CS137 Project: Music Popularity 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.

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.

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.

We propose an architecture that considers both the encoded metadata from each track and the time series data of stream counts of those songs over time. Then we propose using a Recurrent Neural Network to take in time-labeled vectors including the encoded data and stream count and output a regression for stream counts for the next timestep. We hypothesize that by considering stream counts over time, predictions for popularity will be more accurate.

The thought behind such a process is that while a normal Recurrent network will predict the stream count with relatively good accuracy, combining that data with the encoded metadata will enable a feed forward network to use this information to refine the results of the previous network.

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. The overall architecture is shown in 1.

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.

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

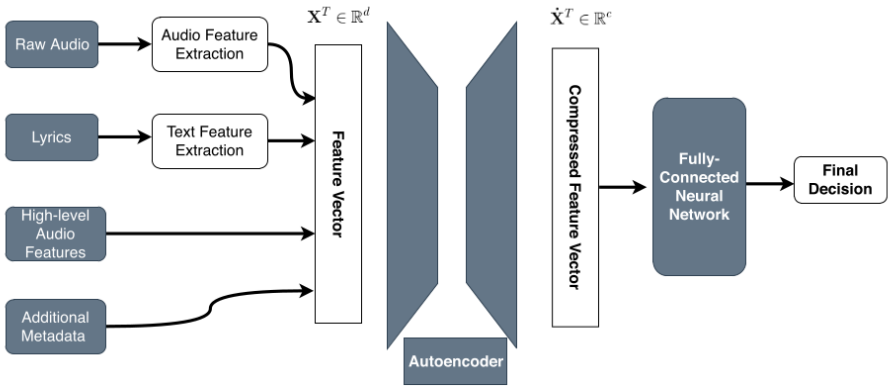

Figure 1: Model architecture presented by Martin-Gutierrez et al [5]. We intend to replace the fully-connected CNN with an RNN.

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 very small gaps (one or two missing values) and filtered out songs with longer sequences of zeros.

The format of the data at this point is tensor of the shape [Timesteps, Stream Count By Song]. The data was still incredibly sparse and in order to further avoid this problem and to form a training and testing set we scanned out data sets for songs which achieved a certain threshold entries over the course of the time.

As well to split out data into X values and Y, we took sub-sequences of various sizes, which we referred to as "lookback" with the next time step being the corresponding Y. After trimming our data set in such a manner the shape is

$$X = [\text{Batch Size, Lookback, Stream Count By Song}]$$

$$Y = [\text{Batch Size, 1, Stream Count By Song}]$$

Due to the sparsity of our data we experimented with different thresholds on whether or not to include a song resulting in a variety of different song sizes.

In order to test our hypothesis, one of our architectures has multiple inputs. The second data was formed by obtaining a variety of metadata for each song in the original data set

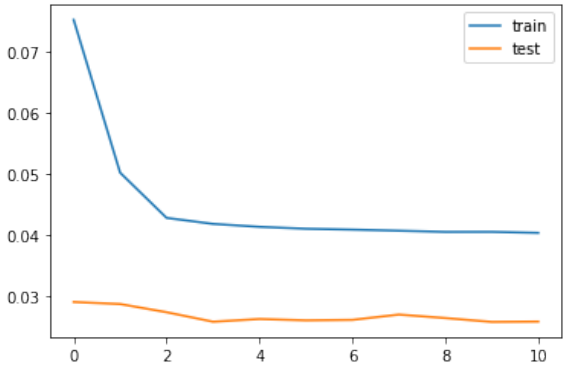

Figure 2: Training and Testing loss (MSE)

from the Spotify API. This metadata includes features such as dancibility or acousticness in the form of a real value from 0 to 1. For each of the songs 13 metadata features were extracted producing a tensor of shape [Song Count,13]

3.2 Model Architecture

The learning problem is a straightforward regression problem. Given the daily stream counts of songs of the course of a certain amount of days, can we accurately predict the stream count of the next day? Given the time-oriented nature of the problem ,the main aspect of our models consist of recurrent layers. Mostly we experimented with the use of LSTMs as our main layer, but GRUs were also used to a degree of success. We explored both basic models, which consisted of entirely just recurrent layers, and multiple input models which included the metadata. We combined these models such that the output of the recurrent layers were being concatenated to the metadata and then feed through a basic feed forward network.

This architecture turned out to be fruitless, which will be touched on more later, and as such we experienced more success with the basic recurrent models.

3.3 Model Evaluation

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

In order to visualize the success of our model, we plotted the predictions alongside the original amount of stream counts as shown below.

4 Experiments

Our first major attempt to improve performance was to train the multiple input model, which was previously described. This did not yield satisfactory results and was not properly training.

There are two intuition as to why the model did not work. The first being that since the recurrent layers are deeper in the model then we have no control over how they train. Therefore they may not be training to predict stream count and rather some other value. The other major issue is that since the RNN is only producing one value which is being

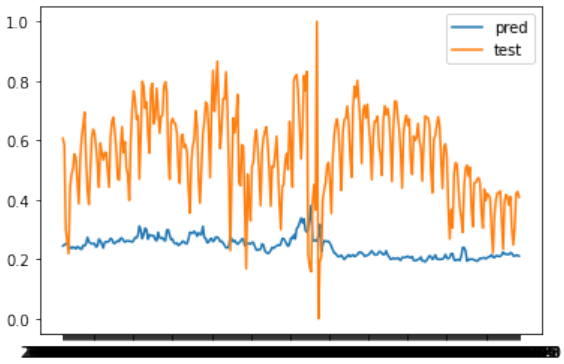

Figure 3: Predicted stream counts and validation data

concatenated onto the metadata for each song, than the model is going to naturally give more weight to the 13 metadata variables.

5 Conclusion

Something about successful vs unsuccessful models

While we were unsuccessful in our attempts to use the metadata of the song to improve performance that may mostly be due to implementation issues. As such it could certainly benefit from further attempts. One potential modification to the multi-input model is to train the recurrent layers as a separate model for a preliminary input. Then proceed to train a model which uses that output alongside the metadata to refine the predicted value. This avoids the issue of not knowing how the recurrent layer is training in the larger model, but still may present the problem that more weight is being given to the metadata than the predicted value. As well a larger and less sparse data set could be used a most likely would improve performance. While adding more data would improve the performance it could also lead to complications due to differing music tastes across different regions. By focusing on the Philippines data set

6 Share of work

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.