Predicting Popularity of Songs

Anna Feng, Steve Feng, Linhao Chen, Alex Chiu,
Alejandro Estrada-Esparza, Emmy Huynh, Ryan Kim,
Joyce Lu, Jose Ordaz, Suryakiran Santhosh, Zhetan Zhang

December 5, 2021

University of California, Davis
ECS 171: Machine Learning
Fall 2021

GitHub Link

1 Introduction

After extensive research and deliberation, our group chose to base our project on a Spotify popularity dataset. From the overall popularity score to unique specifications like loudness and valence, the dataset contained a variety of information on music tracks. We chose this dataset because the specifications of the songs are not something people often think about when listening to music. We were curious how such parameters impact the overall popularity of a song or if there is correlation at all. While some specifications are difficult to quantify and visualize, the dataset provides quantitative data, which we used in our model to correlate the songs' specification measurements to its popularity rating. We began by analyzing the dataset to determine which parameters to focus on and which were unnecessary for our model. In order to decide which parameters to omit, we created heatmaps and pairplots with the dataset to better understand our dataset and correlation values between the different parameters. Once data cleaning was complete, we could hone in on and test the parameters we chose.

Based on the characteristics of our dataset and the goals we plan to achieve, we need to choose a suitable model and find the optimal combinations of hyperparameters. The models we used are Deep Neural Networks and Random Forest. To achieve a high accuracy result, we need to use methods such as k-fold Cross Validation to prevent overfitting and methods such as Grid Search to find suitable hyperparameters. After completing the model, we can help users predict the popularity of their favorite songs through a simple user interface that uses the Spotify API. After entering a valid song name, we can get attributes of the songs from the Spotify dataset and return a predicted score to the user.

2 Literature Review

Machine Learning was recently introduced into the music industry by Spotify. During the 2010s, Spotify was focused on making music quantifiable to allow researchers to conduct computational studies on music. Spotify researchers created thirteen attributes for each song, which have allowed researchers to predict both the relative and historical popularity of songs. Spotify creates the datasets by queuing the most popular songs synchronously. This process works by calculating the attributes of each popular song and then adding it to the dataset. Once the current popular songs have been added, the algorithm continues the process on historically popular songs that are not trending at this specific time, thus precipitating the steady creation of datasets.

Philip Peker, a producer and Music Industry Data Scientist, researches the computational modeling of music. Peker performed experiments with various machine learning models and feature engineering hypotheses to determine which machine learning algorithms have the highest accuracy in predicting the popularity of a certain song. The experiment concluded that Kth Nearest Neighbor Classification has the highest accuracy. This model classifies a song into one of three categories: "low", "medium", and "high". Peker acknowledges that a discrete numerical value in the range of 1 to 100 would provide for a more practical implementation of popularity predictions. Furthermore, Peker believed that there was extreme overfitting of the model but he did not run any experiments to verify his belief. We ran Peker's model on the training and testing sets and compared the accuracy for each one. We found that the model had high accuracy on predictions from the training set and low accuracy on the testing set. Thus, Peker's assumption of overfitting was accurate. We will be using a different model from the one Peker used so that we can reduce overfitting and try to find a different result from Peker's experiment.

3 Data set Description and Data Analysis

Our dataset initially had 122,663 observations and 16 attributes, but the 'popularity' distribution was not balanced. Since they were unrelated to popularity, we removed all categorical attributes, which includes artist_name, track_name, track_id, genre, mode, tempo, and duration_ms. The numerical attributes, which measure from 0 to 1, were used to measure the various aspects of a track like danceability, the likelihood of a person most likely to dance to the song. The next few attributes focus on the performance of the song, such as instrumentalness (predicts whether a track contains no vocals), liveness (the likelihood the song was performed live), and loudness (in decibels (dB), ranging from -60 to 0). We also measure speechiness, which illustrates how speech-like the recording is, similarly to a podcast. The last measured attributes are valence, the likelihood of how happy a track sounds, and the year, which contains the timestamp of our data. All these attributes are used to predict the popularity of the track with a range from 0 to 100.

