# Music Recommendation System

Isha Sehrawat
2019046

Srishti Singh
2019114

Bhavesh Sood
2019355

## 1. MOTIVATION

With the advent of new technology and platforms, the music industry has experienced significant growth over the years. There has been a revolution in the way people access and consume music. The distribution of music digitally, with the rise of platforms like Spotify, Apple Music, etc., has made it easier to connect artists with listeners globally. More and more users are joining online streaming platforms. The revenue from music streaming has also been increasing every year since 2010. To meet the needs of these platforms, it's important to focus on improving user experience, music discovery, and data analysis. Platforms can improve the user experience by providing personalized recommendations, curated playlists, better search functionality, and user-friendly interfaces. Music streaming platforms can enhance music discovery by providing users with better ways to find new music they will enjoy. This could include features such as genre-based recommendations, new release notifications, and more personalized recommendations.

## 2. PROBLEM STATEMENT

With the vast amount of songs in the streaming services database, it's overwhelming for users to be up to date with the newest releases and the songs matching their mood. While many music recommendation approaches are available, they rarely take into account the user's emotions at the current time. They have underutilized the lyrics of the songs that the users have listened to in the past. We are planning to create a novel music recommendation system that takes emotions, user's playing history as well as preferences and the lyrics of the music to recommend the most relevant songs and achieve user satisfaction.

## 3. LITERATURE REVIEW

Music recommender systems based on dataset features have been the focus of significant research in recent years. A wide variety of features are used from the dataset ranging from audio features, user and artist metadata, and emotions associated with songs to train such systems.

Oord et al. [1] proposed a deep neural network model for music recommendation. They used raw audio data to extract more relevant information, like timbre, melody, rhythm, and harmony, in a compact and meaningful way. The focus was on getting latent characteristics from different song audio. To extract these representative features from songs, two approaches were used. The bag of words approach extracts MFCCs from audio and a deep learning approach that uses CNNs consisting of alternating feature extraction and pooling layers. The paper showed how audio data features though not sufficient alone, provide sensible recommendations.

Oramas et al. [2], in their research, presented a multimodal approach to recommend songs. One model is trained to learn feature embeddings for the artists based on their biographies. The second model learns the song feature characteristics from the audio data files. These two models are then combined to get the final track features based upon which suggestions are made. The qualitative analysis of the MSD showed improvement, which suggests a new approach to using artists' metadata.

Wang et al. [3] have pointed out in their recent work how the performance of the current recommender system is bounded due to limited contextual information. Thus, they proposed a new music recommender system that integrates content and contextual information. Their Hybrid music recommendation system utilizes graph embedding techniques to generate dense feature representations of songs in a low-dimensional space, allowing for more effective similarity comparisons between songs. Recommendations are then generated based on cosine similarity between users and song embeddings learned from graphs.

La Gatta et al. [4] propose using a hypergraph data model. The model consists of three parts: hypergraph, embedding generator, and recommendation generator. The hypergraph data model is used to represent complex interactions between users and songs, while the embedding generator encodes context-based information in low-dimensional vectors. The recommendation generator then computes the similarity score between the embeddings to rank the results, generating the final recommendation.

Wang Pengfei et al. [5] proposed yet another new approach that exploits the sequential nature of the user's music listening history. Apart from using user metadata, it makes use of transactions too.

X. Luo et al. [6] present a novel approach to collaborative filtering for recommender systems, which is based on non-negative matrix factorization (NMF). The authors argue that this approach has several advantages over traditional collaborative filtering methods, including improved accuracy, scalability, and interpretability. The paper involves decomposing the user-item rating matrix into two non-negative matrices - a user latent factor matrix and an item latent factor matrix. The latent factors(LFM) capture the underlying features that explain the observed ratings.

| **Model** | **Dataset Features** |
|---|---|
| Deep Content Based [1] | Audio |
| Deep Multimodel [2] | Audio, Artists Biography |
| CAME [3] | Content and Context |
| HEMR [4] | Song Metadata |
| HRM [5] | Sequence |
| Ours | Lyric features, Sentiment |

## 4. NOVELTY

Most of the papers we came across used features like audio, timestamp, user history, and lyrics content to perform recommendations. Lyrics, which have not been worked upon by almost all of the papers we found, are another crucial feature of songs that can be used to extract more information, like emotions associated with it. We utilized some new features to improve upon the recommendation quality. We made use of sentiment pertaining to the lyrics to get a more representative embedding for the lyrics.

