Song Recommendation and Sentiment Analysis Chatbot (Song Buddy) Report

By: Jichuan Li, Alex Wang, Aaron Chang

Github: https://github.com/alexwang9808/NLP-song-recommender-chatbot
Video: https://youtu.be/l88OfJhWgq0

Abstract

This project focuses on a chatbot that recommends songs and analyzes their sentiment using Natural Language Processing (NLP). Users can input a song, then the system retrieves lyrics and analyzes meaning using embeddings from BGE-M3. Finally, the project returns similar songs from a pre-built database. It also evaluates the emotional tone of the song using a RoBERTa-based model and displays both similarity and sentiment through charts.

Problem Description

We wanted to build a music recommendation system that reflected a song's content versus using a user's listening history or searching only by song genres. We created this model as an alternative time-efficient way to discover music based on lyrics and the emotional aspect of songs.

Approach

Our pipeline begins by taking a user's input and searching Genius for matching songs. Since the Genius search engine is not very forgiving or intuitive, we integrated an OpenAI-powered chatbot to interpret free-form user inputs and convert them into a clean query format that Genius can understand. We also added handling for common typos and variations like missing artist names or alternate phrasing (e.g., "song by artist").

Once search results are returned, we display them to the user and allow them to select the correct song. If only one result is found, we auto-select it. After a song is chosen, we scrape the lyrics using requests and pass them through the BGE-M3 model (via FlagEmbedding) to generate a vector representation of the lyrics.

We then use this vector to query our Pinecone vector database, which contains precomputed embeddings of thousands of songs. We retrieve the top 50 most semantically similar results and display the top 10 based on vector similarity. To help users visualize the results, we generate two charts: one showing similarity scores, and another showing sentiment classifications.

Next, we offer an additional filtering option based on sentiment similarity. Using a RoBERTa-based sentiment model, we classify each song into positive, neutral, or negative sentiment scores. We then compare the sentiment of the selected song with the retrieved matches and allow users to view songs with similar or opposite emotional tones.

Throughout the entire process, OpenAI's API plays a key role in making user input interpretation more flexible and enhancing the overall user experience with natural language interaction.

Data Description

The system uses a dataset of 57,650 songs, obtained from the Spotify Million Song Dataset from Kaggle., that are stored in a CSV file (spotify_data.csv). Song data like lyrics were preprocessed and embedded into Pinecone in advance. Each song entry includes title, artist, chunked lyrics, and sentiment scores.

During analysis, the chatbot fetches and processes only one song, but queries the full dataset to find matching songs.

Experiments and Error Analysis

Throughout the project, we experimented with multiple approaches at each stage of our pipeline — from handling user input to optimizing our recommendation engine.

Initially, we only supported basic input formats like "song" or "song artist", as the Genius API was limited. Any deviation from this structure would cause failures. To make input more natural, we added support for queries like "song by artist" using regex. We also explored using TextBlob for spell correction, but it often made things worse. After testing several libraries, we found the best results came from using ChatGPT to intelligently extract the song title and artist from user input, formatting it for compatibility with the Genius search engine.

We evaluated different embedding models, including sentence-transformers and BGE-M3. While both performed well, BGE-M3 provided more meaningful similarity scores between songs with semantically rich lyrics, so we selected it for our final implementation.

For sentiment analysis, we first tried VADER but found it produced inconsistent and overly extreme scores, particularly on longer songs. We then switched to a RoBERTa-based sentiment model, which offered more stable and reliable outputs across a wider range of lyrics.

After the initial version of our product, we noticed that some recommended songs didn’t feel meaningfully similar to the user’s input. To improve relevance, we developed a semantic chunking algorithm that split each song into meaningful sections before embedding and storing each chunk in a separate vector database.

At runtime, we would chunk the user's input lyrics, embed each chunk, and perform pairwise comparisons with the preprocessed chunks. Songs were then ranked based on the number of chunk-level matches.