Prediction Set	1	2	3	4	5
Popularity	1-20	21-40	41-60	61-80	81-100
Data Points	13256	59120	40131	9622	534

3.1 Heat Map

The heatmap shows the covariance of two variables in our Spotify datasheet using the Pearson correlation coefficient. This coefficient is a value between -1 and 1. If the coefficient is close to the number 0 there is no correlation between data points. Positive coefficient indicates a positive correlation, whereas negative shows an inverse correlation. The popularity row is the most important row because we are trying to predict the popularity of songs based on our inputs. The best positive correlations that we have for popularity is loudness, danceability, and energy with values 0.31, 0.26, and 0.2 respectively. For negative correlation we have acousticness, instrumentalness, liveness, and speechiness, with values -0.34, -0.16, -0.19, and -0.18 respectively. However, we also noticed that the correlation coefficients between loudness, energy, and acousticness are high, which are -0.73, -0.69, and 0.82. This means that those three attributes are highly correlated with each other, so we only need one of them to predict our result. Among the three attributes, loudness is easier to

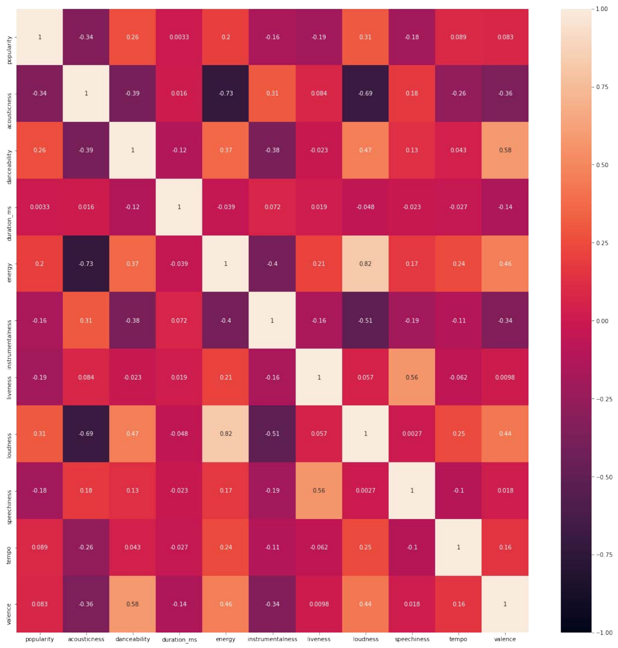


Figure 1: Heatmap of Data Set Attributes

measure in songs and less ambiguous than energy and acousticness. Therefore we chose loudness to reduce the parameters in our model.

3.2 Pair Plots

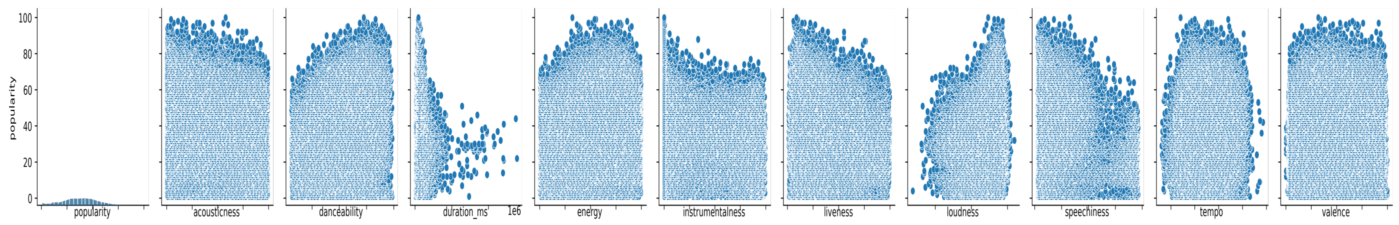


Figure 2: Pairplot of Spotify Data Set Popularity Attribute

As we can see from this pair plot, there isn't an apparent correlation between any pair of attributes. Most pairs have data points evenly distributed throughout the plot (72/110). This pair plot indicates that it is impossible for us to use linear or binary-output machine learning algorithms, such as linear/logistic regressions and SVM, to predict the result. Therefore, we decided to move forward with PCA and t-SNE to analyze the data instead.