## 5. DATASET

We used the Million Song Dataset, which consists of the metadata and audio features of around 1M songs. We used either a subset of data spawned from this dataset for training and testing our model. We used Song_data.csv, 10000.txt and Musixmatch dataset.

The Song_data.csv is a csv file that contains song_id, title, release(album name), artist_name and year(of release).

The 10000.txt contains the count play of 10,000 songs, including the user ID, song ID, and the number of times each song was played by the user.

The Musixmatch dataset consists of the song lyrics corresponding to the songs in the Million Song Dataset.

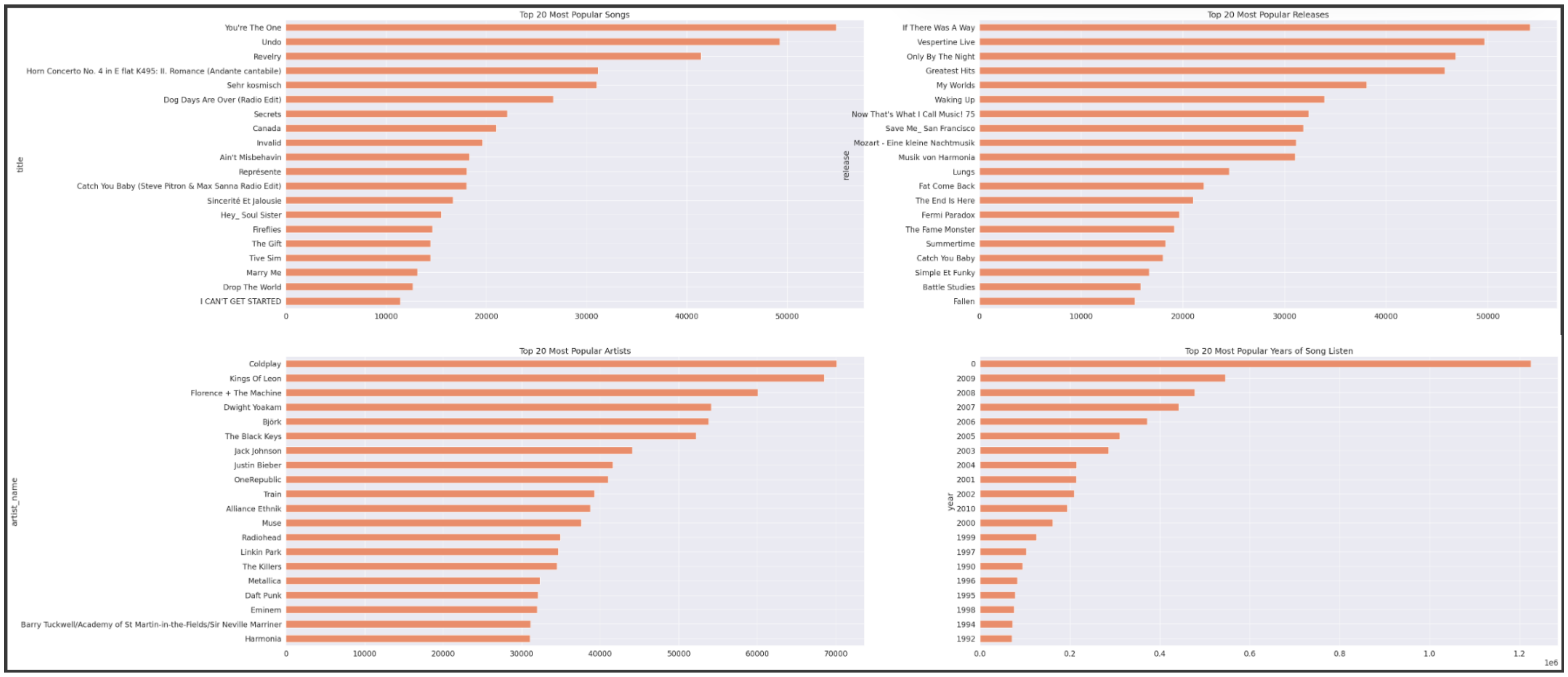


**Fig 1.** Bar Plots of the top 20 in each different feature of a song.

## 6. METHODOLOGY

### 6.1 Data Preprocessing

#### Merging Dataset

We merged song_data.csv and 10000.txt datasets by song ID for easy access. Merging will enable us to work with song names instead of song IDs. Our code groups the song_data.csv data by song ID and selects the maximum values for each group. This effectively removes any duplicate song entries. The resulting data frame is then merged with the 10000.txt data based on the song ID. After the merge, the play_count column is renamed to listen_count, and the song_id column is dropped.

