CS 137 Project: Music Stream Count Prediction

Matthew Cook, Michael Loturco, and Abigail Stone

1 Introduction

Online music streaming services produce a weath of data about song popularity and trends in music over time. Access to this data gives us the opportunity to try to predict popularity and growth of songs. Predicting the stream count trends of a song can be advantageous for artists, producers, and streaming services. Artists may find this information useful for understanding which of their songs are gaining the most popularity, and streaming services may be able to better allocate resources to serve that song to end-users.

Existing work [2, 5] has used metadata and audio features to assign a generalized performance prediction, such as “low”, “medium” and “high” popularity or “hit song” and “not hit song”. The Spotify API [1] provides audio features (e.g. danciness and energy), lyrical features (e.g. valence and mood), and metadata for the track, including artist and album information. We are interested in forecasting the stream counts into future timesteps. Most of the existing literature focuses on computing a popularity score using metadata information, and we are interested in predicting stream count values over time.

Since the popularity of a track changes over time, we hypothesize that estimating “popularity” without considering the time dimension will produce poorer model performance than predicting popularity while considering time.

First, we use basic LSTM and GRU models that use time series to predict stream counts in future time steps. Then, we propose a joint architecture that considers both the encoded metadata from each track and the time series data of stream counts of those songs over time. We hypothesize that by considering stream counts over time, predictions for popularity will be more accurate.

We will evaluate our models using the standard metrics for regression problems: root mean squared error and mean absolute error. We will compare the results of our multiple-input model to the results of our own single-input LSTM and GRU models that mimic existing stream count prediction models in the literature.

2 Related Work

Martin-Gutierrez et. al [5] introduce the SpotGenTrack Popularity Dataset (SPD) and a deep learning architecture called HitMusicNet. SpotGenTrack combines metadata and audio data from Spotify and corresponding lyrical information from Genius and then extracts features from those sources. The extracted features are then compressed and encoded through a network MusicAENet. Finally a CNN (MusicPopNet) consumes the encoded and compressed feature vector to produce a classification of popularity. Similar previous models are noted to be binary classification, “hit” vs “not hit” whereas HitMusicNet is a ternary classifier separating “low”, “medium” and “high” classes.

Li et al [4] present LSTM Rolling Prediction Algorithm (LSTM-RPA), a popularity prediction algorithm based on a Recurrent Neural Network. LSTM-RPA breaks the long-term prediction into several short-term trend predictions. This work is an important example of using an RNN architecture to predict song popularity. Wang et al [7] also present an LSTM model for forecasting popularity.

Araujo et al [2] present a popularity prediction method using Support Vector Machines; this method predicts whether a song will appear on the Spotify Top 50 Global charts. This work has similar goals to our project and will serve as an important reference for evaluating our model's accuracy.

In their 2018 paper, Choi et al [3] compare audio pre-processing methods for deep neural networks. Their primary result demonstrates that many common preprocessing techniques are redundant; this conclusion will help us inform our pre-processing decisions for our project.

3 Methods

3.1 Dataset

Our primary dataset is a modified version of the daily top 200 songs by stream count in the Philippines, from January 1st 2017 to May 20th 2021 [6]. We reshaped this dataset to be of the form $\mathbb{R}^{d \times s}$, where d is the number of dates in the dataset and s is the number of songs.

One issue with this dataset is that it only represents the top 200 songs for a given day; as a result, songs occasionally drop off the charts for a few days and then return, leaving gaps in our time series. To solve this, we interpolated values for gaps of size exactly one (representing one day of off-chart stream counts) and filtered out songs containing zero values greater than a strict threshold. Due to the sparsity of our data, we experimented with different thresholds for including a song resulting, in a variety of different values for s.

To formulate our training data, we took sub-sequences of the overall date sequence for each song, with subsequences each of size `lookback` in our training dataset and the next timestep (of length one) set as the target value.

After trimming our data set in such a manner the shape is

$$X = [\text{Batch Size, Lookback, } s]$$

$$Y = [\text{Batch Size, 1, } s]$$

where X is our training data and Y is our target data.

