Capucine Philippine Leroux (cpl2146) | Vedant Gannu (vg2565) |
Chris Berniel Cobashatse (cc4536) | Xiangming Huang (xh2550) | Narjes Al-Zahl (na2852)

Background and context

The project aims to predict whether a song will become a hit on Spotify by developing a prediction model. The project focuses on Spotify due to its vast reach and open-source web API. Factors such as long-term vs short-term success, number of streams, ratio of plays to followers, chart records, and music certifications can be considered to gauge a song's popularity. The prediction model needs to be consistent, robust, and account for the evolution of a song's popularity over time. The methodology devised by the project predicts a song's popularity score based on the current popularity score provided by the Spotify API at the time of data acquisition.**Our aim is to predict the popularity score of a song based on the current popularity score provided by the Spotify API at the time of data acquisition.**

Identification and Description of Dataset

Our project's baseline dataset consists of over 600,000 songs, which can be accessed at the following link: (https://www.kaggle.com/datasets/yamaerenay/spotify-tracks). Constructed using the Spotify web API (https://developer.spotify.com/discover/), this dataset includes a wide range of features, such audio features (e.g., danceability, energy), release date, duration, popularity score, artist name, genre, and number of followers. Our goal is to ensure that our dataset is as comprehensive as possible to create an accurate model for predicting a song's popularity.

Initial Data Exploration

Exploring the data is an essential step before building any model. In our data exploration phase, we utilized various techniques to understand the distribution of features and identify potential outliers.

Popularity Distribution. We started by using stacked bar plots to visualize the density distribution of popularity in the dataset, which revealed a right-skewed distribution. We found that less than 12.5% of the songs have a popularity score of 60 or higher, indicating an imbalanced dataset (see figure 1).

Feature Distribution and Outliers. To further analyze the distribution of each feature, we used histograms and boxplots. Boxplots helped us measure the distribution of data within each feature, and we found features with a lot of outliers such as duration_ms, danceability, loudness, speechiness, instrumentalness, liveness, tempo, and followers. Although the instrumentalness column had 101,821 outliers, it is worth keeping because it might correlate with popularity (see figure 2). Moreover, this column accounts for one-quarter of the dataset rows.

Correlations. We also used a correlation matrix to determine the relationship between numerical features. We found that with a threshold of 0.6, loudness and energy, and acousticness and energy are highly correlated. Thus, it might be useful to eliminate energy from the numerical feature list (see figure 3).

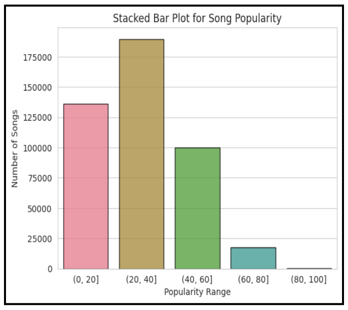


Figure (1)

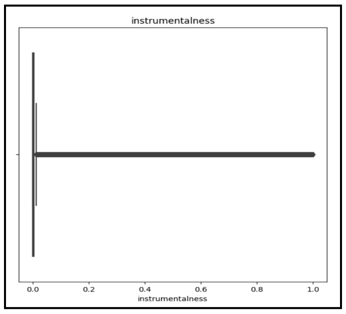


Figure (2)

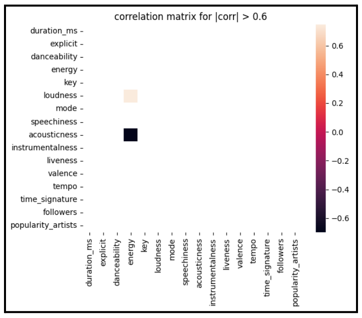


Figure (3)

Track Frequency. We also explored the release frequency of tracks by analyzing the number of entries by day of the week, month, week of the year, and year. We discovered that tracks were more often released at the beginning or end of the year (see figure 4). However, the distribution of tracks released during the week was fairly balanced. Moreover, we found that many tracks in the dataset were released in the 1990s, 2000s, and 2010s, and the average popularity score increased year over year, probably due to the increase in access to mainstream media consumption tools such as radios, TVs, internet, mobile phones, and Spotify (see figure 5).