#### Extracting Lyrics for more information

We merged the 10000.txt and Song_data.csv to form our main dataset. The merged dataset consisted of attributes like user ID, song ID, song listen count, title, release name, year of release and the artist name. We required lyrics corresponding to the songs present; however, we could not find a dataset meeting our requirements. So we used the musixmatch dataset available in the MSD. We used the mapping table available with the Musixmatch dataset to resolve the song ID present in the MSD with the song IDs present in the Musixmatch dataset. The lyrics in MSD are present in a bag of words format. We required lyrics in the format of strings, so we took the bag of words format for every lyric and converted it to string for only the top 5000 words in the entire dataset's vocabulary. We have also weighed every word that appears in the bag of word representation of a song's lyrics with the number of times it appears in that particular song.

To use the string lyrics in a more convenient way, we used the non-contextual model FastText to represent the strings in a way easier to understand for the machine learning model. Non-contextual models do not take into account the context associated with a sentence; hence they are well suited for a bag of word models, which is our case. We used the FastText model, which uses a word embedding technique that captures the meaning of a word by representing it as a vector of real numbers. These vectors are learned by training on a large corpus of text data and can be used to represent any word in the vocabulary. FastText also uses subword information to capture further the meaning of words, which allows it to handle unseen words and misspellings.

We hence used FastText to generate 300-length vector embeddings for the lyrics to every song in our dataset and used it for further analysis and experiments.

## 6.2 Exploratory Data Analysis (EDA)

After preprocessing, we performed exploratory data analysis (EDA) to understand the dataset's characteristics. It began by generating a summary of the 'listen_count' column, which shows that a song was played 2,213 times by a single user.

Next, we created four subplots to display the top 20 most popular songs, releases, artists, and years from user play data. It grouped the data by title, release, artist_name, and year, respectively, and then sorted them in descending order based on the number of times they were played. It then created horizontal bar charts for each of the top 20 lists and adds them to the subplots. Fig. 1 shows the bar plots of the top 20 in each different feature of a song(song title, album(release), artist and year).

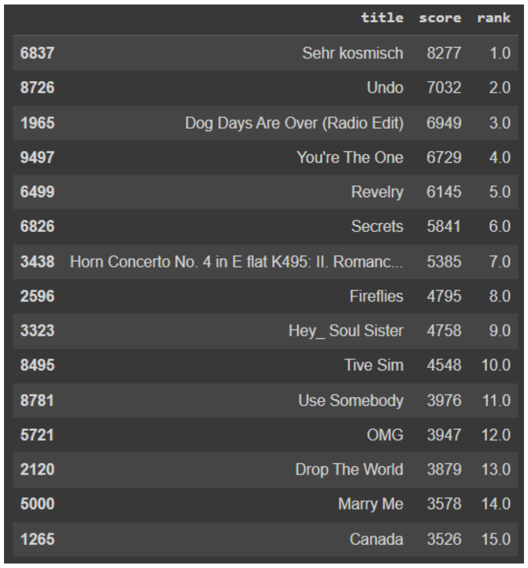
## 6.3 Different Recommendation Engines

Recommendation engines filter out products a user might be interested in based on their previous history. But if the previous history of a user is not available, then different methods are used.

As part of our investigation into music recommendation systems, we examined a few different approaches to generating recommendations in the absence of a user's listening history. Our aim was to establish a baseline for comparison with our proposed recommendation engine that we did as part of our project.

**Popularity-Based Recommendation Engine**

The first method we examined was a popularity-based recommendation engine, which recommends songs or artists based on their global popularity among all the users in our dataset.



| | title | score | rank |
|---|---|---|---|
| 6837 | Sehr kosmisch | 8277 | 1.0 |
| 8726 | Undo | 7032 | 2.0 |
| 1965 | Dog Days Are Over (Radio Edit) | 6949 | 3.0 |
| 9497 | You're The One | 6729 | 4.0 |
| 6499 | Revelry | 6145 | 5.0 |
| 6826 | Secrets | 5841 | 6.0 |
| 3438 | Horn Concerto No. 4 in E flat K495: II. Romanc... | 5385 | 7.0 |
| 2596 | Fireflies | 4795 | 8.0 |
| 3323 | Hey_ Soul Sister | 4758 | 9.0 |
| 8495 | Tive Sim | 4548 | 10.0 |
| 8781 | Use Somebody | 3976 | 11.0 |
| 5721 | OMG | 3947 | 12.0 |
| 2120 | Drop The World | 3879 | 13.0 |
| 5000 | Marry Me | 3578 | 14.0 |
| 1265 | Canada | 3526 | 15.0 |

