Predicting 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 analysing 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

The goal of this project is 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 progress report, we would like to expand on (1) our overall strategy for dataset and feature extraction (2) details about our regression algorithms (3) next steps we plan to take to finish up our project.

II. Methods

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.

Note to the reader: Our original sampling of songs, described in the project proposal, turned out to be highly imbalanced, skewing heavily towards popular songs over unpopular songs. To remedy this, we changed our data sample from the top 10,000 songs to all songs from the top 1,000 albums. This results in a much more balanced dataset. We've re-run our original baseline tests on our new dataset, and we report these figures throughout the paper. Since our data is more evenly spread from 0 to 100, this results in a higher MSE than on our previous dataset.

B. Concrete Example

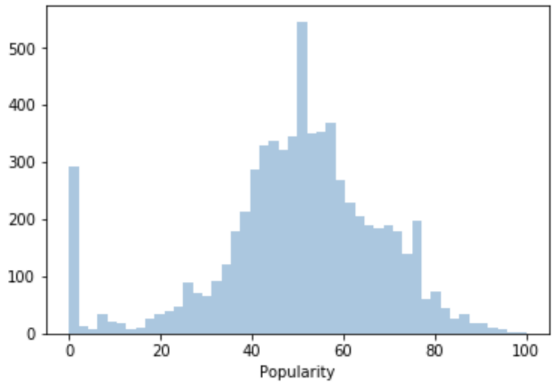
For each song in our dataset, we model its features as extracted from the Spotify and Echo Nest APIs. Here's a sample representation for the song *Never Gonna Give You Up*:

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

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.

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.

Spotify is somewhat slow to deliver audio analysis statistics, as making a call to audio analysis requires Spotify to parse each song. We plan to make these API calls in our future feature extractors.

2) The Echo Nest: 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 propietary, and we are unable to access how they extracted these features, but we use them in all experiments described below.

3) Artist Metadata: We plan to further utilize song and artist metadata such as an artist's past releases, genre, and length of career. Spotify's API yields metrics for an artist's genre and current number of followers. We may also use the powerful feature of Spotify artist popularity; however, Spotify computes this feature using the popularity of an artist's individual songs, so this may not be suitable. [**Question**: Is it fair to use the current popularity of an artist to predict the popularity of one of their songs?]

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, split our data according to a 85/15/15 train/dev/test split, and trained the models described below.

F. Support Vector Regression

As our first algorithm, we chose to to leverage 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. We feed our baseline features and additional features into our SVR, with results reported below.

G. Random Forest

For our second classical algorithm, we will train a Random Forest Regressor on our features. Random Forest Regressors are extremely similar to Random Forest Classifiers, also using a forest of decision trees. In our case, the Random Forest classifier should give us a reasonably strong and interpretable model for use on our features.

III. Initial Results

For our initial evaluations, we ran the following models on our dataset with the basic features described above. We report our models and results below.

A. Linear Regression (Baseline)

For our baseline, we trained a simple linear regression model on our dataset. Solely using the seven features from The Echo Nest, our linear regression model achieved a mean squared error of 264.54.

Worth noting in this linear regression baseline are the learned coefficient weights, which provide a crude approximation for the importance of each factor. The factor most strongly predictive of popularity is 'danceability', with a weight of 21.10. The factor most strongly predictive of unpopularity is 'valence', with a weight of -9.75.

B. SVR

As described above, we then trained a Support Vector Regression model on our dataset. Our SVR used a radial basis function (rbf) kernel, with regularization parameter $C = 10$. With these settings and our basic features, we achieve a mean squared error of 282.99, which is worse performance than our linear regression baseline. In the future, we plan to experiment with various kernels and hyperparameter configurations to optimize the SVR model.

C. Random Forest Regression

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.

Random forests are a class of ensemble algorithms that output the mean result 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 on our dataset, using a forest with 100 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. This resulted in a mean squared error of 248.95, which is a significant improvement over both linear regression and support vector machines.

D. Gradient Boosting Decision Trees

Finally, our best-performing model was a 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 initialized our gradient boosting regressor with 100 trees, just as we did with our Random Forest model, as well as a learning rate of 0.1. This yielded a mean squared error of 243.69, which is 2.1% better than the random forest model, 8.6% better than the initial linear regression model, and 16.2% better than the support vector regression model.

TABLE I
SUMMARY OF RESULTS

Model	LR	SVR	RF	GBR
MSE	264.54	282.99	248.95	242.69

IV. 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.

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.

V. Next Steps

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.

VI. Challenges

One challenge is to consider the effects of the age of a song on its popularity. Concretely, songs that are popular and released in 2018 likely sound very different from songs that are popular and released a decade ago, and furthermore, new songs that sound like popular old songs may not be very successful today. Right now, we get around this problem by only building our dataset from 2018 songs. We may be able to expand our dataset and tackle this problem by simply adding the age of the song as a additional feature, and we intend to investigate.

Furthermore, extracting additional low-level auditory features may be quite challenging. We intend to look into different API's we can use to help us extract useful information, but it is quite likely that most the useful information we would gain from low-level audio features are already represented in and accounted for by the high-level EchoNest features.

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.