CS230

Trendy Tunes: Predicting Popularity of Top 200 Spotify Songs

Dhruv Medarametla
Department of Computer Science
Stanford University
dhruvm2@stanford.edu

Abstract

Spotify is a music streaming service that provides the ability to stream songs without purchase. This project aims to create a model that predicts the number of streams for a given song, given information about the performance of the song on Spotify for the last 30 days. Using an online Kaggle dataset, we create features based around streaming and ranking data for songs in the US. We first use a linear regression model, then a neural network, and finally use an LSTM-based RNN. This last model is found to perform the best, with an average error of under 4%. Finally, we discuss future work, involving gathering more data and extra features.

1 Introduction

In the past decade, the way that people interact with music and audio has drastically changed. With the advent of the internet has come many different ways to experience music, from YouTube to subscription services to streaming devices. With over 30 million songs available, and billions of hours of music listened to, Spotify is a huge player in the music streaming industry and is poised to grow even bigger. With all this usage comes data, especially about the most frequently streamed songs. This data is extremely useful when it comes to predicting future Spotify song trends and can reveal insight as to what makes popular songs popular.

In order to do this, I created multiple algorithms to help predict the number of streams for a song on a given day based off of information for the past 30 days. More specifically, the input to my algorithms was the rank and stream data for the past 30 days within the United States, as well as the day of the week. I then used this data to predict the number of streams on the current day. I used a Linear Regression, a Neural Network, and a Recurrent Neural Network as the three different algorithms to create models off of. As expected, the RNN did the best as it was able to understand the time-series nature of the data, while the Linear Regression did the worst.

2 Related work

Surprisingly, not much work has been done in this area. While there are general projects about trends in music, or the ways in which Spotify or music streaming is impacting the overall industry, there are barely any specific studies or papers regarding the prediction of song popularity.

The one study that is closest to this research is the work done by Lee et. al [3]. In this paper, the authors focus on using the musical complexity of a song, as well as its early-stage success, to predict how successful a song will become. They draw their data from the Billboard Rock Songs Chart rather than Spotify, but the overall trends should still stand. In particular, they found that combining

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

the complexity of the song with its early popularity levels was an effective set of features to predict overall popularity and growth/declination rate. This differs from my research in that they also used the song itself as input, and only studied the early levels of popularity; however, it is definitely very similar, and relies on the same ideas and models.

Krietz et. al [2] studied the ways in which individuals actually interact with the Spotify network. In particular, they looked at how the peer-to-peer network affected the access patterns, as well as the latency period with the servers. While this is certainly interesting in its own right and helps explain why Spotify is as popular as it is today, it doesn't have much to do with predicting song popularity.

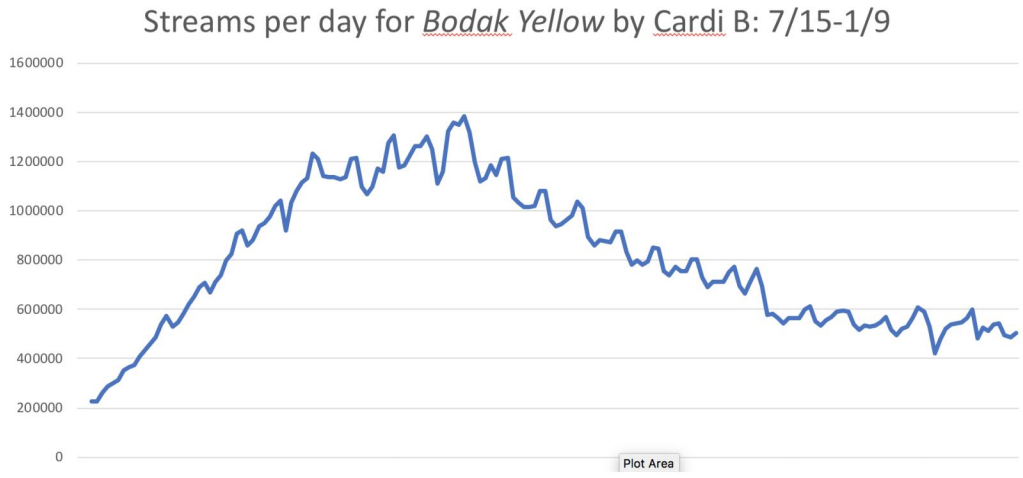
Shen et. al [5] conducted some particularly interesting research where they modeled the popularity dynamics of individual things within a larger system through the use of reinforced Poisson processes. Indeed, this research was a lot more general than just Spotify songs; it could be applied to YouTube videos, Twitter posts, or news articles. The approach used was particularly smart; by modeling the system through a Poisson process, the researchers were able to create a stronger and more explainable model, despite its generality. The reinforced Poisson process proved to be particularly effective, suggesting that approaching these problems from a mathematical perspective may prove more fruitful than simply applying deep learning algorithms.

