Making Track Popularity Rating Predictions based on Track Features

Hongju Lee (hongjlee)

Introduction

People have consistently found music essential in their lives, whether for enjoyment in listening, the emotional support, performing, or expressing their inner creativity. According to a credible news source, more than 100,000 songs are uploaded daily to Spotify and other DSPs such as SoundCloud and Apple. With the rapid development of technology and the significant increase in the number of audio platforms people can enjoy, the popularity of tracks is more easily trackable and visible to the world even compared to a few years ago. We have indeed come a long way from an era where the popularity of songs is calculated by counting the number of records sold.

And with advanced technology, the music itself has also evolved. Instrument sounds can be created with a click of a keyboard and numerous new genres have risen to the surface. Active listeners are now more exposed to diverse music and with that, genres that were new just a few years back have become a steady genre that people cannot live without. Through this research, I hope to conduct deep audio analysis on track features and perform prediction analysis to predict the popularity of tracks based on their track features.

The main question of interest was, "can popularity level be predicted by the track's features?". Although the question may seem simple and easily solvable, it was more complicated than it seemed. It required deep analysis of the feature itself which I must consider to effectively build a viable prediction model. By going through extensive data research, pre-processing, data analysis, and application and validation of various machine learning models, I was able to build a feasible prediction model that predicts the track's popularity level based on its features and characteristic. I strongly believe that this research will not only help the business industry to understand and analyze the music trend to make valid business decisions, but also help artists who are creating new songs to stay on top of trends and to reach more diverse listeners.

Methods

1. Getting the data

Multiple datasets were combined for this project: Spotify API, and Spotify dataset. Because the purpose of this project is to predict the popularity of a track using the track features, I utilized two datasets to train and test the model. The Spotify dataset that was used to train the model was obtained from "Spotify Dataset 1921-2020, 600k+ Tracks" in Kaggle. The dataset provides about 600,000 tracks that were created from 1921-2020. As this project is focused on the track's features and their influence on the popularity of the track, I selected the track features' data, popularity and release date data.

To accurately estimate the popularity of random tracks, I also imported track data from a playlist that contains top tracks from all time using the Spotify API. Although Spotify API is a public data source, registration is needed for each user and there's a 100 rows limit for each import. I utilized a python library that is mainly created for utilizing Spotify API called Spotipy to bring the data from the Spotify network. More information about the utilization of this library can be found in this documentation.

The two main reasons for using two different data sources are to perform an objective analysis of the training data and to check the generalizability of the model. Data pre-processing and data

transformation were essential to this process to avoid data leakage. After extracting the data from the pre-selected playlist, I performed data cleaning to the original dataset to remove all the corresponding data. The original dataset contained randomly selected 234532 rows and after performing the process I mentioned before, the final dataset was calculated to be 234432 rows.

2. Data Preprocessing

Here are the steps that I took to get the final version of the dataset:

- Extracted data using Spotify API to get the track data from the all-time favorite playlist
- Extracted data from the tracks file that was downloaded from Kaggle
- Performed data cleaning on the Spotify dataset to match the original dataset (unifying column names and data types from both datasets)
- Dropped null values
- Dropped all matching rows
- Converted released_date column data type to DateTime and transform the data to only represent the year
- Performed data transformation on duration_ms which represents the tracks' duration in a millisecond and create a new column for duration_min which represents the duration of the track in minutes

3. Data Description

Predictor/Feature Name	**Description**
id	Unique representation of track generated and provided by Spotify
Release_date	Release date of tracks (year)
Duration_min	Duration/length of tracks in minutes (float)
Popularity	Popularity score of tracks (int)
Danceability	Danceability of tracks (float)
Energy	Energetic tracks that feel fast, loud and noisy (float)
Loudness	Decibels of track (int)
Speechiness	Speech likeability of tracks, as it gets closer to 1, the more exclusively speech like recording is (float)
Acousticness	Acousticness of a track (float)
Liveness	Liveness of tracks, detects the presence of audience (float)
Valence	Valence of tracks, shows the level of positiveness (float)
Tempo	Beats per minute (int)

Table 1. Feature Description

4. Feature Engineering

- Performed data normalization using MinMaxScaler() on two-track features: loudness, and tempo. Loudness and temp are continuous numerical values in the range of 0 to 180 which mismatches with other features that are set to range from 0 to 1.
- Binned popularity feature to group the intervals of continuous numerical data into 5 bins. The popularity value was represented as a continuous numerical value from 0 to 100. To apply classification models to the data, the data were binned so that each value was represented as a number from 0 to 4, with zero as the lowest popularity and 4 as the highest.