In order to test our hypothesis, one of our architectures has multiple inputs. We augmented the existing dataset with additional features using the Spotify API. For each track in the existing dataset, we added album and artist information, as well as audio features from the Spotify API such as dancability, acousticness, energy, and loudness, among others. These audio features are all real values between 0 and 1. For each of the songs, we extracted a total of 13 metadata features and produced a tensor of shape [s, 13].

3.2 Model Architecture

The learning problem is a straightforward regression problem. Given the daily stream counts of songs over a period of time, we are looking to accurately predict the stream count of the next day. Given the time-oriented nature of the problem, our models consist of recurrent layers. We built basic models using both LSTMs and GRUs to produce baseline accuracy metrics to be used to evaluate our multiple-input models.

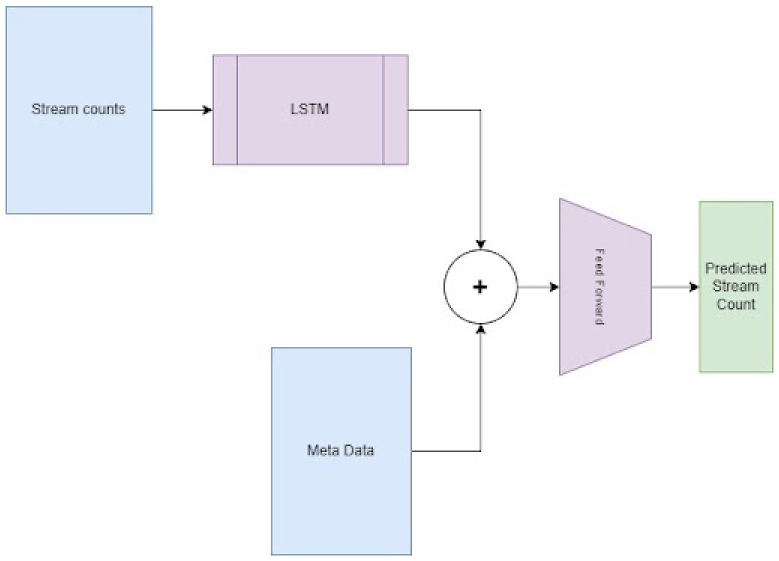

Figure 1: Multiple-Input model diagram

We propose an architecture for a multi-input model that combines stream-count data with the extracted metadata. We first run the stream count information through a recurrent neural network. Then we concatenate the output from this model with the m metadata features to create a feature vector of length $m + 1$. Then we feed this feature vector through a feed-forward neural network to output an improved stream count prediction.

3.3 Model Evaluation

We used Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error as our evaluation metrics.

Additionally, we visualize the predictions of our model by plotting the predictions alongside the original stream counts.

We set up two baseline models to evaluate the performance of our multi-input model: a basic LSTM implenetation and a basic GRU model to predict stream counts based solely on time series.

4 Experiments

4.1 GRU and LSTM

We performed a parameter search over two key dimensions. One parameter we varied was look back, the length of the time sequence provided which. Figure 2 (a) shows how MSE varies as look back changes in a GRU model. Figure 2 (b) shows the same variables searched over an LSTM model.

We note that there is not a clear performance trend based on lookback, but rather it appears to be periodic. We theorize this could potentially be due to stream count behavior appearing to be periodic as well, following somewhat consistent bumps of length 7 as the counts follow a pattern from week to week. Additionally, in the GRU model, as the song count increases, the performance becomes more stable and RMSE varies less over lookback.

We also note that for the LSTM model, that predictions of models trained on sequences on only a single song appear to outperform models trained on sequences from multiple songs. This was counter to an initial hypothesis of ours, where we expected a model to perform better by being able to learn from multiple songs. This potentially indicates that songs do not share enough sequence similarity to be beneficial to our model, and instead the variety of sequences introduced noise that caused the model to perform worse.

