Predicting Spotify Song Popularity

Final Report

Taylor Willingham
2019

Problem

The music marketplace is fickle. Artists that once topped the charts fall out of favor, while new artists may unexpectedly take the spotlight. There never seems to be much of a pattern and, regardless of what any one person may know, it's practically impossible to predict which songs will make the cut.

With that in mind, the goal of this project was to test whether or not a supervised machine learning model could perhaps provide a little more certainty to this task. By knowing which songs are popular on Spotify and the characteristics of those songs, perhaps we can find correlations algorithmically that will provide a little more confidence when choosing the next 'hit'. The hope is that tastemakers can use more than just their intuition when deciding which songs are worthy of their marketing efforts and financial investment.

It's important to clarify that the goal of this project is not to make personal recommendations. This is not guided by the tastes of a specific user in order to make suggestions for that user. Rather, this project looks at popular songs in a collective sense, as determined by the general population. As such, the intended target is more along the lines of a music programmer or an A&R executive: someone who would benefit from knowing which songs will have the most impact. These decision makers choose which songs to promote based on what they think will most likely hold a listener's attention, and any tool that can help them improve their accuracy will help them to be more effective in that role.

The Data

The data for this project is from a Kaggle dataset which, in turn, was originally collected from the Spotify API. The data represent the popularity of the songs at the time of collection, so is likely to change frequently. The collection used for this specific project is from April 2019.

Each observation represents an individual song, and the dataset prior to any modification contains approximately 130,000 rows. There are 17 columns in the dataset, the majority of which contain numerical values providing a score on various musical qualities. Some examples include: loudness, instrumentalness, danceability, liveness and the like. Finally, the last column, the target column, lists a popularity score between 0 and 100 for each song.

Data Wrangling

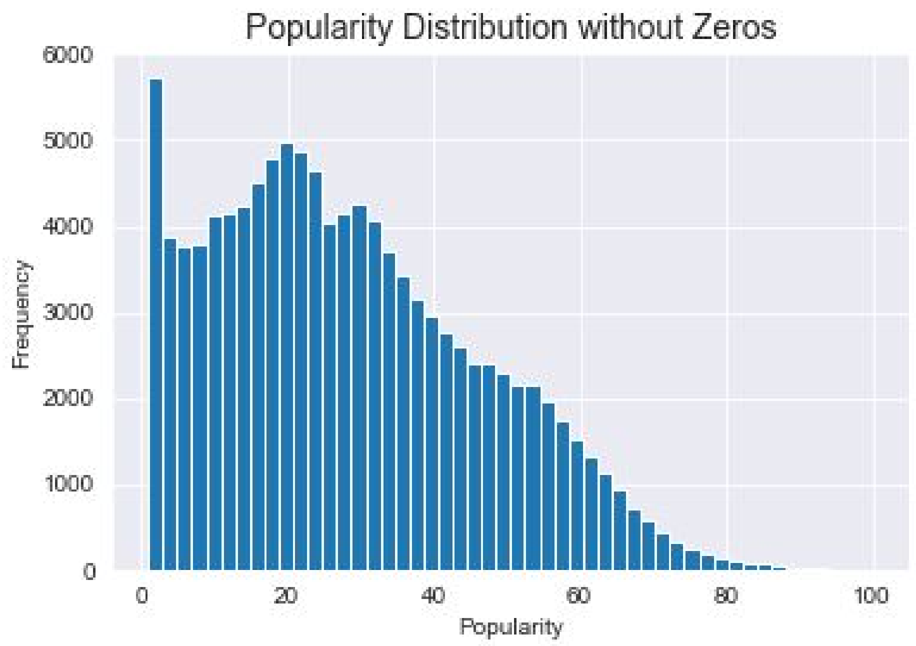
Since the data for this project came from a Kaggle dataset, it was fairly clean and didn't require too much wrangling. However, there are a few things I did to make the dataset a little better suited for analysis. These steps involved sorting the data by the popularity scores and re-indexing the data to create a sort of song ranking. Another step was to convert some of the features from numeric to categorical. For example, the observations for time-signature could be represented with a 3 or a 4 or some other number, but these differences should not be thought of in an ordinal fashion. One particular time-signature is not necessarily better than another, and I didn't want the algorithms to think of these features in such a manner. The columns for key and mode were also adjusted accordingly.