However, this approach introduced significant latency: Our chunking algorithm grouped and embedded multiple overlapping lines to determine chunk boundaries, which was computationally expensive. The pairwise comparison step added further delay. In the end, we chose to drop chunking from the final system due to performance concerns and focus on faster, whole-lyrics-based retrieval instead.

For similarity matching, we noticed that the selected song sometimes showed up in the results, so we added checks to ensure the selected song didn't appear and used the next best song to replace it.

Originally, the bot returned all results at once, but we later added memory so it could pause and wait for a user to choose a song when there were multiple matches to the input.

Testing

We tested our project with 5 people outside of the class, by conducting user studies. For the subjects, we selected users that had various experience levels of using NLP (both a little and a lot). This was most commonly ChatGPT.

For each user, we briefed them with a set of instructions that guided them through our chatbot's features and required them to test finding similar songs and using sentiment analysis for multiple songs of their choosing. The instructions specified how to run the program, how to provide input, briefed them of the project's main goals, and outlined the user study's procedure and purpose.

Throughout each user test, the team was present to guide the user and take notes. The notes include information about the performance of the project, what the user reacted/commented during the testing, and the feedback from the interview at the end of the test. By collecting feedback and comments, we were able to expand on our future plans for the project and analyze crucial areas of the project.

Contributions

Aaron Chang
80 points - significant exploration beyond baseline
30 points - tried semantic chunking and added GPT api for better responses and to handle user inputs.
10 points - discussion of lessons and improvements
10 points - highlighted complexity - did error correction and reformatted the architecture
10 points - exceptional visualization/diagrams/repo

10 points - testing

Jichuan Li
80 points - significant exploration beyond baseline
30 points - tried semantic chunking and added GPT api for better responses and to handle user inputs.
10 points - discussion of lessons and improvements
10 points - highlighted complexity - optimized algorithm for song recommendation.
10 points - exceptional visualization/diagrams/repo
10 points - testing

Alex Wang
80 points - significant exploration beyond baseline
30 points - tried semantic chunking and added GPT api for better responses and to handle user inputs.
10 points - discussion of lessons and improvements
10 points - highlighted complexity - Gathered song database and created visualization
10 points - exceptional visualization/diagrams/repo
10 points - testing

Future Plans (Potential Improvements)

From both our own team discussions and from the user studies we conducted, there are several areas for improvement we'd like to add in the future. One idea is to include a way to preview a song's lyrics within the chatbot, to allow users to more easily recognize which song they were selecting. This was a concern among our testers, when they struggled because they did not remember the exact title or artist of the song they wanted. Another useful idea is to give the option to display generated charts directly in the chatbot (as another option for the user during a conversation with the chatbot) to allow users to more easily visualize the data. We also want to further optimize the project to improve the processing time of the program. During the user tests, the majority of the testers expressed that our chatbot returned results much more slowly compared to models like Open AI's ChatGPT.
Another big thing was chunking

Conclusion

This project successfully integrates natural language processing, semantic vector embeddings, sentiment analysis, and chatbot interaction into a cohesive music recommendation system. By leveraging BAAI's BGE-M3 model for lyric embeddings and CardiffNLP's RoBERTa model for sentiment classification, the system enables users to discover songs that are not only lyrically similar but also emotionally aligned or contrasting. The chatbot interface, powered by OpenAI's GPT-4o API, guides users through song selection, presents multiple search results, and offers sentiment-based recommendations in a conversational and user-friendly manner. Additionally, the inclusion of sentiment and similarity charts enhances the interpretability of results by

visualizing relationships between songs. In conclusion, this project provides an engaging and data-driven approach to music discovery, allowing users to explore songs through both meaning and mood.

References

- BGE-M3 FlagEmbedding: https://huggingface.co/BAAI/bge-m3
- CardiffNLP's Twitter RoBERTa model: https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment
- Genius API: https://docs.genius.com/
- Pinecone Vector DB: https://www.pinecone.io/
- Flask: https://flask.palletsprojects.com/
- Matplotlib: https://matplotlib.org/
- OpenAI API