Predicting Hit Songs Using Repeated Chorus

ABSTRACT

Songwriters and producers have long believed writing a successful hook is what makes a song popular. In this work, we examined this myth by creating a data set of choruses from popular artists and applied five supervised Machine Learning techniques to predict the popularity purely based on the audio features extracted from a 15s chorus. Using a Neural Network, we were able to demonstrate the performance of the model that is better than random. Hence, we conclude the myth is plausible.

1. INTRODUCTION

A music hook is simply the part of the song that catches the ear of the listener. It can be a riff in the song or just a distinct sound, but more often than not it is the first few lines of the repeated chorus of a song. Creating a hook is a commonly used technique in songwriting. The best hooks will be stuck in your head for days on end. Generally, songwriters and producers believe writing a successful hook is what makes a song popular.

In this work, we examined this myth by creating a data set of choruses from popular artists and applied supervised Machine Learning (ML) techniques to predict the popularity of their works purely based on the audio features extracted from the chorus.

2. RELATED WORK

During our survey of the literature, we have found many attempts of using ML techniques to predict the popularity of songs. We can roughly gauge the body of literature by the extent of *metadata* they rely on for the prediction. Here we define *data* of a song as in the audio and lyric features and *metadata* as in any other features that are not data, such as album information, artist information, etc.

Dhanaraj, R., et al. are one of the purists of song data in this sense. In their early 2005 work[5], they extracted Mel-frequency Cepstral Coefficients (MFCC) as the audio features and embedded lyric features by Probabilistic Latent Semantic Analysis and then used a linear Support Vector Machine (SVM) and boosting trees to predict whether or not a song can hit a chart. Some recent works in the same vine include Li-Chia and et al. 's 2017 paper[6] on also using MFCC features but with a Convolutional Neural Networks (CNN) to assign a popularity score of Chinese and Western songs.

On the other end of the spectrum, however, a few previous works solely rely on metadata to predict the popularity. A notable example is the 2016 paper[10] from Eva and et al., where they tried to predict a hit song by metadata retrieved from Twitter hashtags.

Many others [1][7][8][9] took a hybrid approach that combined information from both data and metadata. For instance, Pachet and Roy[7] collected 49 audio features, including spectral characteristics (Spectral Centroid, MCFF), temporal characteristics (ZCR), and harmonic characteristics (Chroma), as well as adding a number of metadata features. For modeling, this work used Radial Basis Function (RBF) SVM and concludes that predicting song popularity is “not yet a science”. Or recently in the 2019 work[1] developed in this course, Kai and et al. included 27 features from Spotify and used five ML algorithms for predicting whether or not a song can reach the Billboard charts.

To fully examine the myth, this work pushes the boundary of this spectrum, where only audio features from pieces of the audio (the chorus) are used for prediction.

3. DATASET AND FEATURES

To the best of our knowledge, there is no publicly available dataset of hooks of songs. Therefore, we have to build the following data pipeline to prepare the data and features.

1. Collect the names of popular songs and unpopular songs from the same artists the from Billboard.com using *billboard.py*[11]
2. Download the full songs from Youtube using *youtube-dl* [12]
3. Extract the repeated chorus using *pychorus* [2][13]
4. Extract the audio features of the hooks using *librosa*[14]

3.1 Definition of popular and unpopular Songs

We first collected the names of popular songs and artists from the Billboard HOT 100 chart.[15] The chart is a music industry standard record list in the United States for songs published weekly by Billboard magazine. Billboard also publishes month-end and year-end charts based on a song's weighted performance [16] in the weekly chart.

In this analysis, we define a song track as popular if it reaches the chart and labels them as 1. Additionally, we looked into the entire discography of the same artists and label their other works as unpopular or 0.

3.2 chorus and Audio feature Engineering

With the names of popular artists and their popular and unpopular songs, we fetch the full audio files from Youtube using *youtube-dl*. For songs with multiple versions, such as remakes or remixes, we choose the first published studio album version.

We then use the *pychorus* to extract 15 seconds of repeated chorus from each song track. The basic idea of *pychorus* is to extract similar structures from the frequency spectrum. Essentially, it is a form of unsupervised learning and the performance of this extraction is subject to interpretation. But in this work, we assume they are the ground truth. Empirically, we inspected a few familiar songs and the chorus extracted did match our intuition.

We extracted several audio features from the chorus using *librosa*. We selected 11 major spectral features for the analysis. Table 1 shows their name, description, and the number of dimensions.

The dimensions of the feature set for each track are also a function of t_i, where t_i isa function of the duration of the chorus (in this case 15s) and the sampling rate of the soundtrack i. Since we do not want the duration or sampling rate to affect the prediction, we transformed the raw dimensions into 7 statistics, which are min, mean, median, max, std, skew, kurtosis. In total, each song track produces 518 audio features after the transformation.

Table 1 Audio features details

Major features	Descriptions	Raw Dims	Transformed Dims
chroma_stft	Compute a chromagram from a waveform or power spectrogram.	$12 \times t_i$	84
chroma_cqt	Constant-Q chromagram	$12 \times t_i$	84
chroma_cens	Computes the chroma variant "Chroma Energy Normalized" (CENS)	$12 \times t_i$	84
mfcc	Mel-frequency cepstral coefficients (MFCCs)	$20 \times t_i$	140
rms	Compute root-mean-square (RMS) value for each frame, either from the audio samples y or from a spectrogram S.	$1 \times t_i$	7
spectral_centroid	Compute the spectral centroid.	$1 \times t_i$	7
spectral_bandwidth	Compute p th-order spectral bandwidth.	$1 \times t_i$	7
spectral_contrast	Compute spectral contrast	$7 \times t_i$	49
spectral_rolloff	Compute roll-off frequency	$1 \times t_i$	7
tonnetz	Computes the tonal centroid features (tonnetz)	$6 \times t_i$	42
zero_crossing_rate	Compute the zero-crossing rate of an audio time series.	$1 \times t_i$	7
Total		$74 \times t_i$	**518**

3. The final project dataset

As one can see, the data collection and feature engineering process for the project requires a non-trivial amount of work. Therefore, to avoid spending too much time and effort on data collection, we create a small dataset as a proof-of-concept for the final project report.