Furthermore, while there weren't any missing values that needed to be dropped or imputed, I did decide to drop all rows with a popularity score of zero. This resulted in a reduction of 18,889 rows. The thinking for this is that unpopular songs might not be popular simply due to the fact that the artist lacks any sort of recognition or the necessary fan following to have a song played. Even if the song has the exact same qualities as a popular song, if no one knows of the song or the artist, then it won't get played. This is merely an effect of pop-culture in general, and completely accounting for this dynamic is beyond the scope or ability of this project. However, by taking the step of dropping these observations I hoped to at least cut out some of the noise inherent in the data. After dropping these rows, roughly 111,000 observations remained.

Finally, I also created a new column titled pop_rating to help differentiate the various levels of popularity. After linear regression proved ineffective, I decided to use a classification approach instead. Based on intuition, I split the observations into three groups: those with a popularity of at least 75 labelled as popular, those between 50 and 74 labelled as medium, and those below 50 labelled as unpopular.

Exploratory Analysis

With the data shaped, it's important to get an understanding of the different variables, in particular the target variable. First, here's a histogram to visualize the distribution of the target variable values:



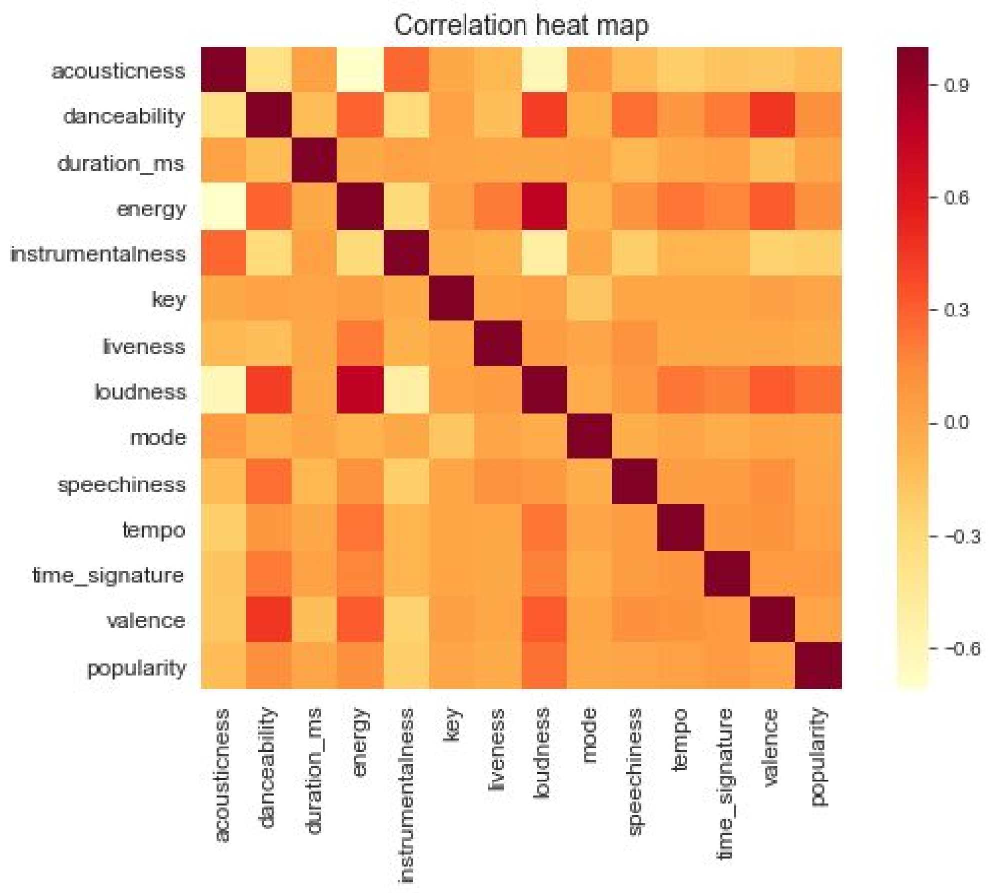
Even with the zero values dropped, the popularity scores are heavily weighted towards the bottom of the scale, and the distribution tapers off severely as it moves towards the higher popularity values. Studying the statistical moments for the popularity variable, all of the measures of centrality for the target variable are low. The mean and median are both in the