- Resampled imbalanced popularity feature using RandomOverSampler() to avoid imbalance data. Because the popularity feature had a skewed distribution, an over-sampling strategy was applied to the popularity data so that the number of examples of the minority class match the majority. The final distribution count for all the classes has become 7157 after applying this approach.

Audio Feature Analysis

A feature analysis was also conducted an initial investigation of the data to discover patterns, spot anomalies, analyze the relationships between features and check assumptions. Going through several processes to examine the data from multiple standpoints, I was able to uncover some interesting points about the data.

- Overall loudness of the tracks has been highly unified to stay in the range of 0.75 to 0.85 which means that the loudness tracks have been in this small range for over 8 decades. However, it was interesting to also see that there is a mild but definite increase in general loudness over the years.
- The energy of the tracks has been gradually increasing over the years. During the early 1920s, the energy level was at a low of 0.2 and increased to 0.6 by 2010.
- There are also observations where features showed a high correlation with each other. As shown in figure 3, energy and loudness features show a high positive correlation of 0.9 which means that when the level of loudness increases, energy increases as well. In contrast, energy and acousticness showed a strong negative correlation of 0.98 where energy increases or decreases and acousticness changes in the opposite direction.

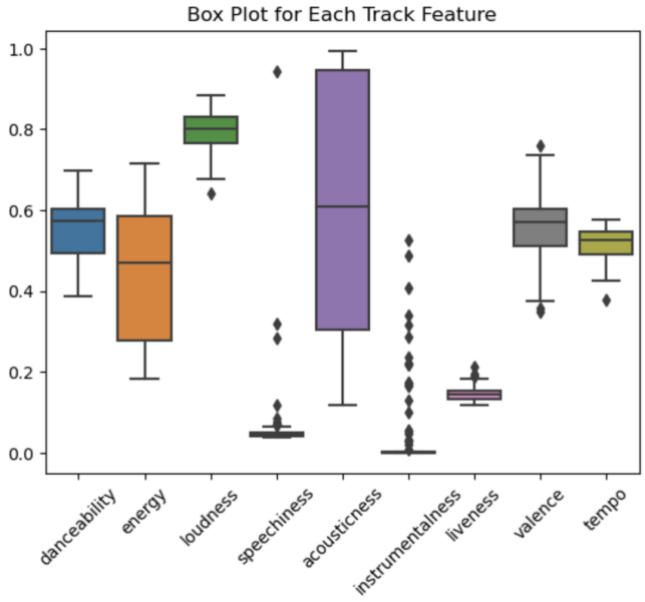

Figure 1. Box Plot for each Track Feature

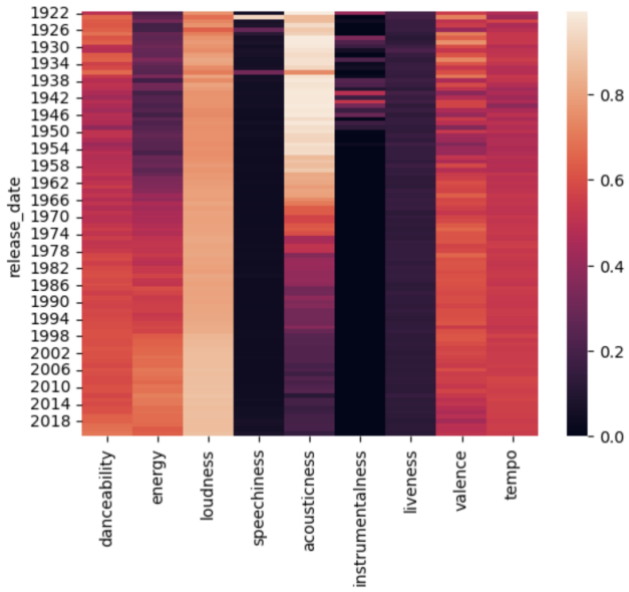

Figure 2. Heatmap for Each Track Feature by Year

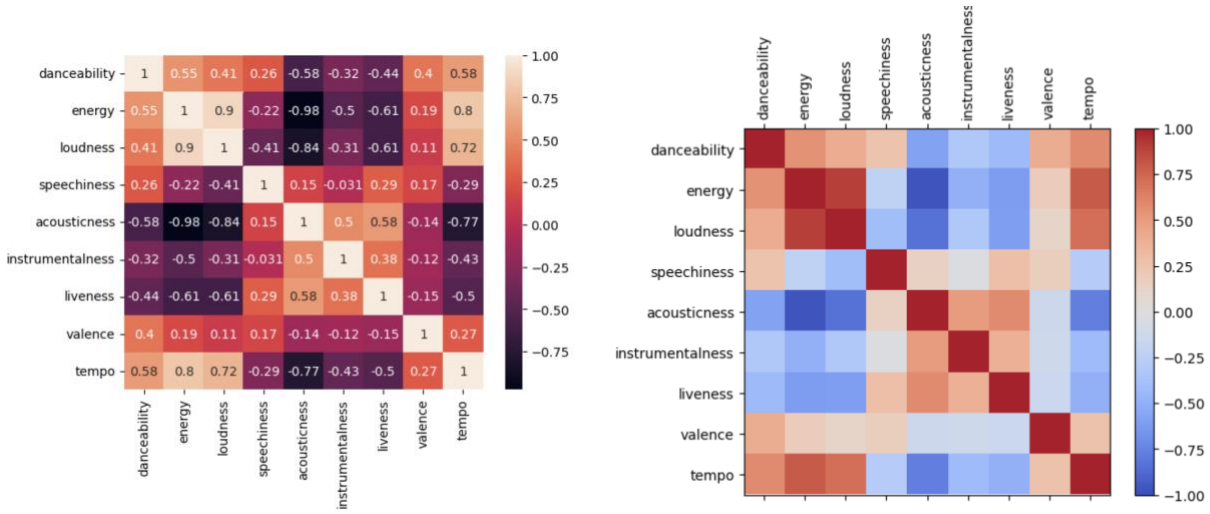

Figure 3. Correlation Maps for Track Features

Model

To find the best model to accurately predict the popularity of the tracks, I fitted the dataset on various classification models: k-means clustering, logistic regression, ridge regression, support vector machine, k-nearest neighbor, random forest, gradient boost, ada boost, XGBoost, and multi-layer perceptron. These algorithms are selected based on an IBM developer source where authors, Madhavan and Sturdevent implemented a comparative study of the current most popular classification algorithms.

I begin with conducting feature importance to select the best predictors and fitted my dataset to the pre-selected classification models. Feature importance was also an important step for using the KMeans clustering model to find the optimal number of clusters to inject into the parameter.

Figure 4 shows release date, loudness and acousticness are the top three features of importance well as the ranking of the features based on their importance.

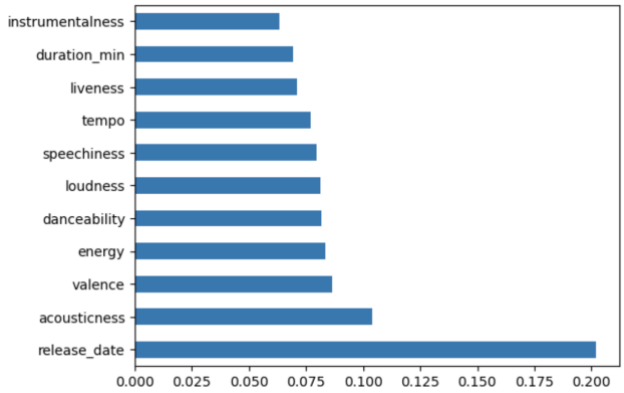

Figure 4. Random Forest Top 15 Feature Importance

Based on this information, I performed KMeans clustering analysis starting with the elbow method to find the optimal number of clusters. As shown in Figures 4 and 5, the best number of clusters to inject into the KMeans algorithm parameter was 4 as the slope of the line starts to flatten from that point.

