Predicting Spotify Song Popularity

Taylor Willingham

Objective

- Can we use a supervised machine learning model to predict which artists or songs will be popular.

The Data

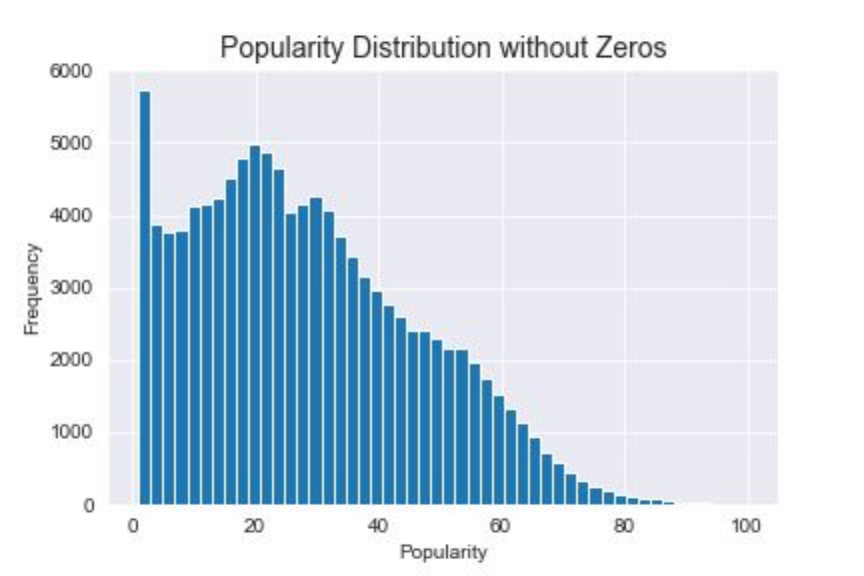
- To find out, we can try using a dataset originating from the Spotify API.
- Each observation represents an individual song, with 17 columns describing the song.
- The target column lists a popularity score between 0 and 100 for each song.

acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	popularity
0.00582	0.743	238373	0.339	0.0	1	0.0812	-7.678	1	0.409	203.927	4	0.118	15

Sample of what the data features look like, excluding the artist name and track name.

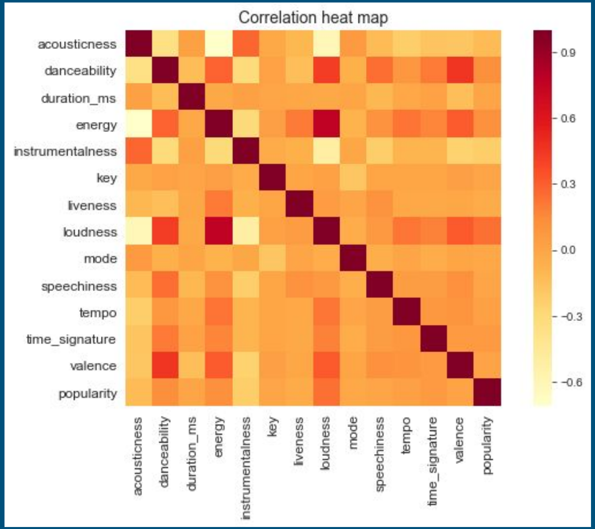
Exploring the Target Variable

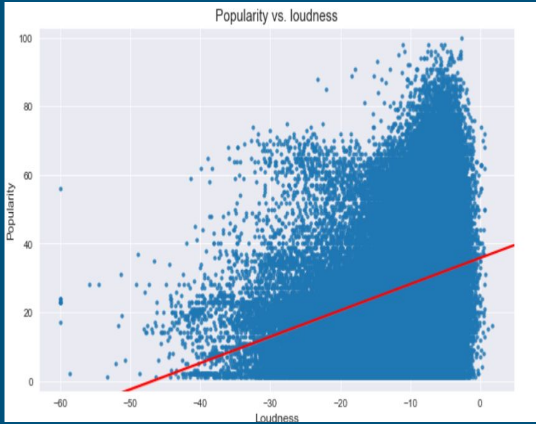
- Some adjustments were made to the data, including dropping songs with popularity of 0.
- Even so, the popularity scores are heavily weighted towards the bottom and taper at the top.
- The measures of centrality -- mean, median -- are very low.

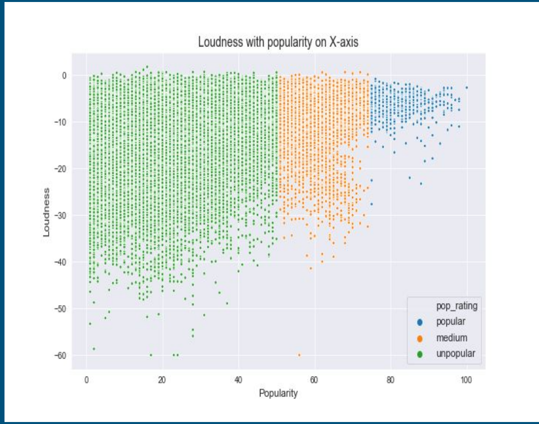


Do any variables correlate to popularity?

