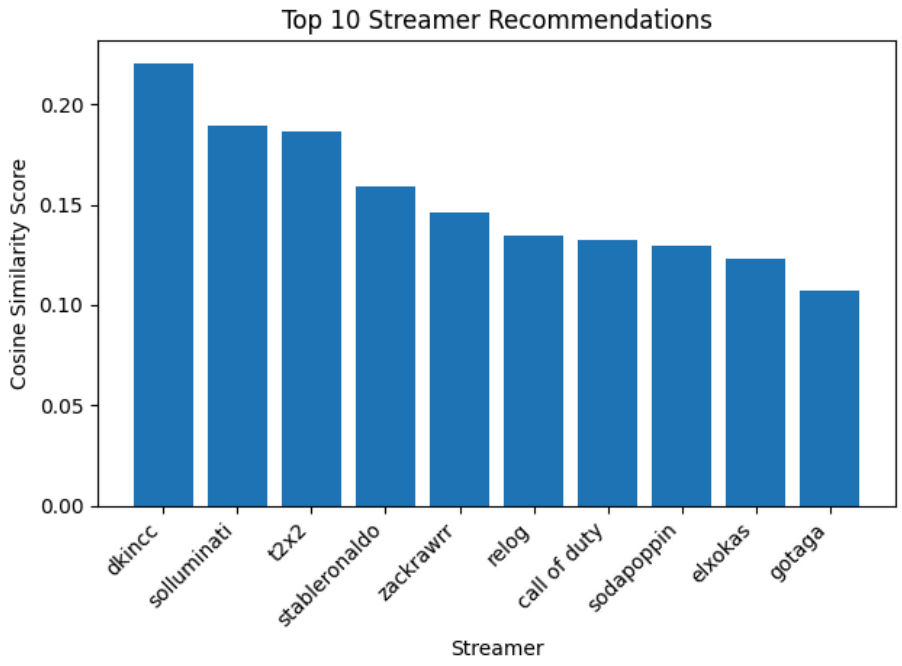
Success is defined by the following criteria:

1. **Accuracy of Recommendations:** The percentage of cases where the recommended streamer matches the user's preferences based on both chat content and contextual alignment.
2. **Engagement Rates:** Measure how well users engage with the recommended streams compared to baseline recommendations (e.g., those based solely on viewer count or category).
3. **Robustness Across Prompts:** Evaluate the model's consistency in handling diverse and ambiguous user prompts without significant degradation in recommendation quality.
4. **Scalability:** Test the system's performance on live data streams to ensure it can handle real-time recommendations without significant delays.

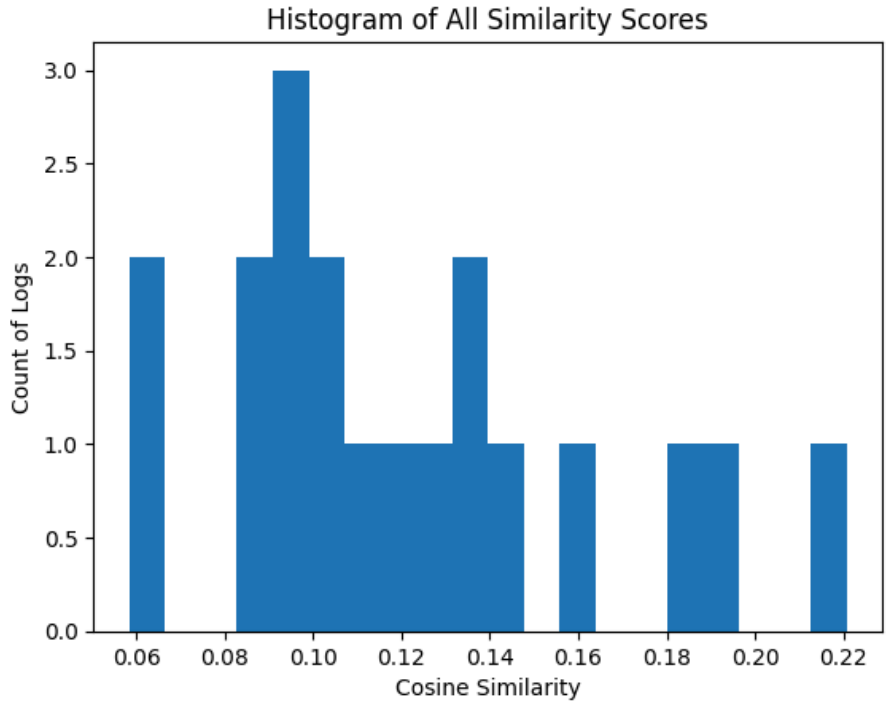
Results:

The results show proof that a chat based recommendation system is possible. Recommendations are lining up with the experiences the chats show, and this can further enhance Twitch's recommendation system as chats are a crucial part of the live streaming experience. Embedding based systems prove this further with similarities between the chat messages and user prompt showing higher recommendations while differences result in lower.