20s and the IQR only extends to 41. Even songs of a medium popularity as defined in the classified dataset would be fairly rare. This makes sense given that popular radio is often branded as "Top 40" and there are thousands of songs in the data, but the large imbalance will need to be accounted for.

mean:	28.300132
Std:	18.398418
Min:	1.000000
25%:	14.000000
50%:	26.000000
75%:	41.000000
Max:	100.000000

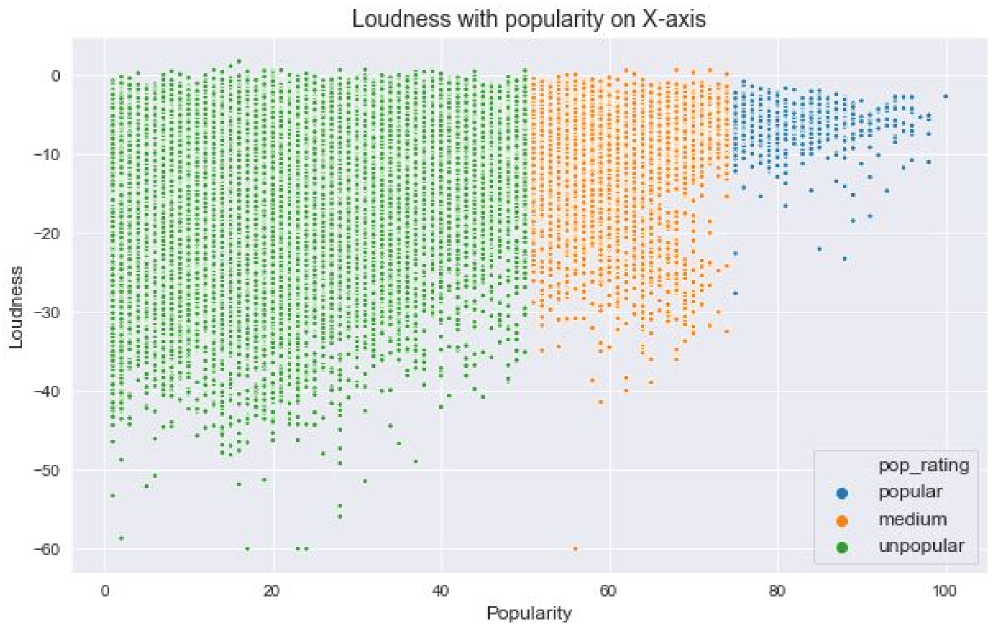
Correlations

With this understanding of the target variable, it was important to next examine the rest of the features and see how they correlate with each other, in particular how they correlate with the popularity column. To also visualize these correlations, a heatmap is useful.