- The correlations are mostly pretty weak.
- The correlation between loudness and popularity looks positive, but looking at it linearly shows that the fit is not very strong.
- The dataset lacks a strong predictor variable.



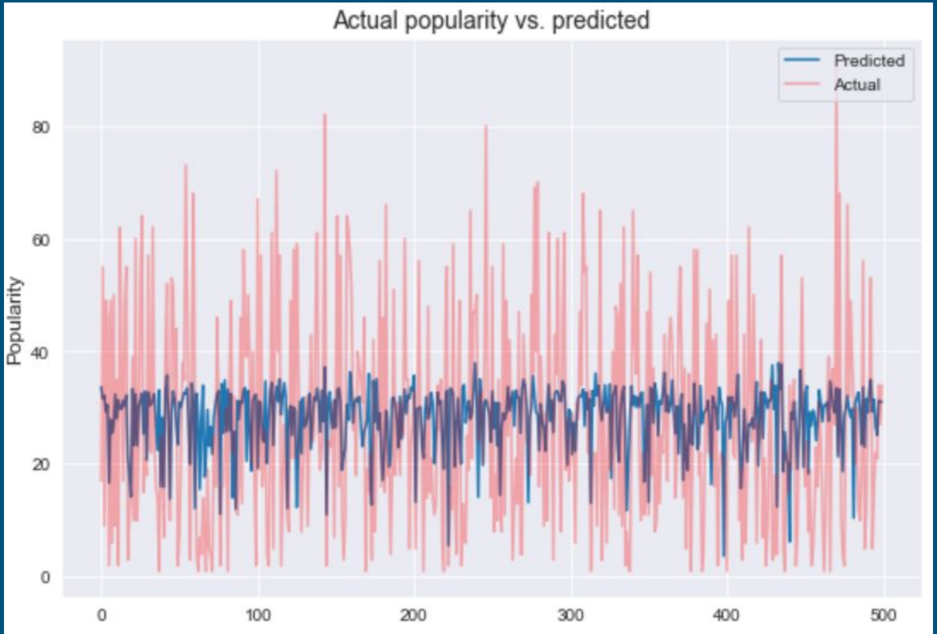




Prediction through Machine Learning

- I first attempted to approach the problem through linear regression. Unfortunately, the regression results were very poor, even after regularization.
- Because of this, I made the decision to tackle the problem using a classification approach by adding a column of popularity labels.
- The decision to use a classification approach opened the door to several different algorithm options including: decision tree, Random Forest and AdaBoost.

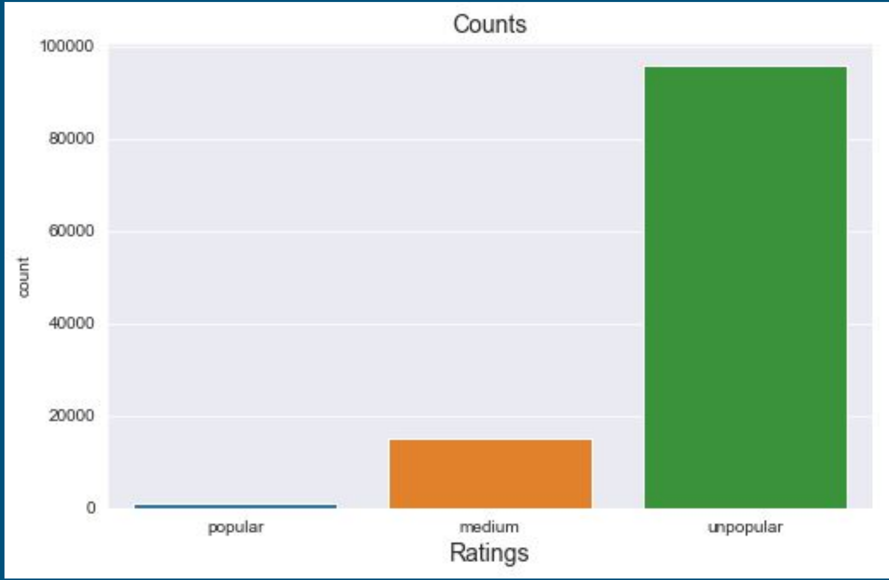
Predicting with Linear Regression



This plot demonstrates the failing of the linear models. The predicted range is too narrow and doesn't capture the full variability of the data.

Setting up for Classification

- I created a new column with class labels. Based on intuition, I split the observations into three groups:
  - Popular - Score of at least 75
  - Medium - Score between 50 and 74
  - Unpopular - Score below 50
- I split the data into a training split and a final test split, making sure to maintain the class balances in each split.



This charts demonstrates the class imbalances.

Models & Scores

- This chart focuses solely on the accuracy scores to get an idea of how each one performed.
- Random forest performed the best, but not by much.

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

Confusion Matrix & Class Weights

- Despite the accuracy scores, the models were poor at predicting popular songs.
- To combat this issue, I adjusted the class weights for the different labels, applying larger weights to the minority class.
- With this adjustment, the accuracy score suffered, but it increased the ability to correctly predict popular songs.

Final Results

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

- The random forest model is slightly more precise than the rest, but that advantage is nullified by the poor predictive performance.
- Bagging technically has the lowest accuracy score, but it was far away the best at predicting popular songs.
- It also has 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.