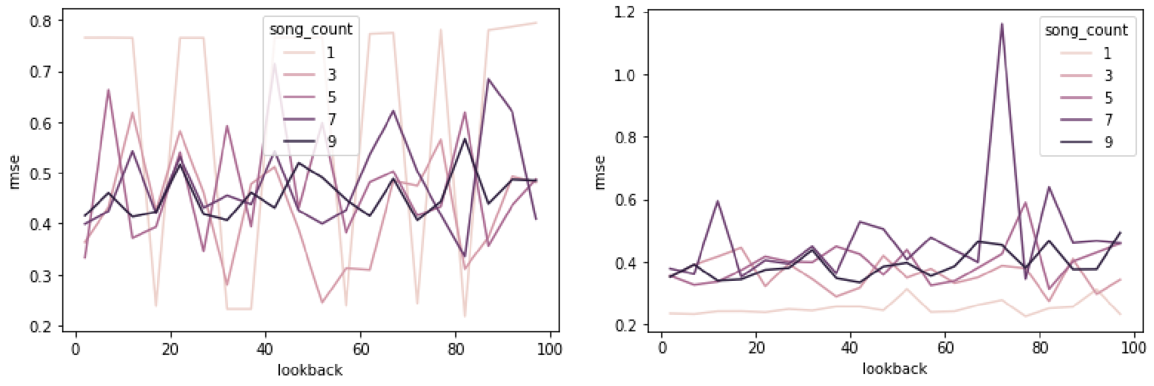

Figure 2: Lookback experiment results for (a) GRU (left) and (b) LSTM (right)

	LSTM	GRU	Multiple-Input
RMSE	3,460	3,959	7,742
MAE	2,369	2,777	7,159

Figure 3: Root Mean Squared Error and Mean Absolute Error for our models

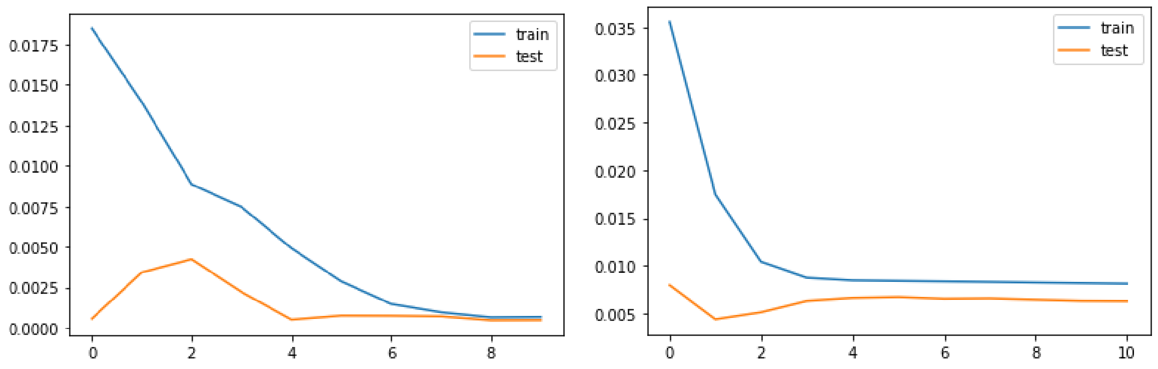

Figure 4: Training and Testing loss (MSE) for LSTM (left) and GRU (right)

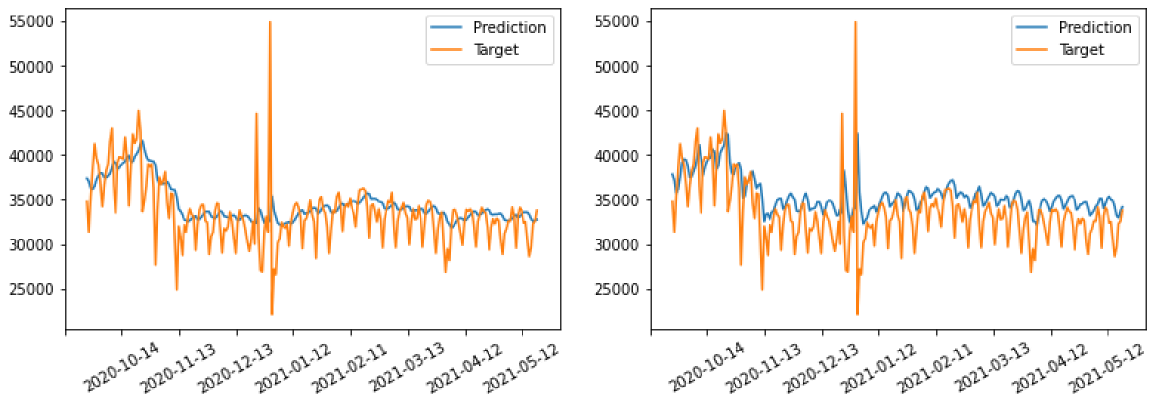

Figure 5: Predicted stream counts and target data for LSTM (left) and GRU (right)

4.2 Multiple-Input Models

Our multiple-input model is a Keras functional model that takes the stream count data as input to an LSTM, concatenates the result of the recurrent network to a vector of 13 metadata features, and feeds this result through a feed-forward network.