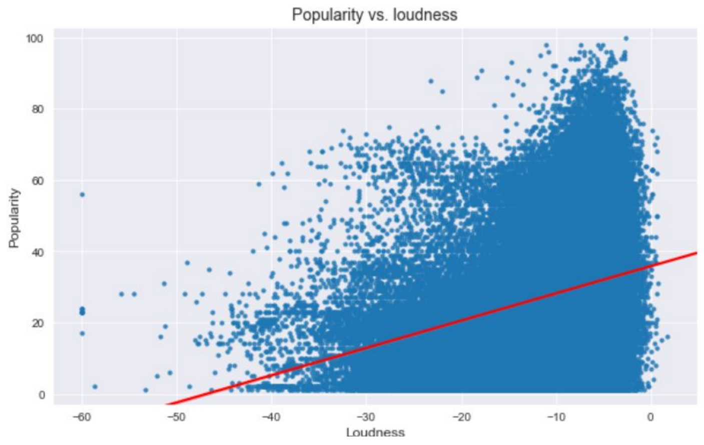


From the above, the correlations are mostly pretty weak. Energy, loudness and danceability have fairly strong relationships, which makes sense, and acousticness is naturally opposed to loudness and energy, but there's not much to go on outside of those. Unfortunately, this trend appears to hold true for the popularity variable. The correlation between loudness and popularity looks positive, but it would be nice to have a dependent variable that strongly relates to the target variable.

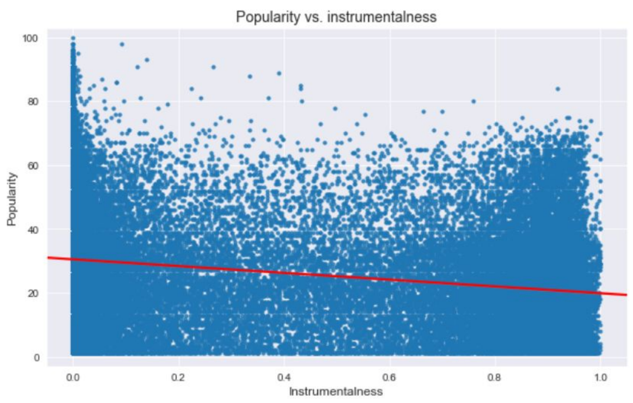
Exploring the loudness variable further, the variable with the strongest correlation, generates the following plot, demonstrating how popular songs tend to consistently have high loudness values.

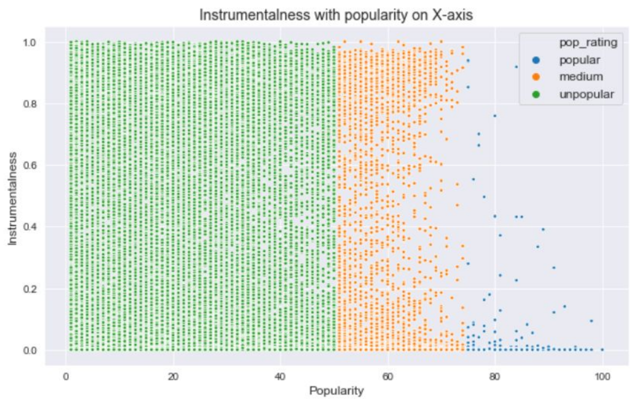


Not all loud songs are popular, but all popular songs tend to have high loudness values. It’s good to see some evidence of a trend, but looking at it linearly shows that the fit is not very strong.



Other variables exhibit similar behaviour. For example, here is a look at a couple plots comparing instrumentalness to popularity.