3 Dataset and Features

I collected data from a Kaggle dataset containing daily song rankings and number of daily streams for Top 200 Spotify songs in several regions [1]. The initial dataset given had data on the top 200 songs each day from Jan. 1, 2017 to Jan 11, 2018. Each data point contained the date, a region, the name of the song, the title of the artist, the URL to the song on Spotify, the rank of the song in the particular region, and the number of streams of the song in that particular region on that given date.

My first step was to restrict all data to only from the United States. I did so because that was the region I found interesting, and because otherwise there would have been regional overlap in the data.

The next step was to take all of this data and make it into a matrix that I could work with. After research, I ended up using Excel's VLOOKUP function to find the number of streams, as well as the rank on the US charts, for each pair of song and date. Thus, I was able to transform this data into data for each song, though the data was only available for days that the song had been in the Top 200. Shown below is some of this data for *Bodak Yellow*, one of the songs that had months of stream data.



The next step was to create the features. Using this data, I created 1500 data points; each point had features involving the last 30 days of stream and rank data, as well as the day of the week. More specifically, the Linear Regression model and the Neural Net model had 124 features, involving the last 30 days of stream data, the last 30 days of rank data, the last 29 days of the sign of the change of the streams, the last 29 days of the sign of the change of the rank, and 6 variables to represent the day of the week. The RNN had 30 time steps, with 10 features per day–one for the stream data, one for the rank, one for the sign of the change of the stream, one for the sign of the change of the rank, and six for the day of the week. The output was the number of streams in the following day.

We used 80% of these examples for the training set, and 20% for the test set. As a result, we had roughly 1200 training examples, and 300 test examples.

4 Methods

Before we started to implement any of the algorithms, it was important to define the appropriate loss function. I wanted to make sure that our algorithm didn't prioritize examples with a higher number of streams, but would instead treat each data point equally. Furthermore, I cared about the percent error on the prediction, rather than the actual number. This led me to implement the following loss function, where $y^{(i)}$ is the actual number of streams and $\hat{y}^{(i)}$ is the predicted number of streams:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{m}\sum_{i}(\frac{\hat{y}^{(i)}}{y^{(i)}} - 1)^2.$$

This formula helped made sure that each data point was treated equally by looking at the ratio of the predicted value to the actual value. While it is important to note that a loss function like this may not have been appropriate for when the error was significantly large, such as 30%, because in such scenarios a prediction of 70% of the actual value would be worse than a prediction of 130%, our algorithms were off by 6-7% at the most, so this loss function worked perfectly fine.

The first algorithm I used was a standard Linear Regression. Because the input had 124 features, the formula for the prediction would look like this, so we had 125 parameters:

$$\hat{y}^{(i)} = w_0 + \sum_{j=1}^{124} w_j x_j^{(i)}$$

The second algorithm was a Neural Network. Because there were 124 features to start off with, I chose to use 15 neurons in the hidden layer, as I felt that it was a good medium to allow for just 1 neuron in the last layer. The equations for a forward propagation under a neural network are the following, where W_1 is a 15×124 matrix, b_1 is 15×1, W_2 is 1×15, and b_2 is 1×1:

$$\begin{gathered} Z_1 = W_1 x^{(i)} + b_1 \\ A_1 = \text{ReLU}(Z_1) \\ Z_2 = W_2 A_1 + b_2 \\ \hat{y}^{(i)} = A_2 = \text{ReLU}(Z_2) \end{gathered}$$

Therefore, this model had a total of 1891 parameters.