3.3 PCA

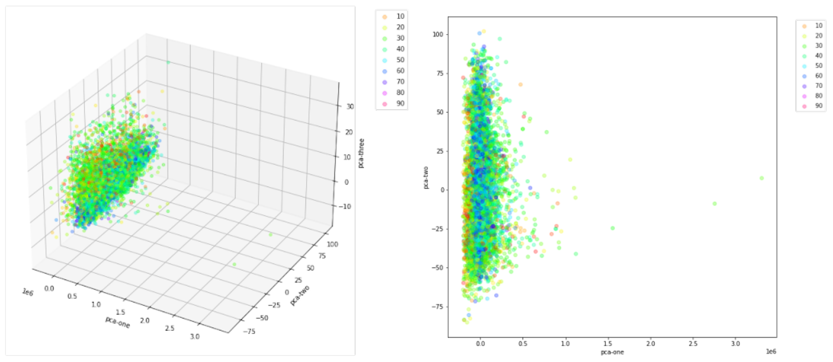


Figure 3: PCA of Spotify Data Set Attributes

PCA is a method of reducing the number of variables while minimizing data loss by choosing the dimensions that maximizes variance of the data. We also analyzed the PCA plot of the data and found that there seems to be an ideal range of variables for popular music. Most music that deviates from the range has a low popularity score.

3.4 t-SNE

The t-SNE is useful when we need to deal with complex, high-dimensional datasets. It can show the correlations between variables by simplifying a high-dimensional graph into a two-dimensional graph. For our research, we split our dataset by the ratio of 70:30, and use the 30% part to train the t-SNE. It not only decreases the train time but also makes a less dense graph for us to analyze. We ran the data at a perplexity of 100 and 1,000 iterations. The result shows an elongated shapes graph with no significant space between labels, which indicates that our data has a random distribution among values. In t-SNE graphs, data that is similar or close in the high-dimensional space will stick together in the low-dimensional space, which is what creates the long lines of clusters. The graph minimizes the space between points of a cluster while maximizing the space between clusters, thus creating lines with many points but that have significant gapping with other lines.

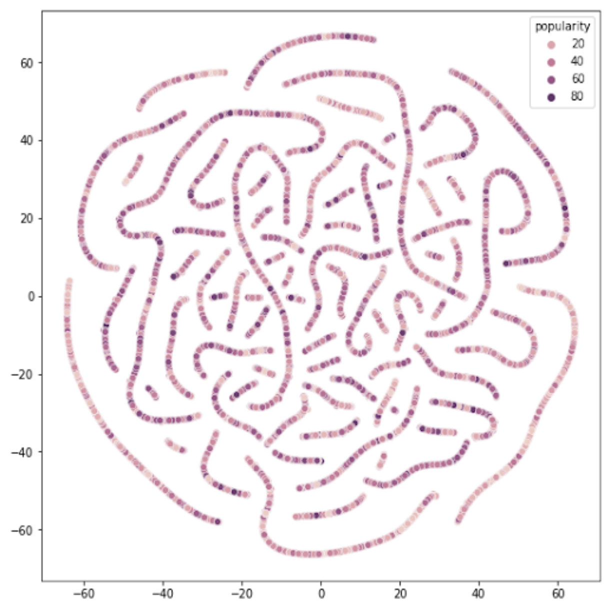


Figure 4: t-SNE

4 Proposed Methodology

4.1 Preprocessing

We started data preprocessing with data normalization. Since we wanted all of our data points to be somewhere between 0 and 1, Min-Max Normalization was used to trim all the data for later use. Then, we decided to use "popularity" as our output variable. The popularity attribute in the dataset ranged from 1-100, and for our purposes we partitioned the popularity into 5 groups (1-20, 21-40, 41-60, 61-80, and 81-100) so we could rate songs on a popularity scale of 1-5. As result, we created a multi-layer perceptron model that consumed selected inputs to make predictions on the popularity score from 1-5. Finally, to ensure the generalizability of the model, we had split the data into training and testing sets, where the training set used 80% of all data and the rest for testing. The logistics behind choosing a 80-20 split was simply because it was the most commonly used split in Machine Learning.