**Fig 2.** Recommendations on the basis of popular songs

**Item Similarity-based Recommendation Engine**

The second method we evaluated was the item-based recommendation engine. This method focuses on the similarity between the user's list of songs (listening history, their playlists, etc.) and song data for other users that are present in the training data. This method calculates the similarity between the user list of songs and songs in our dataset using the Jaccard index based on common users of songs. Two songs can be considered similar to each other if a significant portion of the same users listens to both out of the total number of listeners.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

This method provides more personalized recommendations based on the user's listening history and preferences, thus improving the overall user experience and engagement. We compared the results by listening to the top song in the recommendation list and the song most listened to by the user.

**Latent Factor Model-based Recommendation (Matrix Factorisation)**

Matrix factorization is a powerful technique used in machine learning to discover latent features between two different kinds of entities. It has been successfully applied to a wide range of domains, including recommendation systems, image processing, and natural language processing. Matrix factorization can also be used to discover latent features of songs, such as beats, tempo, and other musical features. Once these features have been defined, they can be used to find matches for a user based on some similarity criteria, such as recommending songs that have similar beats or tempos to songs that the user has previously liked.

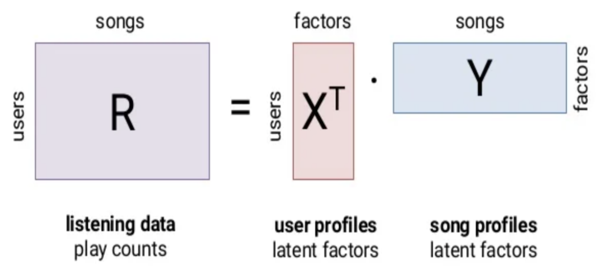


**Fig 3. Matrix Factorization:** Model listening data as a product of latent factors

The algorithm aims to minimize the reconstruction error between the observed listen counts and the predicted listen counts obtained by the product of the user and songs latent factor matrices.

Assuming the process helps us identify latent factors/features, meaning as K, our aim was to find two matrices X and Y such that their product (matrix multiplication) approximates R.

X = |U| x K matrix (A matrix with dimensions of num_users * factors)

Y = |P| x K matrix (A matrix with dimensions of factors * num_songs)

There are multiple algorithms available for determining factorization of any matrix. We used one of the simplest algorithms, which is the singular value decomposition or SVD.

Since we don't have a 'rating' in our database, we relied on play counts. We replaced the play count with a fractional play count to measure the 'likeness' of a song within the range of [0, 1].

### 6.4 Final Model

**Regression Models**

We used the lyrics incorporated dataset for training these set of models. Lyrics can be used to enhance the user experience of a recommendation system, as they can be displayed alongside the recommended songs to provide additional context and insight into the music. This can help users discover new artists and genres that they may not have otherwise considered.

The sentiment associated with songs also plays a key role in determining whether a user will like a particular song or not. Some users tend to like slow songs with melancholy lyrics, while some enjoy songs with joyful lyrics. We thus used the Vader Sentiment library to extract the sentiment associated with song lyrics. We have added a new attribute that represents the emotion of that song on a scale of -4 to 4, with -4 being the most negative and +4 being the most positive.

In this methodology, we used the fast text embeddings of the title and the weighted lyrics of each song to create a feature matrix. We then label-encoded the user IDs since they were textual data. This resulted in a dataframe of size 400k x 603.

We then used various regression models to predict the target variable (the last column of the dataframe). We started with linear regression and then tried XGBoost. We also created our own deep-learning model using Keras. This model had 602 input neurons, and we added five more layers, including batch normalization layers, to improve its performance.

Overall, this methodology involved preprocessing the data using text embeddings and label encoding, followed by trying out different regression models to find the best one for the task. The deep learning model was able to take advantage of the non-linear relationships in the data and was able to outperform the other models in some cases.

## 7. EVALUATION

### 7.1 Evaluation Metrics

To evaluate the performance of our system, we do a qualitative analysis of the initial dataset. We will manually look at some of the predictions made by our model for different users. The other training metric we have used is MSE and R^2 error. We predict the listen count for a particular song by a particular user, which helps us know how likely that user will enjoy a particular song.

