An Ensemble Learning Approach to Scoring Song Popularity

Sam Xu *samx@stanford.edu*, Joyce Xu *jexu@stanford.edu*, and Eric Tang *etang21@stanford.edu*

***Abstract*—This project aims to predict the popularity of songs by analyzing acoustic features such as tempo, duration, mode, loudness, key; artist information such as location and popularity; and metadata such as release title and year. We extract audio features via EchoNest and the Spotify API, and train a variety of models including linear regression, support vector regression, random forest, and gradient boosting machine to predict the success and popularity of the song. We find that ensemble methods such as random forests and gradient boosting machines perform the best on the high-level audio features of songs.**

I. Introduction

Music is a multi-billion dollar industry in which the majority of revenue is being generated by a limited number of mainstream "hits." These hits, however, are challenging to predict and produce. For one, top songs can vary widely in sound and style. Meanwhile, songs that are similar in sound and style can and do also vary widely in popularity. In addition, trends in music come and go rapidly, so popularity looks different year to year.

With our project, we intend to build a variety of models to predict the popularity of a song by analyzing its audio features and metadata. Such a tool would be valuable for record labels, streaming services, and average consumers; it would also help the research community understand what acoustic features are currently popular with the public. In this report, we present a cohesive feature extraction pipeline and strong ensemble leaning models to predict song popularity in a highly-interpretable manner.

II. Data

A. Dataset

Our dataset comprises 7,473 songs extracted from the Spotify API, which provides acoustic features, author information, song metadata, and a popularity score. Spotify's popularity score ranges from 0 (least popular) to 100 (most popular), and is determined as a function of number of plays on Spotify and the recency of those plays.

We pulled these songs from Spotify's 1,000 top albums released in 2018, which gives us a range of songs across popularity, genre and acoustic features. In addition, limiting our analysis to songs released in 2018 partially controls for the effect of age on popularity.

We further made sure to eliminate all duplicate songs from our dataset, as some songs are cross-listed on multiple albums. This should prevent our score from being artificially inflated, as it would be if some songs appeared on both our training and test set. We use an 85/15/15 train/dev/test split throughout the paper. In this progress milestone, as we are prototyping models without tuning hyperparameters, we report evaluation metrics on our development set.

B. Concrete Example

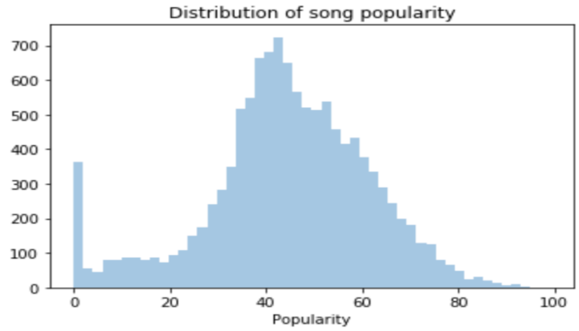
For each song in our dataset, we model its features as extracted from the Spotify and Echo Nest APIs. Here's a sample representation for the song *Never Gonna Give You Up*:

$$
\begin{aligned}
&{}''TRAXLZU12903D05F94'' : \{\\
&\qquad ''features'' : \{\\
&\qquad artist\_name : RickAstley\\
&\qquad loudness : -7.75\\
&\qquad tempo : 113.359\\
&\qquad duration : 211.69587\\
&\qquad mode : 1\\
&\qquad energy : 48.149\\
&\qquad previewurl : p.scdn.co/mp3/99\\
&\qquad ...\},\\
&\qquad \}
\end{aligned}
$$

We format songs' feature dictionaries as Pandas dataframes for consumption, then input the song dataframes into the models described below. Our desired output is simply the Spotify popularity rating. Although a song's popularity fluctuates over time, we predict only the current popularity rating.

C. Data Exploration

Below, we've plotted the distribution of song popularity in our dataset.



Our dataset appears approximately normally distributed around mean 50. Also of note, there is a significant spike of highly unpopular songs with score 0. These unpopular songs likely comprise a ”long tail” of music on Spotify which users rarely listen to.

As a gut check that Spotify’s popularity rating is sensible, we also printed out the most popular songs in our dataset. The five most popular songs in our dataset are:

- ’Promises (with Sam Smith)’, 95
- ’SICKO MODE’, 96
- ’Without Me’, 96
- ’Happier’, 98
- ’Taki Taki (with Selena Gomez, Ozuna Cardi B)’, 100

As we can see, the most popular songs based on Spotify’s rating indeed correlate to the most streamed new releases by famous artists, so our labels appear to be in order.