Artist Frequency. Furthermore, we plotted the frequency of each artist showing up in the dataset and found that most artists had 0 to 10 tracks in the dataset (see figure 6). Only a small number of artists had tracks larger than 50. We further separated all artists with total tracks larger than 100 and plotted all the feature histograms, which showed that the distribution was very similar to the overall dataset.

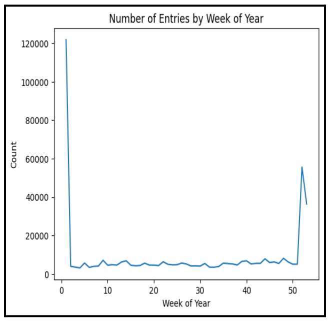


Figure (4)

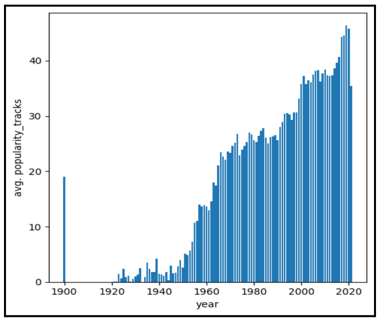


Figure (5)

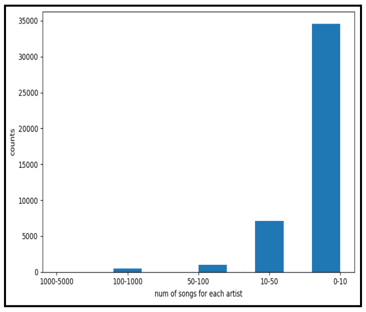


Figure (6)

Data Cleaning and Feature Selection

Cleaning and Sampling. We reformatted the artists' column by stripping away the brackets and applied the same preprocessing on columns genres and id_artists in tracks.csv. We merged the two datasets on the id of the artist to obtain 470k rows, 55k unique artists and an average of 8.5 tracks per artist. We dropped about 37k rows that had empty genres, resulting in an average of 10 tracks per artist. We also removed false duplicates in songs that were released several times by the same artists.

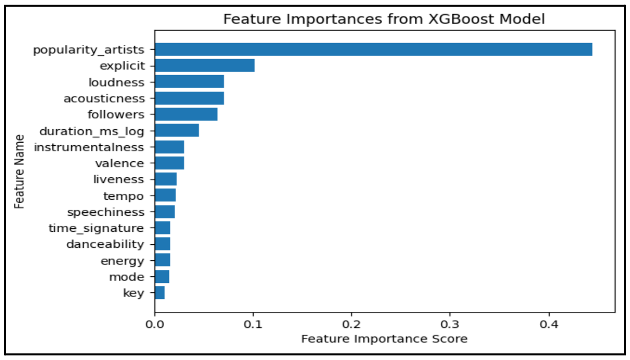
Outliers. We kept the outliers for instrumentalness as it might correlate with track popularity. For liveness, we removed all 30k rows with a value greater than 0.8; if a song is performed live by an artist, there is a high possibility that there is an identical song that is not live. For speechiness, we found that tracks with a value above 0.66 are entirely made of spoken words, and about 40k rows had speechiness values above this threshold. Since this project focuses on identifying popular songs, we decided to remove all outlier rows for speechiness.

