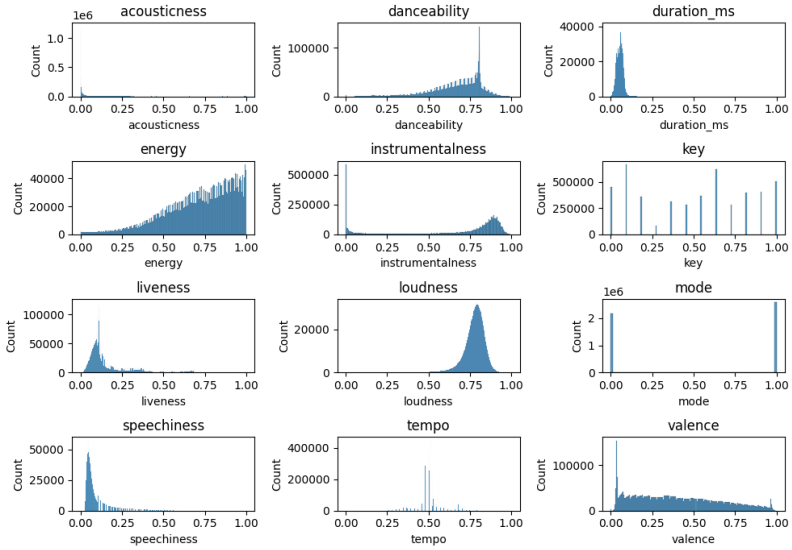
CS 474 Final Project: Spotify Playlist Track Predictor

Darren R. Skidmore, BYU Fall 2023 Semester

Abstract

In the Digital Era, music has become more popular and diversified than ever due to unprecedented access to hundreds of millions of tracks created by millions of artists around the world. The most popular and profitable service for music distribution is Spotify. For this project, I designed a neural network that could take a selection of ten Spotify tracks and predict the next one. Although there are selection algorithms which are much more diversified than mine, I was hoping that I could use mine to focus on not just very popular songs but search within a greater breadth of tracks.

Dataset



The dataset was quite complicated and involved some work in putting it together. My primary dataset was derived from two Kaggle datasets: a CSV file containing machine-calculated audio features (e.g., loudness, danceability, energy) for 4,687,104 tracks and a series of 1,000 JSON files containing 1,000 Spotify playlists each. I opted to use the ISRC (International Standard Recording Code) as the primary key because many Spotify track IDs may refer to a single ISRC and because the audio feature dataset primarily used ISRCs. I then randomly split the 1,000,000 playlists into a test/train split of 1,000 playlists each with cardinality greater than or equal to 10. Finally, I had to use the Spotify API to obtain missing audio features from tracks in the 2,000 playlists.

Problem

This is a regression problem. Instead of trying to predict an exact song, which would be infeasible and ineffective with a dataset of over four million tracks, I wanted to predict the audio features of the next track and find the closest match. The learning method was supervised, with the ground truth values being the preceding ten tracks in their respective playlist.

Some other approaches to this problem have been tackled in previous years, such as a Decision Tree Regressor and a seq2seq RNN [1][2]. However, in my research, I found that Spotify song *popularity* predictors were far more common, meaning that I was working on a relatively untouched problem.

Approach

I decided to use an RNN (Recurrent Neural Network) for this project with LSTM (Long-Short Term Memory.) RNNs are very useful for this task because they are designed to work with sequential data. Playlists can be represented well in a sequential form. LSTMs increase the ability of an RNN to

retain patterns from long ago (i.e., long-term memory) and are immune to the vanishing gradient problem. I didn't add anything else super fancy as I wrote the RNN from the ground up.