D. Feature Selection

We extract three broad categories of features for use in our regression models. The first are low-level audio features, such as song duration and time signature, extracted from Spotify’s Audio Analysis feature. The second are high-level audio features such as ”danceability” and ”valence”, extracted from audio files using The Echo Nest API. The third are metadata for each song, such as artist information and genre information.

1) Spotify Audio Analysis: The most low-level audio features are those provided by Spotify’s Audio Analysis feature. This provides both concrete song features, such as duration and average volume, as well as estimated features like the song’s key and tempo. The audio analysis also provides more high-level analysis by subdividing the song into sections (i.e. Chorus, Bridge, Verse) and further subdividing sections into segments.

2) Echo Nest Audio Features: We then add seven high-level audio features extracted by The Echo Nest API, which is integrated into the Spotify API. The Echo Nest uses proprietary audio analysis algorithms to extract high-level features from audio files. The seven features we used are ”acousticness”, ”danceability”, ”energy”, ”loudness”, ”speechiness”, ”tempo”, ”valence”. These audio features are extracted directly from the audio files of each song. The Echo Nest’s algorithms are proprietary, and we are unable to access how they extracted these features, but we use them in all experiments described below.

3) Song and Artist Metadata: We further experiment with utilizing song and artist metadata. In particular, for the artist, we try adding a feature corresponding to the artist’s number of followers on Spotify and report results both with and without this additional feature.

For songs, we experiment with adding genre information. We first extract all labeled genres in our training set, which can be found for each song with Spotify’s API. This provides us 563 different genres, containing everything as generic as ”pop” to oddly obscure labels such as ”miami hip hop,” ”brostep,” and ”vapor soul.”

For every genre that appears in at least 1% of our training set, we add a binary feature to our training data indicating its presence for that song. In our training set, there are 96 genres that appear in at least 1% of songs. This adds 96 more features to our data, which are mostly sparse.

E. Data Split

After extracting these features for each song in our dataset, we converted our list of songs and features into a Pandas dataframe for consumption by scikit-learn models. We then split our data according to a 85/15/15 train/dev/test split, and trained the models described in the following section.

III. Models

For our initial evaluations, we ran the following models on our dataset with the features described above.

A. Linear Regression (Baseline)

For our first baseline, we trained a simple linear regression model on our dataset. Linear regression fits a coefficient or ”weight” to each feature in the training data, which can be multiplied with the feature and summed to obtain a prediction for the popularity score. We implemented and fit the linear regression model in scikit-learn.

B. SVR (Baselin)

As another non-ensemble baseline, we chose to to leverage a non-linear Support Vector Regression (SVR). SVR applies hinge loss and non-linear kernels to regression problems, performing regressions on a dataset and generating non-linear contours. Our SVR used a radial basis function (rbf) kernel, and through hyperparameter tuning we found the optimal regularization parameter to be $C = 10$. We also trained the SVR and performed tuning in scikit-learn.

C. Random Forest Regression

Next, we focused on ensemble models.

Random forests are a class of ensemble algorithms that output the mean prediction from its many decision trees, each of which a) subsamples the feature space and b) subsamples the dataset.

A decision tree takes a set of data points and continuously splits it, one feature at a time, in attempt to accurately predict the class or value of each data point at its leaf node. Decision trees are notoriously high variance, because they easily overfit to specific (possibly non-generalize) patterns in the training set. Random forests reduce this variance by constructing a multitude of trees and ”bootstrapping” (or randomly sampling with replacement) from both the sample space and the feature space. Since the random forest takes the average of multiple trees that are trained on only a subset of the data and a subset of the features, it is more robust to noise in the training data.

Specifically, after training a random forest of B decision trees, we predict:

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

We trained a Random Forest Regression model with scikit-learn on our dataset. Again, through hyperparameter tuning, we found our best performing model on the validation set was a forest with 200 trees and a minimum samples split of 2, meaning the nodes of the tree will continue to be expanded until all leaves of the tree are pure, or contain less than 2 samples.

D. Gradient Boosting Decision Trees

Finally, we tried a second ensemble model: the gradient boosting machine.

A gradient boosting machine is another ensemble model of decision trees. However, unlike random forest, it performs *boosting* instead of bagging (or bootstrapping). Specifically, it is an additive model, constructing one new decision tree per iteration in order to minimize the loss of the overall ensemble.