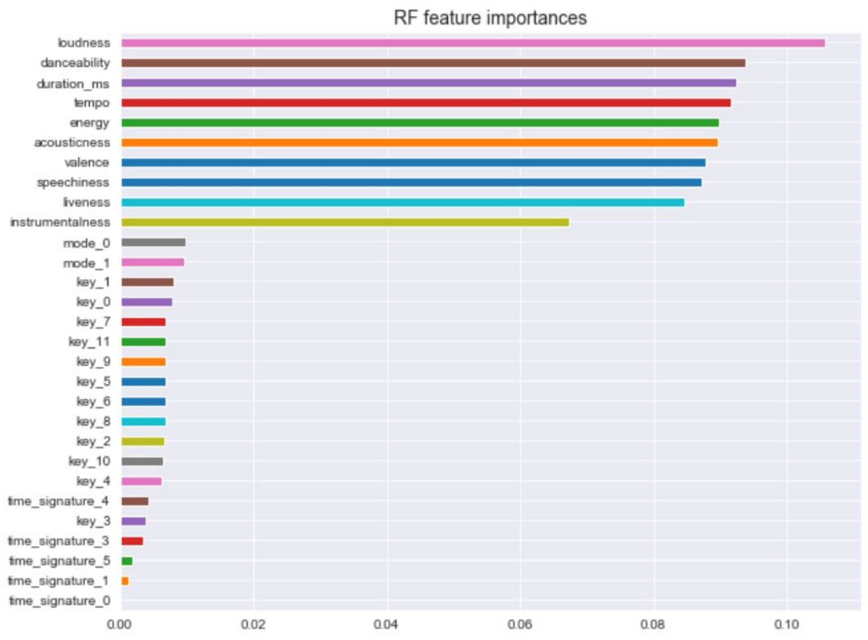




The correlation for this variable happens to be negative, but the other characteristics are similar: the popular observations tend to coalesce in the same area while observations in the other categories exhibit high degrees of variability.

In general, the EDA showed that the data is heavily weighted towards unpopular songs and that there is a lot of noise and variability amongst the observations. None of the scatter plots demonstrated much definition in terms of a consolidated shape. A strong predictor variable was lacking, and there was a wide range of values across the popularity spectrum. In other words, songs with very different popularities can have identical values for any one particular variable.

The lack of a strong predictor variable can also be seen via this plot of feature importances. The categorical variables all rank toward the bottom, but they are split up so their importance might be magnified when considered collectively. Ultimately, no single feature clearly dominates the rest.



But honestly, thinking intuitively about song popularity, this shouldn't be too surprising. Unpopular songs and popular songs can have many similarities, and all popular songs can have many differences. The purpose of this project is to see if an algorithm can sort out these differences and similarities better than a human could.

Models

For the modelling portion of the project, I originally set out to predict popularity scores through linear regression. The scores are laid out from 0 to 100, and predicting specific values seemed like a natural thing to do. However, after various attempts with linear regression and ridge regression this did not prove effective, so I eventually moved to classification. After splitting the training and final test sets, done manually to ensure the imbalanced class ratios are preserved, I started simple by applying a decision tree model. From there, I worked my way up to ensemble methods using bagging, Random Forest and boosting, which improved the accuracy.

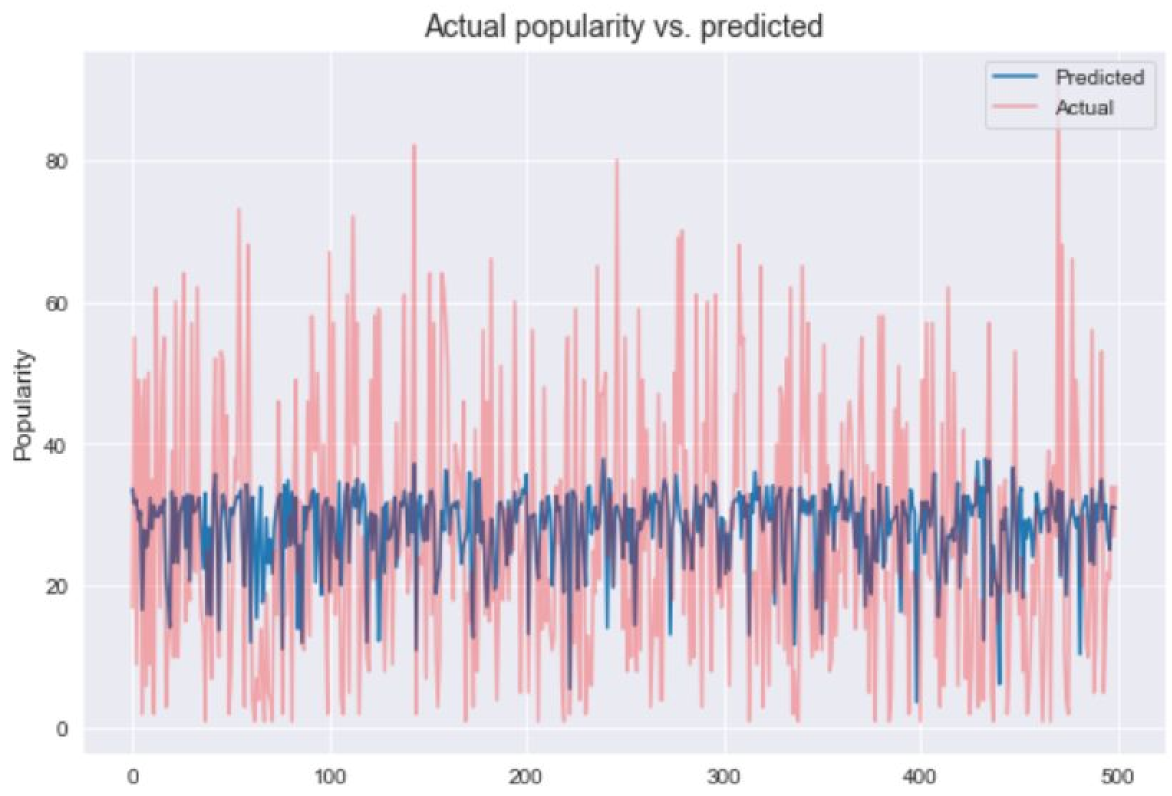
Also, before running the models I decided to drop the artist name column. While song popularity is often correlated with artist popularity, the hope is for this model to work just as well on an artist it hasn't seen before. Part of the goal is to be able to discover new talent, and that becomes more difficult if the algorithm is inclined to discredit a song just because it doesn't recognize the artist. Therefore, the column is dropped for training purposes.