The results of this model (see Figure 3) demonstrate that the addition of metadata features did not improve the stream-count prediction.

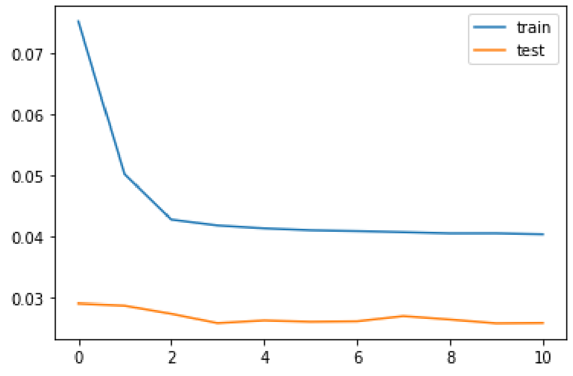

Figure 6: Training and Testing loss for mutliple-input model

5 Conclusion

Through these experiments, we found that our baseline models were able to predict stream counts for the next time steps. Given the parameters that we identified through a grid search, we found that our LSTM model out-performed our GRU model.

When compared to our baseline models, our multi-input model did not have better predictive performance. This contradicts our initial hypothesis that the inclusion of metadata with the time-series stream count data would improve the predictive performance. Instead, we found that our multi-input model predicted a constant value over all time steps. We hypothesize that this is due to a feature imbalance between our time-series data and metadata when combined into a feature vector. Since the metadata stays constant across all timesteps, the model prediction was less able to distinguish variations over time.

Future work could investigate rectifying this imbalance between time and metadata features. Additionally, other models could consider training time-series and metadata networks separately before combining features to feed into the final regression model. Additional experiments may also benefit from training on less sparse data. Despite careful data selection for our models, we were limited by the format of a Top 200 charts dataset, which inevitably had missing data points for many songs.

Additionally, training on songs that are of lower popularity than the top 200 songs might be able to better predict stream count across all levels of popularity; this is likely to be more beneficial for artists, producers, and streaming services.

6 Share of work

Overall, we believe that work was divided equitably.

- writing of the proposal (all)
- dataset creation and augmentation (Abigail 80%, Michael 20%)
- coding (Abigail 40%, Michael 40%, Matthew 20%)
- discussions (very evenly split)
- writing of the final report (Matthew 40%, Abigail 40%, Michael 20%)

References

[1] Spotify developer API.

[2] Carlos Soares Araujo, Marco Cristo, and Rafael Giusti. Predicting music popularity on streaming platforms. In *Anais do Simpósio Brasileiro de Computação Musical (SBCM 2019)*, pages 141–148. Sociedade Brasileira de Computação - SBC.

[3] Keunwoo Choi, Gyorgy Fazekas, Mark Sandler, and Kyunghyun Cho. A comparison of audio signal preprocessing methods for deep neural networks on music tagging. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1870–1874. IEEE.

[4] Kun Li, Meng Li, Yanling Li, and Min Lin. LSTM-RPA: A simple but effective long sequence prediction algorithm for music popularity prediction.

[5] David Martin-Gutierrez, Gustavo Hernandez Penaloza, Alberto Belmonte-Hernandez, and Federico Alvarez Garcia. A multimodal end-to-end deep learning architecture for music popularity prediction. 8:39361–39374.

[6] JC Peralta. Spotify daily top 200 tracks in the philippenes.

[7] Zhenye Wang, Chengxu Ye, and Wentao Wang. Music trend forecast based on LSTM. In *2019 4th International Conference on Computational Intelligence and Applications (ICCIA)*, pages 30–35. IEEE.