The final gradient boosting machine is of the form:

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

where each h_m is a base decision tree and the $gamma_m$ is a parameter controlling the contribution of each tree. The gradient boosting machine iteratively ”boosts” - or improves - the overall ensemble by adding an extra decision tree to minimize the loss L at each step:

$$F_m(x) = F_{m-1}(x) + \text{argmin}_{h_m} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i))$$

For training, we built a gradient boosting regressor with XGBoost. Again, through extensive hyperparameter tuning and search, we found the best-performing model used 400 trees, a learning rate of 0.1, and a max depth per tree of 4.

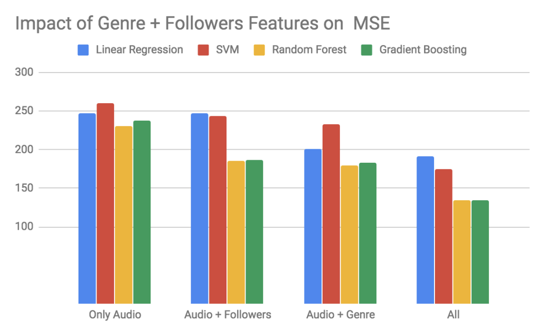
IV. Results

We first report test set results of training the aforementioned models on the full set of features, which includes the audio features, the artist information, and the genre information.

Model	**MSE Loss**
Linear Regression	191.81
SVR	174.03
Random Forest	134.35
Gradient Boosting	135.42

As we can see, whereas our first two models did not perform particularly well on our dataset, our next two models - random forest and gradient boosting trees - perform significantly better. The linear regression baseline is the weakest, and random forest the strongest. However, seeing as the random forest and gradient boosting losses are within a point of one another, they appear to have nearly identical performance.

We reported the model performances with the full set of features above, but we also trained the same models on just the audio features, just the audio features and genre information, and just the audio features and artist followers. We wanted to evaluate how much the genre information improves model performance, as well as how much artist information improves performance. A summary of the comparison is shown in the following graph and table.



Comparative MSE Losses

	Audio	Audio + Artist	Audio + Genre	All
LR	247	247	201	191
SVR	260	244	223	174
RF	231	185	179	134
GBR	234	187	183	135

Unsurprisingly, adding information about artist followers significantly helped the performance of our models, especially the ensemble models. Interestingly, genre information helped *more* than artist information, especially for our baseline models. This is likely because the artist follower information is encapsulated in one feature, and for models such as linear regression, it is easier to learn to put a high weight on a single feature than to learn complicated relationships between 96 genre features and song popularity.

Finally, in the following table, we show the actual popularity and predicted popularity of a few of the most popular and least popular songs in our test set:

Example Random Forest Predictions

Song	**Actual**	**Predicted**
Dangerous (feat. Jeremih and PnB Rock)	0	10.03
Action - Revised Version	0	0.0
Kings Of The World	0	0.0
Almost Slipped	0	9.655
Nuketown (feat. Juice WRLD)	0	21.14
Trio Sonata No. 3 in D Minor	0	5.46
Mozart: Requiem in D Minor, K. 626	0	4.58
...	...	...
Youngblood	85	75.58
Lean Wit Me	86	73.20
Jackie Chan	87	61.88
changes	89	68.86
lovely (with Khalid)	90	55.94
SAD!	92	72.86
Taki Taki	99	63.30

In general, we find that our best-performing random forest model (trained on the full set of features) is fairly good at predicting relative popularity, and can distinguish between very high-popularity and low-popularity songs. However, from the example predictions above, it appears that the model tends to predict more median values, and rarely makes a popularity prediction above 80. This is consistent with the model learning to minimize the l2 loss, which heavily penalizes large prediction errors.

V. Evaluation

A. Model Evaluation

Both linear regression and support vector regressors perform comparatively poorly on our dataset. For the linear regression model, this may be because our dataset likely does not have a decision boundary that is linear in the features of our data points, making it impossible for a linear regression model to learn properly.