Linear Regression

Out of the box, the linear regression R^2 score was 0.09785, which is very low. After including the encoded categorical variables, the score improved only marginally to 0.09938. This was confirmed through cross-validation.

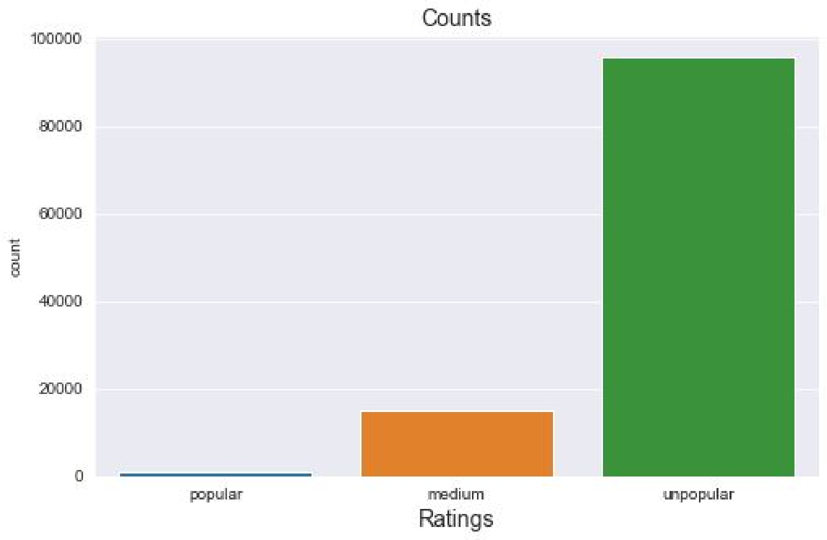
Moving to ridge regression, the score was even worse initially. After tuning the *alpha* hyper-parameter, the score did improve to 0.10035, but that wasn’t a significant change.

The problem with the regression models is that the range of predicted values is much narrower than the actual popularity values. This makes sense when thinking about linear regressions and recalling the scatter plots from the EDA section: the prediction is a single line through a cloud of observations. This plot helps demonstrate that dynamic:



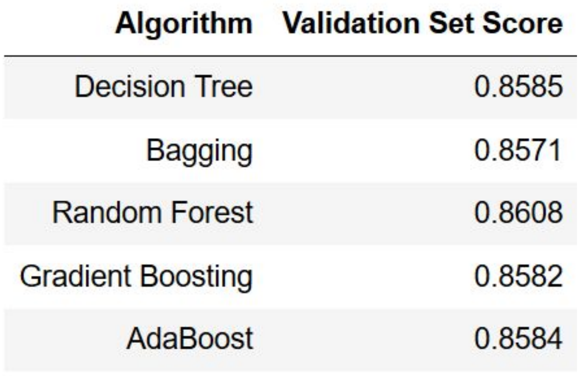
Classification

Because the regression models were not getting the job done, the next step was to try a classification approach. Before jumping in, first it's useful to get an idea of the class distribution.