4.2 Development

4.2.1 MLP Classifier

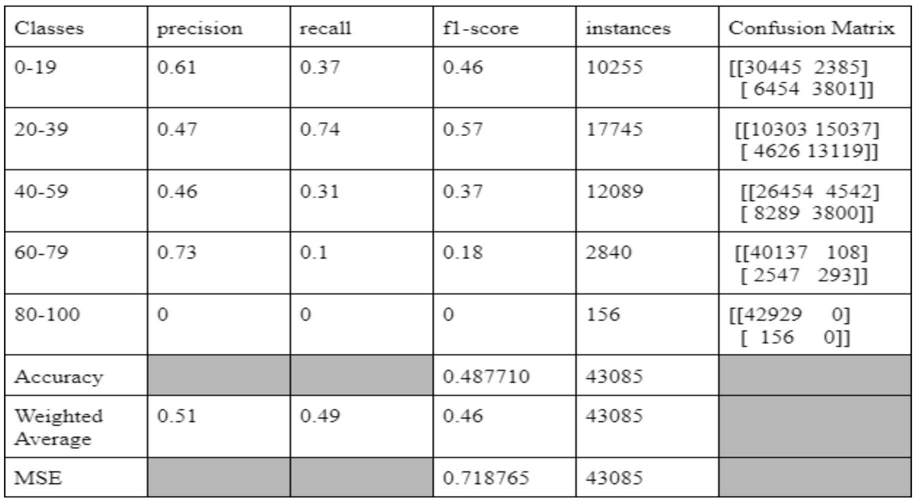
Neural networks(NN) are a subset of machine learning in which objects called nodes mimic the way neurons would fire in a human brain. They are composed of three types of layers of varying size, the input layer, the hidden layer and the output layer. MLPClassifier is a neural network designed to take in some input, put it through some bias known as the weight in the hidden layer and come up with an output. The weight is generated by giving the MLPClassifier data to train with. To create our MLPClassifier, we imported it from the sklearn.neural_network library, trimmed the useless part of the data, and normalized it. We then took our data and filled in our hyperparameters and ran a grid search to feed our MLPClassifier the best parameters from among them. The hyperparameters we had to fill in were, activation, learning-rate-init, hidden-layer-sizes, and max-iter. Activation dictates the functions that are used to determine whether a "neuron" fires or not. Learning-rate-init is the initial learning rate the model uses, it controls how much the weights are updated. Hidden-layer-size determines how many input layers, hidden layers, and output layers are used in the model. Max-iter defines the number of iterations that the model can run through.

In order to choose the optimal hyperparameters for our model, we used Grid Search to try different combinations of hyperparameters. Grid Search is a technique that takes the hyperparameters we gave and automatically runs each combination of them. Then, it will provide us with the optimal combination of hyperparameters and their accuracy score. Due to our large dataset, it is impossible for us to run the test of all hyperparameters at the same time, because it will take hours to finish. Therefore, we cut the tests into parts and decide each hyperparameter separately. First, we choose the hidden_layer_sizes and learning_rate_init for activation "logistic" and "relu", and found the appropriate max_iter by running the following two tests combinations.

The result shows that logistic has a higher accuracy score when layer size is 3 and the learning rate is 0.2 (score: 0.506321). Relu has a higher accuracy score when layer size is 2 and the learning rate is 0.02 (score: 0.498369). Then, we run the test for two activations separately to determine the best performance.

The optimal combination of these values is {'activation': 'relu', 'hidden_layer_sizes': (19, 35), 'learning_rate_init': 0.02, 'max_iter': 400} which give us an accuracy score of 0.514996. This is the highest accuracy we have for our dataset. After this, we also run several tests on smaller learning_rate_init and larger hidden_layer_sizes, but there is no significant difference in the accuracy score.