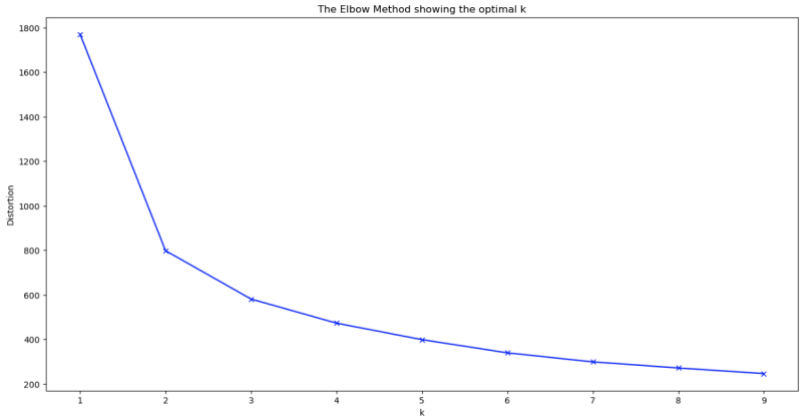

Figure 5. The Elbow Method Showing the Optimal K

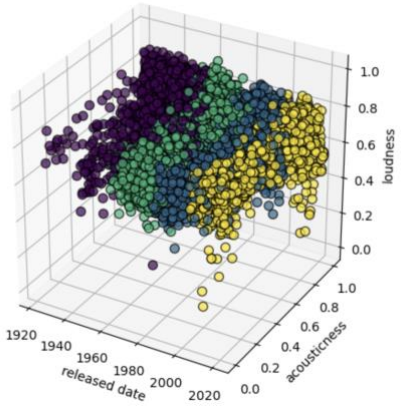

Figure 6. 3-D Scatterplot for KMeans Clustering on the Top Features

For initial prediction modeling, I set the models to their default settings and performed hyperparameter tuning after finding the best-performing model for time efficiency. Also, to develop the models, 80% of the dataset was used to train the model and the rest was used to test the model.

Best initial model: Random Forest Classifier
Hyper-tuning parameter:

- max_depth: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]
- max_features: ['auto', 'sqrt']
- min_samples_leaf: [1, 2, 4]
- min_samples_split: [2, 5, 10]
- n_estimators: [5,20,50,100]
- 'bootstrap' : [True, False]}

Final model with the best parameter:
Random Forest Classifier(bootstrap=False, max_depth=60, max_features='auto', min_samples_split=5)

Evaluation and Analysis

There are 5 metrics that I used to evaluate classification models:

1. Confusion matrix

· A confusion matrix is a cross table that shows the number of data classified to each category. According to credible data science blog, Mohajo, the author, explained that unlike binary classification, there are no positive or negative classes for multi-class classification. Looking at the diagonal elements in figure 6, which represent the total correct values predicted per class, prediction accuracy for each class is relatively high. Eg: 1472 values have been correctly predicted as belonging to class 4 out of 1490 values. The diagonal elements for class 0 and class 1 are nearly the same green with mediocre accuracy which could have been caused by the Random Over Sampling process as those two classes were the majority class.

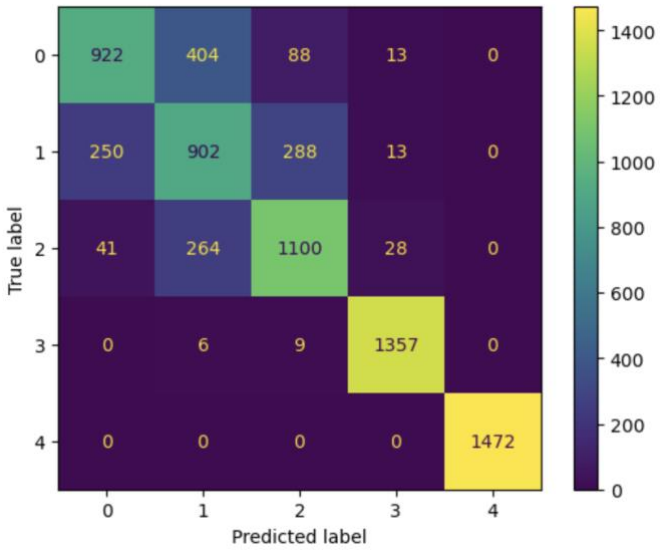

Figure 7. Confusion Matrix for Each Popularity Class

2. Precision

· Precision is a good metric when you would like to minimize the Type I error. (Reject the null hypothesis when it is true).

$$\text{Precision} = TPTP + FP$$

3. Recall

· Recall is a good metric when you would like to minimize the Type II error. (Accept the null hypothesis when it is false)

$$\text{Recall} = TPTP + FN$$

4. F1-score

· The F1 Score is a good metric to select a model with considering both Precision and Recall and to use for imbalanced datasets.

$$\text{F1 score} = 2 * \text{Precision} * \text{RecallPrecision} + \text{Recall}$$

5. ROC-AUC score

· Receiver Operation Characteristic Curve (ROC) shows how True Positive Rate (TPR) is changing accordingly when False Positive Rate (FPR) is changing. A good ROC curve is pushed towards the top left side both for positive and negative classes. Calculating ROC AUC, Area Under the ROC Curve, formulates number that tells us how good the curve is and how good at ranking prediction the model is.