It’s evident that the data is highly imbalanced, with the large majority of the songs labelled as unpopular. Knowing this, I made sure to maintain this balance when splitting the train and test sets.

When running the various models, I initially focused solely on the accuracy scores to get an idea of how each one performed. I started with a simple decision tree before moving on to ensemble methods such as bagging, random forest, AdaBoost and gradient boost. In the end, after tuning the various models the accuracy performance across the different types was similar.



Algorithm	Validation Set Score
Decision Tree	0.8585
Bagging	0.8571
Random Forest	0.8608
Gradient Boosting	0.8582
AdaBoost	0.8584

Random forest performed the best, but not by much.

However, simple accuracy isn’t necessarily the end goal. The real value would be the ability to predict popular songs, not just correctly predict a large number of unpopular songs. This was tested through a confusion matrix. Under this criteria, all the models actually performed rather poorly, failing to predict many popular songs, if any, and in some cases predicting everything as unpopular. This is to be expected given the data imbalance.

To try and combat this issue, I adjusted the class weights for the different labels. By applying larger weights to the minority classes, the idea is that the model will give those labels more emphasis. This directs the model to focus more on producing true positives for the class of interest. With this adjustment, the accuracy score may have suffered, but it increased the ability to correctly predict popular songs.

Here is the final results table:

Algorithm	Validation Set Score	Validation Set Popular Recall	Hold Out Popular Recall	Execution Time
Decision Tree	0.8585	0	--	1.03 s
Bagging	0.8571	0.67	0.76	18.6 s
Random Forest	0.8608	0.02	0.03	40.1 s
Gradient Boosting	0.8582	--	--	1 min 48 s
AdaBoost	0.8584	0.39	0.46	1 min 5 s

Somewhat surprisingly, the model that improved the most was the bagging model. The random forest model is slightly more precise than the rest, but that advantage is nullified by the poor predictive performance. Meanwhile, bagging technically has the lowest accuracy score, but it was far away the best at predicting popular songs. Additionally, the run time compares favorably with most of the models giving the bagging model a nice balance of effectiveness and computational performance. Based on the results above, I think it's clear that the bagging model is the best choice.

Final Thoughts

Thinking intuitively about the Spotify data and song popularity in general, it seems apparent that clear cut correlations between a song's features and its popularity would be difficult to discern. Two songs may be structured very similarly, and yet may have entirely different popularity scores. On the other side of the coin, two songs with the same popularity score may sound completely different. One song may be fast and the other slow; one may be loud while the other not so much. The EDA and the feature importances from above demonstrate that there isn't a clear set of features to pinpoint as a surefire predictor of popularity. This makes sense, because if there were a clear formula for increasing a song's popularity then every song would be popular which, inherently, is not possible.

All of this speaks to the difficulty, perhaps even folly, of trying to predict song popularity, particularly through numeric data. There is just too much confounding information and too many factors that are difficult to quantify. For example, things like an artist's image, name recognition or unforeseen cultural trends all play a role in determining what is popular, and all of these are difficult or simply impossible to convey mathematically.

On the other hand, the final model did seem to make some gains. It was able to predict most popular songs correctly, albeit with low total precision, and the results of the validation data were generalizable to the unseen hold out data. I think a music programmer or executive would benefit from the model. They would still need to sift through a number of unpopular songs to find the popular songs, but the total amount of work could potentially be greatly reduced. Perhaps this isn't quite the gain we were hoping for, but it is a net gain overall, which is at least part of the goal.