4.2.2 Evaluation for MLPClassifier



Classes	precision	recall	f1-score	instances	Confusion Matrix
0-19	0.61	0.37	0.46	10255	[[30445 2385] [6454 3801]]
20-39	0.47	0.74	0.57	17745	[[10303 15037] [4626 13119]]
40-59	0.46	0.31	0.37	12089	[[26454 4542] [8289 3800]]
60-79	0.73	0.1	0.18	2840	[[40137 108] [2547 293]]
80-100	0	0	0	156	[[42929 0] [156 0]]
Accuracy	▨	▨	0.487710	43085	▨
Weighted Average	0.51	0.49	0.46	43085	▨
MSE	▨	▨	0.718765	43085	▨

Figure 5: MLPClassifier Evaluation

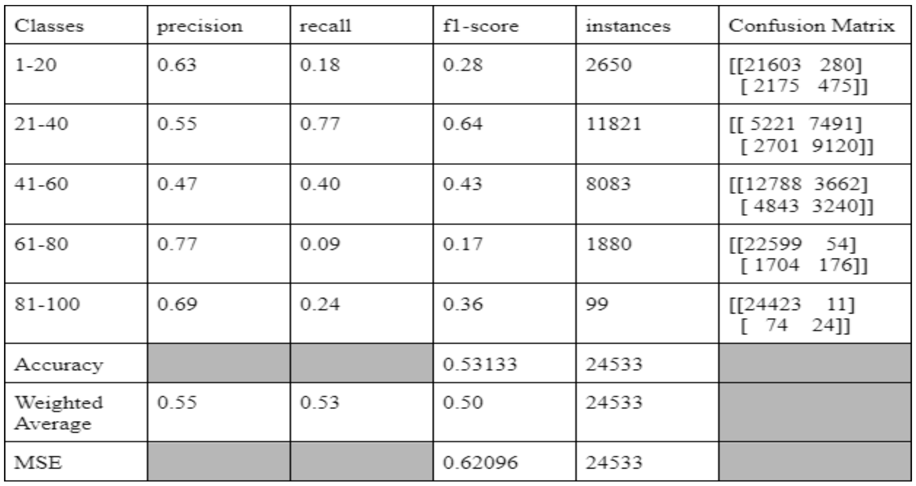
After running our model, we achieved a mean squared error of 0.72 and an accuracy of 0.49. These numbers demonstrate that the model is able to roughly estimate the popularity of each song. However, this comes with a caveat: the model completely discards the possibility that a song may have a popularity of higher than 80. This is shown by the fact that true positives and false positives are zero in the highest class. This is likely caused by the fact that these songs only represent a small percentage of the data. They are thus overshadowed and were not learned by the model. The class [0-19] also has a significantly higher false negative rate due to its small data size. A method to fix this issue is to oversample the data by copying existing data points and give each class an equal amount of data points. However, performing oversampling actually decreased the accuracy of our model. Due to the poor prediction accuracy of the deep neural network model, we decided to develop a new model by using a different algorithm.

4.2.3 Random Forest

Decision Trees are the foundation of many classification algorithms in machine learning such as Bagging, and Boosted Decision Trees. Random Forest, a subset from the Decision Trees, fits data into more than one decision tree and uses the average result to increase the accuracy. It also prevents overfitting by randomly choosing the trees from its tree pool.

We implemented our model using the RandomForestClassifier class from the sklearn library. However, we ran into some trouble during hyperparameter tuning. At first, we thought increasing the n_estimator (number of trees) and the depth of trees would increase the accuracy of our prediction. Therefore, we used n_estimators =1500 and unlimited depth for our data. However, after developing models with smaller n_estimators values and limit depth, we realized that decreasing the n_estimators values and depth not only decreased the running time, but also increased our model accuracy. After several tests, we finalized our parameters, which is (n_estimators=200, max_features = 'auto', max_depth = 20).