Implemented ML models

Ensemble Models

Overview: Gradient Boosting and XGBoost were used both for classification and regression. Initial training of classifiers for both models ran for a long time (more or less 40 min) while yielding very low scores (6% accuracy). These results (see below) led us to shift our focus toward regression as we are trying to predict many classes (scores from 0 to 100), a problem that can be better solved through regression. Regression is not ideal as it makes predictions continuous values but it is the better approach in our case. See results below.
Training and Hyperparameter Tuning: Baselines of both models had relatively okay scores (40-50%). We then tuned 3 hyperparameters (n_estimators, learning_rate, and max_depth) to identify the most optimal ones. Hyperparameter tuning ran for a very long time (more than 3 hours), so we deduced optimal values from trends in initial results.
Feature Importance: We also run iterations of the tuned XGBoost with the top 4, 6, and 8 important features.
Results: Overall, hyperparameter tuning and feature importance improve the models, but only to a certain extent.

Model	Dev Score	Test Score
XGBoost Classifier	0.3174	0.0682
Gradient Boosting Regressor (Baseline)	0.4456	0.4404

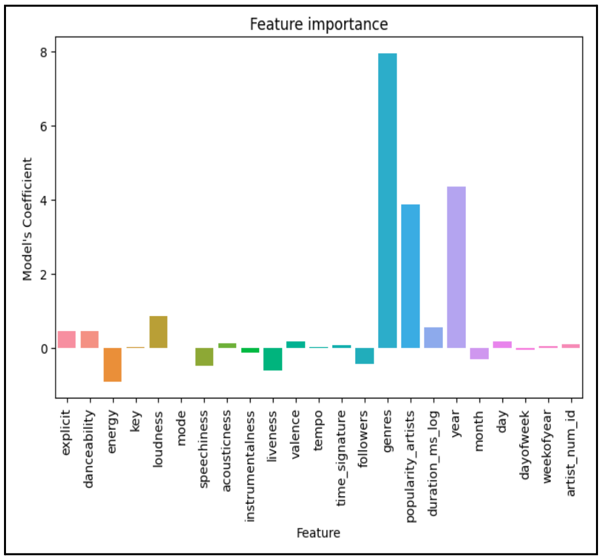


XGBoost Regressor (Baseline)	0.5435		0.5108	
Gradient Boosting Regressor (Tuned)	0.7658		0.5593	
XGBoost Regressor (Tuned)	Initial	0.7524	Initial	0.5591
	Top 8 features	0.7125	Top 8 features	0.5593
	Top 6 features	0.6900	Top 6 features	0.5610
	Top 4 features	0.5420	Top 4 features	0.4107

Regression Models

<u>Preprocessing, Encoding and Data Splitting:</u> First, we encoded the categorical features. We used ordinal encoding for the id and name of the artists, as it provided a natural way to classify them. For the genres, which had a large number of categories (4407), we opted for target encoding.Although we initially considered a structured split based on release date, we ultimately chose a random split (80-20) due to the relationship between popularity and date. We then standardized the data and checked for highly correlated features, but found none.
<u>Regression:</u> We tested several regression models, including linear regression, regularized linear regression (ridge, lasso, and elastic-net), tree regression, and random forest regression. For regularized regression and tree models, we started with default settings and performed hyperparameter tuning. However, for tree and random forest models, we observed significant overfitting on the development set and addressed this by early-stopping using a maximum tree depth.
<u>Performance and Feature Selection:</u> We compared the performance of each model and visualized the importance of different features (see below). Then, we ran a second pass on the model, removing the features that were consistently less important across models ("tempo", "valence", "dayofweek", "key", and "mode", fig = linear model).
<u>Feature Selection:</u> The feature-selection improved the tree and random forest scores, but not the linear and regularized methods. The overall best model is the tuned random forest with feature-selection (score of 74.86% on the test set).