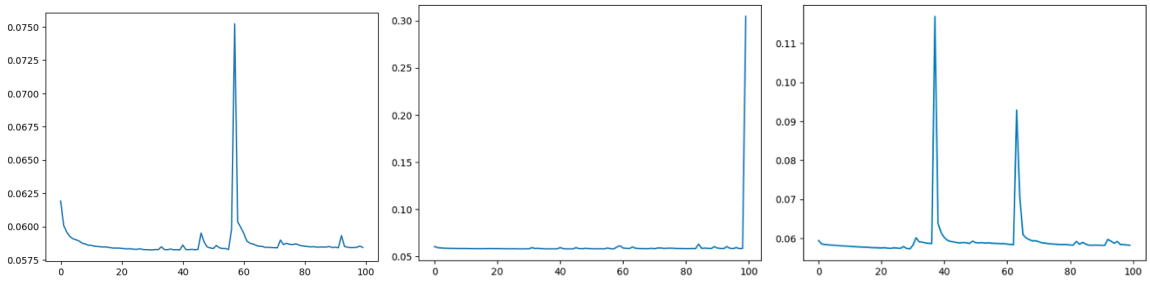
I tried several different combinations of hyperparameters to see what would give the best results. For example, I adjusted the number of layers in the model, the size of the hidden layers, adjusted the learning rate, and toggled between a ReLU and no ReLU. I used the Adam optimizer for the entire training period and MSELoss for criterion. I chose not to use any pretrained weights and trained my neural net from the ground up.

Analysis

Unfortunately, I was unable to get very good results from this project. While my loss was generally good, it started at that value (approximately 0.06) and never seemed to go down more than a few hundredths of a unit (peaking at 0.057 or so.) I objectively analyzed my performance using loss and subjectively analyzed it by seeing what the predicted song was for a list of ten previous tracks in one of my techno playlists. For trial #3, there was a weird bug where my test loss spiked right at the last epoch, so it would unfortunately be a fluke. Because the model is a regression classifier, I chose the song in the 4 million audio feature dataset with the smallest Euclidean distance to the output.

Trial#	Description	Parameters	Avg. Test Loss	Song Fit
1	30 epochs, 2 hidden layers, layer size 50, learning rate 0.01, no ReLU	9,479	0.058787	Great
2	100 epochs, 2 hidden layers, layer size 50, learning rate 0.01, no ReLU	9,479	0.058144	Average
3	100 epochs, 4 hidden layers, layer size 50, learning rate 0.01, no ReLU	9,479	0.653309	Very Poor
4	100 epochs, 4 hidden layers, layer size 50, learning rate 0.1, no ReLU	17,479	0.075675	Poor
5	100 epochs, 4 hidden layers, layer size 100, learning rate 0.01, no ReLU	54,929	0.058243	Good
6	100 epochs, 4 hidden layers, layer size 50, learning rate 0.01, ReLU	17,479	0.058653	Good

I believe this project has a lot of potential though. If I had more time, I would like to continue to try to expand the model a bit more and also try a GRU in-place of an LSTM. However, the predictive ability of a song on Spotify using mathematics alone may not be possible, and I would also want to consider other traits like artist/albums or song popularity.



Left: Trial #6 with loss generally stagnant except for a random spike
Center: Trial #3 with loss steady except for an extreme spike at the end which messed up the analysis
Right: Trial #2 with loss generally stagnant except for two random spikes

Citations

[1] https://medium.com/swlh/spotify-song-prediction-and-recommendation-system-b3bbc71398ad

[2] https://towardsdatascience.com/building-music-playlists-recommendation-system-564a3e63ef64

Time Log

Dec 10, 2023	1.5h	1.5 hours of data prep (downloading data from Kaggle and compiling it)
Dec 11, 2023	4h	3 hours of data prep (continuing to compile it) and 1 hour of model design (setting up initial structure)
Dec 12, 2023	7h	3 hours of data prep (getting missing data from API) and 4 hours of model design (initial training and first design)
Dec 19, 2023	1h	30 minutes of reading (getting some more ideas from articles), 30 minutes of model design (started the Euclidean similarity factor)
Dec 20, 2023	6h	6 hours of model design (completed Euclidean similarity factor and custom playlist predictor, started modifying hyperparameters)
Dec 21, 2023	6h	6 hours of model design (finished hyperparameter modification/testing)

Summary: 7.5 hours of data prep, 30 minutes of reading, 17.5 hours of model design. **25 hours total.**