4.2.4 Random Forest Evaluation



Classes	precision	recall	f1-score	instances	Confusion Matrix
1-20	0.63	0.18	0.28	2650	[[21603 280] [2175 475]]
21-40	0.55	0.77	0.64	11821	[[5221 7491] [2701 9120]]
41-60	0.47	0.40	0.43	8083	[[12788 3662] [4843 3240]]
61-80	0.77	0.09	0.17	1880	[[22599 54] [1704 176]]
81-100	0.69	0.24	0.36	99	[[24423 11] [74 24]]
Accuracy			0.53133	24533	
Weighted Average	0.55	0.53	0.50	24533	
MSE			0.62096	24533	

Figure 6: Random Forest Evaluation

After training our random forest model, we achieved an accuracy of 0.53 and a mean squared error of 0.62, both of which show improvement over our deep neural network model. Furthermore, the distribution of predictions is more even, especially for songs with high popularity. Rather than learning and updating weight to predict the output, random forest divides and conquers the data into groups, which reduces bias towards overrepresented data points (classes [21-60]). Since it provides better accuracy and distribution of predictions, random forest seems to be the better-suited model for the task. There are still some problems with the model, as shown by the low recall and f1-score for classes [1-20] and [61-80]. However, that may be caused by the inherent complexity of song popularity that is not described by the data.

5 Conclusion and Discussion

Creating a model to predict the popularity of a song is difficult because song preferences are constantly evolving throughout time, and it is the reason why our model can't find a clear correlation to popularity. Another reason could also be that the Spotify API does not give us all the information we need to predict the popularity of a song. In conclusion, the main reason our model has low accuracy is because of the massive scope of the data set and insufficient information from the Spotify API to analyze music.

The low correlation to popularity between our inputs is due to the vast amount of data that mixes tastes from older and newer generations. According to Myra (2018) there has been a downward trend in happiness and an increase in sadness in popular songs. Human taste is not static, and just how popular fashion changes on a yearly basis, so does music taste. Another reason is that the Spotify API calculates popularity based only on recent plays and not overall popularity. This means we can have two similar songs with popular attributes that differ only by the time they were released but have two different popularity scores. The best way to solve this would be to add a date attribute to the datasheet. Using the Spotify API we were able to add a date attribute and were able to slightly increase our model's accuracy.

The Spotify API information might also not have all the answers we need to produce a model to predict the popularity of songs. The Spotify API is hidden from us, so we have no way of knowing how they calculated their features besides the description they gave in their documentation. Music is multifaceted and has more features than the Spotify API provides, so we may be missing important pieces of information to predict the popularity of songs. For example, Nunes (2014) talks about how repetitive lyrics tend to be more successful because they are processed more fluently and therefore adopted more widely. To increase accuracy, we would need new functions to analyze songs, which is a difficult task to do.

Through creating this model, we learned that music is not two-dimensional and has more layers underneath that make it difficult to predict whether a song will be popular. Predicting if songs will be popular or not is much harder than we expected because of all the layers that songs have, and it would require a much more complex, holistic dataset.

6 Bibliography

Bedre, Renesh. “T-Sne in Python [Single Cell RNA-Seq Example and Hyperparameter Optimization].” Data Science Blog, 6 Jan. 2021, https://www.reneshbedre.com/blog/tsne.html.

Interiano, Myra, et al. ”Musical trends and predictability of success in contemporary songs in and out of the top charts.” The Royal Society Publishing, 16 May, 2018. https://royalsocietypublishing.org/doi/10.1098/rsos.171274

Nunes, Joseph, et al. ”The power of repetition: repetitive lyrics in a song increase processing fluency and drive market success.” Journal of Consumer Psychology, 31 July, 2014. https://doi.org/10.1016/j.jcps.2014.12.004.

Peker, Philip. “Predicting Popularity on Spotify-When Data Needs Culture More than Culture Needs Data.” Medium, Towards Data Science, 17 Nov. 2021, https://towardsdatascience.com/predicting-popularity-on-spotify-when-data-needs-culture-more-than-culture-needs-data-2ed3661f75f1.

Wattenberg, Martin, et al. “How to Use T-Sne Effectively.” Distill, 13 Oct. 2016, https://distill.pub/2016/misread-tsne/.

”Web API Reference: Spotify for Developers.” Home, https://developer.spotify.com/documentation/web-api/reference//operations/get-several-audio-features.