Precision Score	Recall Score	F1 Score	ROC AUC Score
0.8038	0.8038	0.8038	0.9582

Table 2. Performance of Random Forest Algorithm for Predicting Popularity based on Track Features

The main metrics I utilized to measure the performance of my prediction model are ROC AUC and f1 scores. Both metrics were useful to use as they value different goals. F1 scores provide a score that shows how well the model has produced well-calibrated probabilities whereas ROC AUC provides a score that shows how well the model identifies a class. Both scoring metrics were valuable metrics to review and since the target popularity variable was balanced using random oversampling before fitting into the model using F1 and ROC-AUC seemed appropriate.

Among the classification models I have implemented through this project, the random forest model had the highest ROC AUC and F1 scores of 0.958 and 0.798 when the data was fitted to the initial model with default settings. Using RandomSearchCV, I tuned the parameters to find the best hyperparameters for this model and was able to increase the f1 score to 0.804. Table 2 shows the result of the scoring metrics for the final random forest model.

Conclusion

The best classification model I was able to build from this research was using a random forest classifier with an f1 performance resulting in approximately 80% accuracy and ROC of 95%. These results prove that the model can be used to predict the popularity of tracks based on the track features. However, this research can be continued in several directions to address its limitations. As shown in Figure 3 which shows the Top 15 feature importance calculated using random forest, the most impactful feature was the release date, which is the year a track was released. Second came acousticness, the level of acousticness of tracks. Although the track features data Spotify offered were sufficient to predict the popularity, the difference between the release date and the rest of the features is rather high. To mitigate the issue and increase the overall accuracy of the model, additional feature extraction may be promising.

Moreover, applying different benchmark algorithms such as RFQ, SMOTE-RF, SMOTEBoost and RUSBoost to avoid the inescapable nature of the imbalance dataset. According to a data science practitioner named Vasilyeva, mentioned in the one of her posts that SMOTEBoost and RUSBoost are both widely used to resolve class imbalance problems. For this research, the Random Oversampling method was used to balance the number of examples in each of the popularity feature classes. However, this method is very vulnerable to overfitting as the method requires the same information to be copied in the minority class to match the majority class. The limitation of this method can be solved by SMOTE which works by utilizing a k-nearest neighbor algorithm to create synthetic data.

Through this research, I have learned the importance of data reliability which results from proper data cleaning and transformation steps. Most of the time was heavily invested to make sure that I have good data quality because unqualified data could lead to inaccurate results. According to IBM research publication, many researchers and practitioners focus on improving the quality of models while investing very limited efforts towards improving the data quality and reliability. The authors mentioned, "One of the crucial requirements before consuming datasets for any application is to understand the dataset at hand and failure to do so can result in inaccurate analytics and unreliable decisions." Because I utilized multiple data from different sources, it was important to make sure the data does not contain any data gaps, anomalies, and duplicates and ensure the data I put into our models are accurately structured and correct.

Although the model I have created in this research has an acceptable prediction score, there are still many ways to improve the model on top of some possible future work I have explained earlier. Given the state of the music industry and its continuously evolving nature that it's in, I believe that extracting other related variables can improve this research significantly. Those variables include but are not limited to, nostalgia score which represents the popularity of preceding tracks and artists' popularity score as I can suspect that tracks made by a famous artist are more likely to gain more listers than those that are made by a comparably unknown artist.

Full code for this research can be found on my Github.

References

"The Importance of Music in Our Society." *GILBERT GALINDO*, www.gilbertgalindo.com/importanceofmusic.

Jain, Abhinav, et al. "Overview and Importance of Data Quality for Machine Learning Tasks." *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &Amp; Data Mining*, ACM, Aug. 2020, https://doi.org/10.1145/3394486.3406477.

---. "Overview and Importance of Data Quality for Machine Learning Tasks." *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &Amp; Data Mining*, ACM, Aug. 2020, https://doi.org/10.1145/3394486.3406477.

Mohajon, Joydwip. "Confusion Matrix for Your Multi-Class Machine Learning Model." *Medium*, 14 Dec. 2021, towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

"Music Streaming Hits Major Milestone as 100,000 Songs Are Uploaded Daily to Spotify and Other DSPs." *Variety*, 6 Oct. 2022, variety.com/2022/music/news/new-songs-100000-being-released-every-day-dsps-1235395788. Accessed 14 Dec. 2022.

Vasilyeva, Anna. "Using SMOTEBoost and RUSBoost to Deal With Class Imbalance." *Medium*, 13 June 2018, medium.com/urbint-engineering/using-smoteboost-and-rusboost-to-deal-with-class-imbalance-c18f8bf5b805.

Web API Reference | Spotify for Developers. developer.spotify.com/documentation/web-api/reference.