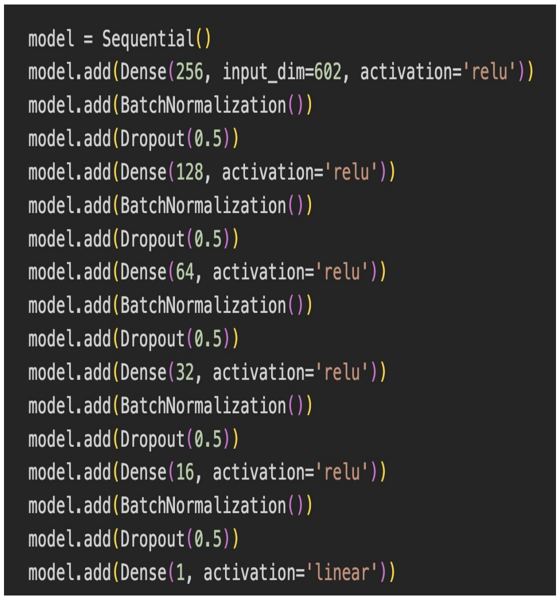
### 7.2 Evaluation Results and Analysis

| **Model** | **MSE** | **R^2 Error** |
|---|---|---|
| XG Boost | 36.61 | -0.01 |
| Linear Regression | 35.66 | 0.02 |
| Deep Learning based | 36.21 | 0.00 |

We observe that the simplest model Linear Regression gave a better MSE and R^2 , this must be due to the huge data, and also that a line fits well since most values lie between 1 to 10.

For the DL model:

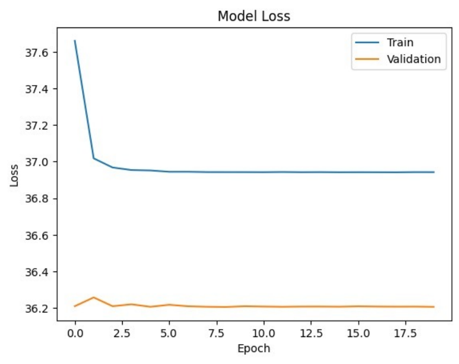
We made this model using a [Dense ->BatchNorm -> Dropout ] block strategy going down the dimensions.

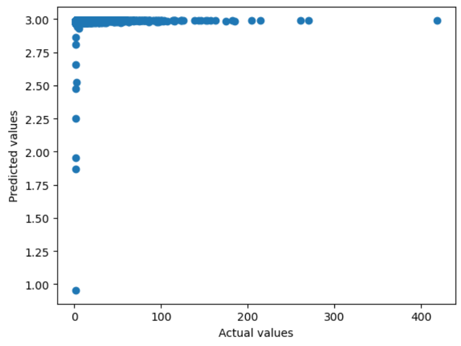


```
model = Sequential()
model.add(Dense(256, input_dim=602, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(1, activation='linear'))
```

The error however didn't decrease very much as compared to the ML models, because

- the model is underfit, as it was a large dataset it took a long time to train.
- we see that the model somehow made a max limit on prediction at 3.





## 8. CODE

The code of our Baseline Models and Final Model can be found here.

## 9. REFERENCES

[1] Van den Oord, A., Dieleman, S. and Schrauwen, B., 2013. Deep content-based music recommendation. *Advances in neural information processing systems*, *26*.
[2] Oramas, S., Nieto, O., Sordo, M. and Serra, X., 2017, August. A deep multimodal approach for cold-start music recommendation. In *Proceedings of the 2nd workshop on deep learning for recommender systems* (pp. 32-37).
[3] Wang, D., Zhang, X., Yu, D., Xu, G. and Deng, S., 2020. Came: Content-and context-aware music embedding for recommendation. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(3), pp.1375-1388.
[4] La Gatta, V., Moscato, V., Pennone, M., Postiglione, M., & Sperlí, G. (2022a). *Music Recommendation via Hypergraph Embedding. IEEE Transactions on Neural Networks and Learning Systems*, 1–13. https://doi.org/10.1109/TNNLS.2022.3146968
[5] Wang, P., Guo, J., Lan, Y., Xu, J., Wan, S. and Cheng, X., 2015, August. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 403-412).
[6]  X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrixfactorization-based approach to collaborative filtering for recommender systems," IEEE Trans. Ind. Informat., vol. 10, no. 2, pp. 1273–1284, May 2014.