Finally, the last algorithm I chose to use was a Recurrent Neural Network. Specifically, I used an RNN with Long short-term memory (LSTM) structure. I had 30 days of data in the network before it made a prediction, and each day carried with it 10 features. As a result, I decided to have the hidden activations have dimension 10 as well. The equations for an LSTM RNN are the following:

$$\begin{gathered} h_0 = 0 \\ z_t = \sigma(W_z[h_{t-1}, x_t^{(i)}]) \\ r_t = \sigma(W_r[h_{t-1}, x_t^{(i)}]) \\ \tilde{h}_t = \tanh(W[r_t * h_{t-1}, x_t^{(i)}]) \\ h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \\ \hat{y}^{(i)} = W' h_{30} + b' \end{gathered}$$

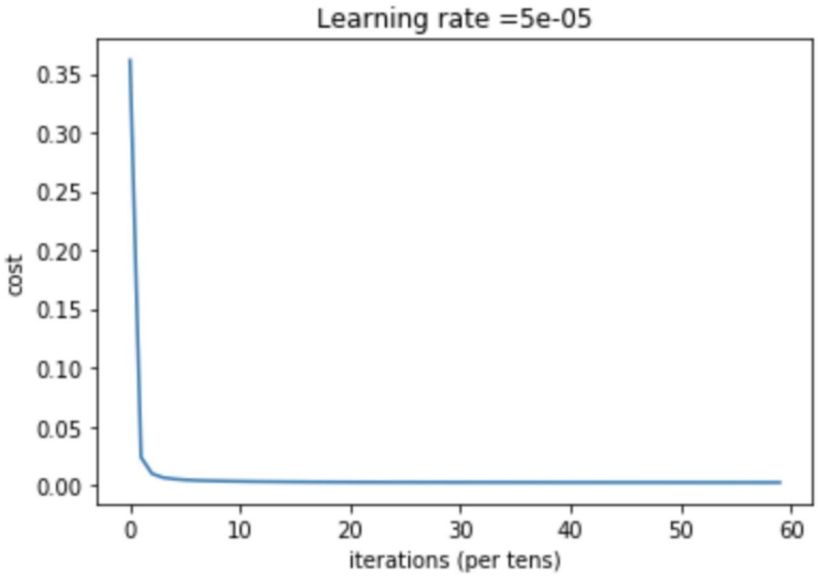
Because W_z, W_r, and W all take a vector of size $10 + 10 = 20$ and transform it into a vector of size 10, each is a 10×20 matrix. Additionally, note that W' is a 1×10 matrix, and b' is 1×1. Therefore, I had a total of 611 parameters for the RNN.

All three of these models were able to successfully train on our 1200 training examples, and performed about as expected relative to each other.

5 Experiments/Results/Discussion

One of the biggest issues that I ran into when training these models was simply selecting the hyperparameters. The learning rate that applied to the Linear Regression would not apply to the Neural Network, and both of those learning rates weren't at all close to the appropriate learning rate for the RNN. In particular, the Linear Regression model had a learning rate of $\alpha = 0.0002$, while the Neural Network model had a learning rate of $\alpha = 0.00005$. I hypothesize that the Neural Network a lower learning rate due to the fact that there were much more parameters, each of which had a significant impact on the outcome. Meanwhile, the RNN had a learning rate of $\alpha = 200$. I chose this much larger learning rate for the RNN because previously, the model was showing no change whatsoever on different iterations; it turned out that to make any progress, the gradient had to be multiplied by a much larger constant.

I kept most other parameters constant throughout the running of the model. One particularly important one was the mini-batch size. I chose to keep the mini-batch size at 50 for all three models, as a much smaller size led to a slower model that ultimately didn't perform much better, while a larger batch size meant that there would only be about 10 batches for the entire training set. 50 was a healthy balance.



Shown above is a graph of the cost of the Neural Network model after a given number of iterations. As seen in the graph, the cost barely changes after 100 epochs; the difference between the cost after 150 epochs and the cost after 300 epochs differ by less than 10%. This suggests that for the future, the number of epochs could potentially be sacrificed for a stronger model or a more intensive dataset.

As mentioned earlier, the primary metric was the average squared difference between the ratio of the predicted to the actual and 1. Thus, a cost of 0.04 may indicate that the average variance of the predictions from the actual value was roughly 20%. What follows is a table of our results, on both the training and test set.

Table 1: Model Results: Loss on Different Sets

Model	Train Performance	Test Performance
Linear Regression	0.002178	0.002156
Neural Network	0.001944	0.002105
RNN	0.001212	0.001549

As we see, based on the fact that the RNN has a loss of about 0.0015 on the test set, the average prediction is off by a little less than 4%. For reference, in my attempts to guess the number of streams, I was off by about 5% on average, which would have led to a cost of about 0.0025. Therefore, the models all performed slightly better than I did.