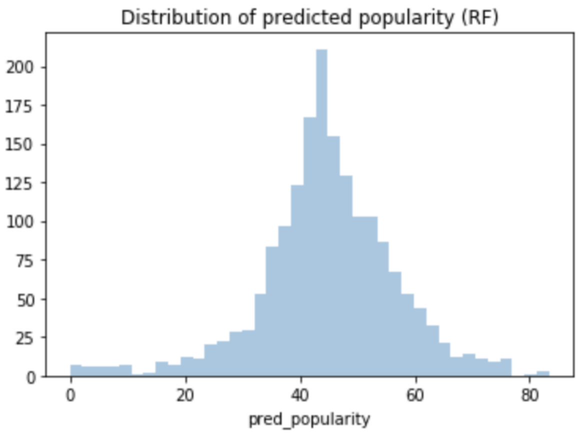
Meanwhile, the support vector regressor likely performed poorly because we were using a Gaussian kernel over the data points, which may be ineffective at learning over the small dataset we have right now. Furthermore, the support vector regressor has the additional downside of being less interpretable. Whereas with linear regression, the coefficients can indicate the ”contribution” of each feature to the final decision, support vector machines with Gaussian kernels go through more transformations over the data than we can observe or explain.

Finally, both the random forest and especially the gradient boosting model work fairly well. There are a couple reasons why these models might outperform our other baselines.

Critically, a large problem with our dataset is that it is fairly small, and models can easily suffer from both high bias (i.e. not fitting the training data sufficiently) and high variance (i.e. overfitting to the training data). Since both random forests and gradient boosting machines are ensemble methods that iteratively attempt to reduce the variance and bias of the ensemble respectively. This allows them to learn a good bias-variance trade-off that is especially beneficial for datasets like ours.

B. Prediction Evaluation

We plot the predicted popularity distribution of songs in the test set using our best-performing random forest model below:



This distribution is consistent with the example song popularity predictions we gave earlier. In particular, we note that popularity predictions are normally distributed and clustered around a mean in the low 40’s, with very few predictions being made in the 0-20 range and the 70-100 range.

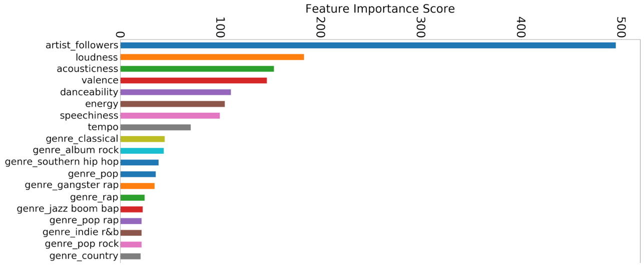
C. Feature Analysis

The primary contribution and most interesting aspect of our models is their high degree of interpretability.

For example, even in the linear regression baseline, we can extract information about how the model makes predictions. The learned coefficient weights provide a crude approximation for the importance of each feature. This is especially insightful for examining our audio features. For example, the audio feature most strongly predictive of popularity is ’danceability’, with a weight of 7.52. Meanwhile, features such as ’speechiness’ are strongly indicative of unpopularity, with a weight of -6.83.

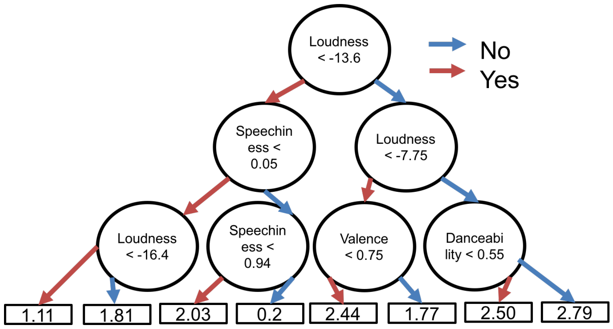
In addition, since both random forests and gradient boosting machines use decision trees, both are highly interpretable. We can examine which features are most responsible for prediction, and how various splits in the model contribute to overall performance.

First, we can examine which features contribute most significantly to the final model prediction:



As expected, the number of artist followers is at least twice as significant as any other single feature in the prediction. The next most important features are audio features such as loudness, acousticness, and valence. Finally, some genres are also fairly indicative of song popularity. ”Classical” is the genre that is most telling of popularity, followed by ”album rock,” ”southern hip hop,” and ”pop.”

Furthermore, we can examine individual decision trees to understand *how* each feature affects the final decision. Below is one such example of a tree:



Recall that the gradient boosting regressor makes its final prediction by taking a weighted sum of its trees' predictions. This particular tree's contribution is a score of anywhere between 0.2 and 2.79. By looking at the individual decision trees, we can get an understanding of feature value cutoffs and how certain features relate to others. For example, in the tree above, 'speechiness' is only a relevant factor for predictions where the 'loudness' score is less than -13.6.