Model	Dev Score Before Feature Selection	Dev Score After Feature Selection	Test Score Before Feature Selection	Test Score After Feature Selection
Linear regression	0.6168	0.6168	0.6170	0.6169
Vanilla Ridge	0.6168	0.6168	0.6170	0.6169
Tuned Ridge	0.6168	0.6168	0.6170	0.6169
Vanilla Lasso	0.6026	0.6026	0.6035	0.6035
Tuned Lasso	0.6168	0.6168	0.6170	0.6169
Vanilla Elastic-net	0.5747	0.5747	0.5756	0.5756
Tuned Elastic-net	0.6168	0.6168	0.6170	0.6169
Vanilla Tree	0.9996	0.9996	0.4847	0.4909
Tuned Tree	0.6789	0.6792	0.6785	0.6793
Vanilla Random Forest	0.9645	0.9646	0.7467	0.7473
Early stopping RF	0.9636	0.9645	0.7468	0.7486

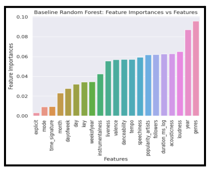
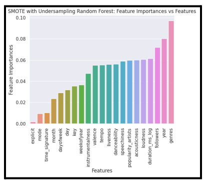


Classification using Ensemble Trees andOversampling

<u>Overview:</u> This task aims to assess whether we can achieve equally as high results as regression, by using RandomForest Classifiers and a combination of thresholding undersampling, and SMOTE oversampling.
<u>Preprocessing:</u> All features were numerical except for genres, which had 4407 unique genres and 18524 combinations of genres. We used target encoding to preserve the potential importance of genres, but prevent a

Capucine Philippine Leroux (cpl2146) | Vedant Gannu (vg2565) |
Chris Berniel Cobashatse (cc4536) | Xiangming Huang (xh2550) | Narjes Al-Zahl (na2852)

sparsity issue arising from techniques like label encoding and one-hot encoding. Scaling was not used. For oversampling the development dataset, we removed rows with popularity tracks greater than 70, randomly undersampled class ‘0’ to 7000 rows, and used SMOTE to oversample all minority classes. Our resulting dataset went from 299640 to 516312 rows.
Training and Tuning: Two Random Forest Classifiers were created: 1) baseline for unbalanced dataset and 2) SMOTE tuned. A 5 iteration Randomized Grid Search was performed on each to select best classifiers from each.
Feature Importance: For the baseline classifier, top 5 features were genres, year, loudness, acousticness, and duration_ms_log. For the SMOTE classifier, they were genres, year, followers, duration_ms_log and loudness.
Results. Combining undersampling and SMOTE oversampling proved effective since an OOB score of 0.213, accuracy of 0.485, and weighted F1 score of 0.486 on testing dataset beat the balanced Random Forest Classifier. In the future, we could use LightGBM or HistogramGradClassifier since it uses a histogram approach and has faster convergence.

Model	OOB Score	Accuracy Score	F1 Score (weighted)	Feature Importance
Balanced Random Forest Classifier	0.0954	0.0682	0.0952	
Tuned SMOTE Random Forest Classifier	0.2133	0.4853	0.4864	

Neural Network Regression

Preprocessing: To build a neural network model for predicting track popularity, we used the TensorFlow Keras library and the cleaned dataset. To ensure data integrity, we removed rows with duplicate track names, keeping only the ones with more recent published dates. Since non-numeric columns don't contribute to popularity, we dropped them from the dataset. After verifying that none of the remaining columns were highly correlated using a correlation matrix, we split the data into development and test sets using an 80:20 ratio. To standardize the data, we called a standard scaler from the sklearn library and applied it to the development set using fit-transform. We then transformed the test data accordingly. Next, we further split the development set into training and validation sets.
Training and Scores: Our model had two hidden layers with 150 and 10 nodes and used the ReLU activation function. As this was a regression model, the output layer had a size of 1 and used a linear activation function. The model had 4821 parameters. We compiled the model using the ADAM optimizer and the MeanSquaredError loss function. After training the model using the development set, we achieved an R^2 score of 0.6 on the test set.

Conclusion.

Using Spotify's open-source statistics, we aimed to develop a predictive model for a song's popularity by identifying its key features. Through data exploration and cleaning, we selected the most important features and built our models. Although not perfect, our models offer a promising framework for predicting a song's success and could be refined with further research. This project demonstrates the power of data analysis in the music industry, providing a valuable resource for artists and professionals.