It's important to note that there is some variance, particularly in the last two models; the train set does significantly better than the test. However, when I tried to increase the regularization parameter to combat this, both the training loss and the testing loss suffered, suggesting that this wasn't the appropriate way to remove this variance.

I think the biggest reason for this variance was the size of the dataset. Because I only had 1200 examples to train on, the model was not able to create a set of parameters optimal for the entire population, but rather optimized specifically for the dataset. Both the RNN and the Neural Network had over 600 parameters; with such a large parameter to data ratio, there is bound to be overfitting. Unfortunately, because it was impossible to get any more data–finding Spotify stream data for songs that are not in the top 200 is a much more difficult task–I don't see any easy way to mitigate this in the future.

6 Conclusion/Future Work

To reiterate, the purpose of this project was to find a way to predict the number of Spotify streams for a given song based on data about the song's performance on Spotify for the past 30 days, as well as the day of the week. The raw data was found on Kaggle, and was manipulated into a form that resulted in 1500 data points.

I utilized 3 different models to solve this problem. I started off with a simple linear regression, then moved on to a neural network, before ending with a recurrent neural network with a long short-term memory structure. I defined the loss as a variation on the typical linear regression loss, in a manner that did not bias the predictions toward larger data points.

As predicted, each model did better than the previous one. In particular, the RNN did the best, with an average error of less than 4%. However, its performance was not much better than the linear regression, with the test cost of the RNN being about 70% of the test cost for the linear regression.

The RNN did better most likely because it was able to take advantage of the time-series nature of the data. Because the inputs were all put in as separate variables into the linear regression and the neural network, the models may not have been able to take as much advantage of this temporal data.

However, the RNN and the neural network exhibited significant variance, with the performance on the train set being much better than the performance on the test set. This is most likely due to the fact that there wasn't much data to train on; because the number of parameters was on the order of the amount of data, the RNN and the neural network most likely overfit to the training set, resulting in poorer performance on the test set.

I think that if there was more time and resources, the first step I would take to improving this model would be finding a way to get more data. Powerful models aren't of any use if there isn't much data to train them on. Maybe this would mean looking at Spotify regions outside the United States, or even looking at Billboard or iTunes data. This could also mean data augmentation; seeing how one song performs in different regions of the world, and including all of those data points separately could generate more data.

The second step would be to create an input that took the song itself into account. None of these models had any features based off of the actual song itself; it was only the performance of the song on the charts. I think that through further analysis of the song, similar to the work done by [3], it would be possible to factor in ideas like song complexity, catchiness, beat, and more, all of which would lead to more accurate predictions of streams.

Another interesting application would be to try this model out on YouTube music videos, and see how it fares at predicting the increase in views per day.

7 Contributions

I worked on my own for this project, so all the work that you see was completed by me.

All code for this project can be found in the Github repository with handle dhruvm2.

References

This section should include citations for: (1) Any papers mentioned in the related work section. (2) Papers describing algorithms that you used which were not covered in class. (3) Code or libraries you downloaded and used. This includes libraries such as scikit-learn, Tensorflow, Pytorch, Keras etc. Acceptable formats include: MLA, APA, IEEE. If you do not use one of these formats, each reference entry must include the following (preferably in this order): author(s), title, conference/journal, publisher, year. If you are using TeX, you can use any bibliography format which includes the items mentioned above. We are excluding the references section from the page limit to encourage students to perform a thorough literature review/related work section without being space-penalized if they include more references. Any choice of citation style is acceptable as long as you are consistent.

[1] Eduardo. Spotify's Worldwide Daily Song Ranking | Kaggle, 12 Jan. 2018, www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking/data.

[2] Kreitz, Gunnar, and Fredrik Niemela. "Spotify–large scale, low latency, P2P music-on-demand streaming." Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on. IEEE, 2010.

[3] Lee, Junghyuk, and Jong-Seok Lee. "Predicting music popularity patterns based on musical complexity and early stage popularity." Proceedings of the Third Edition Workshop on Speech, Language & Audio in Multimedia. ACM, 2015.

[4] "Sequence Models." Coursera, www.coursera.org/learn/nlp-sequence-models/.

[5] Shen, Hua-Wei, et al. "Modeling and Predicting Popularity Dynamics via Reinforced Poisson Processes." AAAI. Vol. 14. 2014.

[6] "TensorFlow." TensorFlow, www.tensorflow.org/.