Decision trees unveil dynamics of and between different features in a highly interpretable manner. Our models have revealed the audio features that are most salient in song popularity prediction, as well as the genres and genre combinations that achieve the greatest success.

VI. Error Analysis

We ran three experiments to anayze potential causes of error in our model. In each of these experiments, we analyze the performance of our two best models: the Random Forest and Gradient Boosting Regressor models.

A. *Eliminating Outliers*

In our data exploration section, we observed that the popularity distribution is bi-modal: an approximately normal distribution centers at popularity 40, but there is a large spike of songs with 0 popularity. Looking at the distribution of our predictions, we can see see that our model fails to predict the secondary, large cluster of songs with zero popularity. In general, our model failed to accurately predict extremely popular or unpopular songs.

To test the impact of these outliers on our final accuracy, we trained and evaluated the Random Forest and Gradient Boosting Regressor models on our dataset, but with all songs of 0 popularity removed. This removes 273 songs from our dataset, amounting to 2.3% of our dataset. Unfortunately, removing this mode from our dataset fails to dramatically improve performance. Our Random Forest model evaluates to an MSE of 140.01, and our Gradient Boosting Regressor evaluates to an MSE of 143.80. This lack of improvement may be due to the small number of songs clustered at popularity 0, relative to the size of the entire dataset. It may also indicate that our model still struggles to predict songs in the middle of the popularity range, as well as those at the extremes.

B. *Single Genre Classification*

We hypothesized that another possible contributor to error is the different popularities associated with features in different genres. For example, the negative coefficient of 'speechiness' in our linear regression model indicates that 'speechiness' is associated with lower popularity. In genres like rap, however, 'speechiness' may imply greater popularity.

To test this hypothesis, we evaluated our models using only songs from a single genre. We evaluated our models solely on music tagged under the 'rap' genre, since this genre is one of the most popular genres in our dataset yet remains different from other modern genres (as described above). We have $1,763$ songs in our database tagged as 'rap', which amounts to 15% of our dataset. With this smaller dataset, the evaluated Mean Squared Error is highly dependent on our train-test split. We average across five different random train-test splits, findinh an average MSE of 151.60 for the Random Forest Regressor, and an average MSE of 167.46 for our Gradient Boosting Regressor. While these are significantly higher than our results on the full dataset, the increased MSE is likely due to the smaller size of the dataset when limiting to one genre. Since we have simplified the problem by limiting the data to one genre, yet simultaneously toughened the problem by reducing the dataset size, the true impact of limiting the problem to one genre remains unclear.

C.

VII. Next Steps – TODO: REPLACE

A. *Data Processing*

First and foremost, we would like to extract more songs from the Spotify database to bolster our dataset. This should be straightforward to do, by increasing the number of albums we draw songs from, though calls to the API our rather slow. We would also like to use k-fold cross-validation in the future, which will allow us to be more robust to differing train-test splits and give us leverage to tune the hyperparameters and measure consistent performance.

B. *More Feature Engineering*

As mentioned at the beginning, we will soon incorporate low-level audio features from Spotify's "Audio Analysis" tool, as well as artist metadata from the Spotify API. We will also explore more features from the Spotify database such as song age, location, audio sections, and signature.

We may further try to construct more advanced features from historical data such as musical trends. Using Spotify's provided links to song previews, we may be able to extract more advanced features from the audio recordings of each song. This step will involve a lot more feature tuning and experimentation.

C. *Tuning Models and Analyzing results*

For our existing models, we plan to tune hyperparameters via grid search on our development set to improve performance.

We also intend to analyze the features and their contributions to the model predictions. We would also like to compare and contrast what features different models decided were most important (and in what ways), and see if we can further tune our best-performing models based on this analysis.

REFERENCES

[1] Myra Interiano, Kamyar Kazemi, Lijia Wang, Jienian Yang, Zhaoxia Yu, Natalia L. Komarova, *Musical trends and predictability of success in contemporary songs in and out of the top charts* Royal Society Open Sciences, 2006.

[2] James Pham, Edrick Kyauk, Edwin Park, *Predicting Song Popularity* Stanford University CS 229, 2015.

[3] `https://developer.spotify.com/documentation/web-api/`

[4] Li-Chia Yang, Szu-Yu Chou, Jen-Yu Liu, Yi-Hsuan Yang, Yi-An Chen, *REVISITING THE PROBLEM OF AUDIO-BASED HIT SONG PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS* Academia Sinica, Taiwan, 2017.