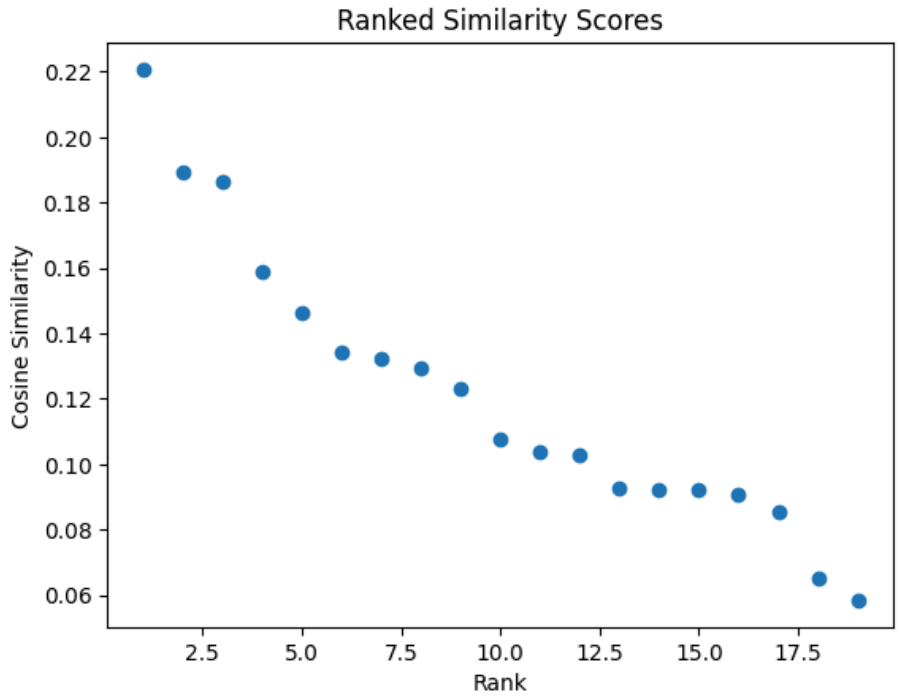
Visualizations



The graph above shows a trend with cosine similarities being led by dkincc for a prompt. This also shows a falloff, providing that some streamers have a better matching chat to the prompt than other streamers. This chart was used for general visualizations in cosine similarity to make sure it is working and to show the strength in recommending streamers.



This chart is comparing logs, which are the streamer's chats, with cosine similarity in a histogram to see the distribution of scores. You can see in this that there is a diverse amount of cosine similarities leaning towards the lesser side. This shows that prompts can pull out streamers chat's that relate to them and depending on how specific they are, they can influence how specific the streamer can get.



This chart is to compare the rank, which is the ranking of the recommendation compared to the cosine similarity. Similarly to the previous charts, it shows a falloff indicating that cosine similarity of the embeddings is functioning and a steeper falloff can bee seen with more specific prompts. It also interestingly shows a somewhat linear relationship as it goes down, as popular streamers can be similar in topics, chat's can be having a middle range which slowly doesn't relate to the prompt.

Further Analysis

Although the insights are strong, further improvements can be made. This system is a proof of concept proving that chat based recommendation systems with embeddings have merit. However, further work can be done in real time chat logs, longer chat logs, extracting sentiment over time, and even separating analysis of the chat messages between sentiment, emoji relationships, etc and seeing how an aggregated scoring system compares in similarity. These can increase the lower cosine similarity score compared to our model, and make recommendations stronger.

Discussion:

Outcome

The expected goal was to build a system that could analyze sentiment from real-time chat data of top Twitch streamers and recommend streamers with similar community vibes. I expected the model to accurately suggest streamers whose chat environments aligned with the user's input mood (such as cozy, chaotic, or hype) and overall, the results met these expectations. The model effectively categorized chat moods and returned appropriate streamer recommendations based on sentiment alignment.

Difference in Models

In our project, we presented two models: DistilGPT2 and all-MiniLM-L6-v2. While the all-MiniLM-L6-v2 model was able to generate streamer recommendations more effectively, we encountered several challenges with the DistilGPT2 approach. This model ultimately proved inefficient for a few key reasons. First, generalization was poor. DistilGPT2 wasn't powerful enough to learn meaningful patterns from the streamer chat logs. Rather than understanding context, it tended to memorize prompts, which often led to repeated or invalid recommendations like "blastberrd," a streamer that doesn't actually exist.

Another major limitation was its lack of embedding-based awareness. DistilGPT2 wasn't trained to interpret sentiment or mood properly, which meant it couldn't meaningfully differentiate between a "hype" chat and a "cozy" one. This created a clear semantic gap between the chat content and the intended mood. While the system was occasionally able to categorize moods and generate predictions, it struggled to consistently recommend streamers that matched the desired vibe. In several instances, the predicted mood and the suggested streamer were misaligned, revealing a disconnect between mood classification and final recommendation.

One potential reason for this is that certain streamer chats lacked meaningful sentiment content. For example, users might spam random letters, symbols, or slang words which dilutes the emotional signal in the data and reduces model accuracy. This type of noise made it difficult for a generative model like DistilGPT2 to extract consistent mood-related patterns, especially when those patterns are subtle or highly contextual.

This was solved through our final implementation of the all-MiniLM-L6-v2, which proved to be the most effective toward our project goal. Unlike DistilGPT2, which relied on text generation, all-MiniLM-L6-v2 is a sentence embedding model that transforms both user input and chat logs into dense vector representations. Using cosine similarity, we measured how semantically close each streamer's chat was to the user's desired vibe.

We also added a cleaning function into our new model implementation. By cleaning and normalizing the chat logs and comparing them directly to the input, the model was able to cut through noisy text patterns and focus on the underlying sentiment. This method significantly improved recommendation accuracy and made the system much more interpretable. As a result, we were able to use this model accurately to recommend streamers whose chats most closely aligned with the emotional tone or mood described by the user.

How did your actual results differ from your expected results?

Overall, we remained mostly true to our original proposal, but there were notable deviations. Initially, we planned to incorporate viewer engagement metrics, such as watch time, chat participation frequency, and following behavior, to personalize recommendations. However, due to Twitch's API limitations and data privacy restrictions, we were unable to access this level of user data through Twitch's public APIs.

As a result, we pivoted to a prompt-driven framework that relied solely on user mood inputs and chat sentiment analysis. While this approach allowed us to explore community-based mood matching, it limited the system's ability to make deeply personalized recommendations. If user-level engagement data had been available, the model could have been more precise by tailoring suggestions to individual users' viewing habits.