In this dataset, we picked the top seven artists with the highest number of hit songs on the HOT-100 quarter-end chart between 2006 and 2020. For unpopular songs, we collected other songs from the

same artists and the same albums. Therefore, the dataset in this final report has a 554×518 audio feature matrix and a 554×1response vector.

4. METHODS

4.1 Data exploration and dimension reduction

To build up the intuition and seek to reduce the dimensions, we used two steps to examine the ultra-high dimensional data, a pair scatter plot and the Principal Component Analysis (PCA).

Pair Scatter Plot Figure 1 shows the pair plot of the median values of all major audio features. As one can see from the graph, the signal to noise ratio is quite high as there is no clear separation between any pair of the audio features between the response class. In addition, some pairs show linear correlation, which opens up the possibility to use PCA to reduce the dimension in the next section.

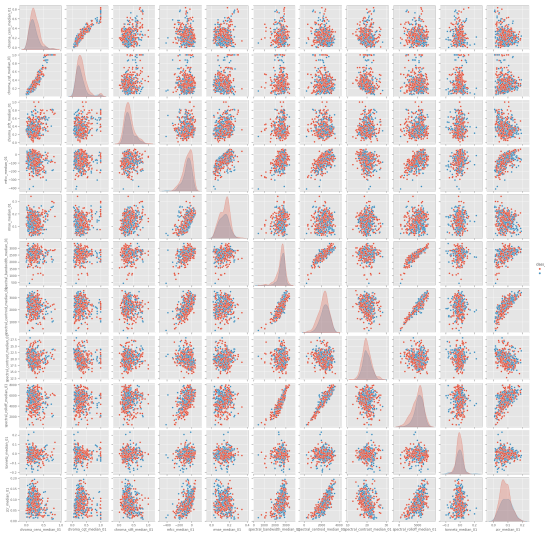

Figure 1. Pair plot of the median value of all major audio features

PCA To understand the intrinsic dimensionality, Figure 2 shows the cumulative explained variance of PCA. It takes 142 Principal Components (PC) to explain 95% of the variance. Therefore, given the large number of features and limited data size, we decided to apply the PCA projection first to reduce the dimensionality.

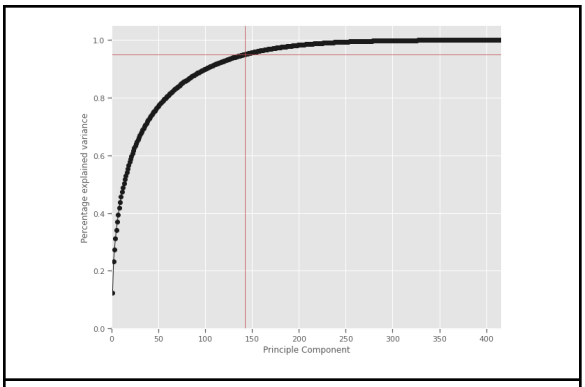

Figure 2. cumulative explained variance of PCA.

Figure 3 shows the biplot of the top two PCs. Similar to the previous pair plot, there is no clear clustering within the first two PCs. We have also visually examined other PCs and none have shown clearly clustering, which further confirms the challenging nature of this problem.

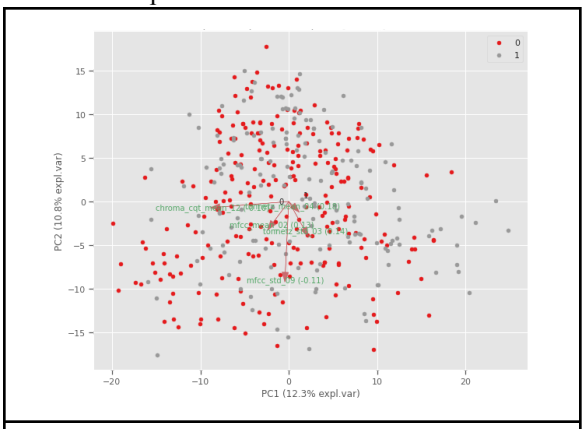

Figure 3. Biplot of the top two principal components. The loadings plot shows how strongly each characteristic influences a principal component.

4.2 Algorithms and models

This project is an exemplary case of bias and variance trade off. On one hand, the number of observations is very limited in comparison with the number of features. On the other hand, however, the problem is immensely complicated without a clear boundary for classification as we discussed in the previous section. Therefore, the key focus is to use regularization in conjunction with the ML models to avoid overfitting. To fine tune the hyperparameters from the regularization term or the models, we used 5-fold Cross Validation (CV) for all candidate learning algorithms.

In this supervised classification setup, we used six different ML algorithms: Logistics Regression (LR), Linear Discriminant Analysis (LDA), SVM, Random Forest (RF), Gradient Boosting Machine (GBM), and NN.

LR with Elastic Net LR is a linear model that uses a logistic function to model a binary dependent variable. Elastic Net is a regularization method that linearly combines the L1 and L2 penalties, which introduces the hyperparameterα, representing the ratio of the L1 penalty.

$$\sum_{i=1}^{p}((1-\alpha)\beta_j^2+\alpha|\beta_j|)$$

LDA with shrinkage LDA is also a linear model but makes stronger assumptions on the underlying densities (Gaussian). The shrinkage method allows one toshrink the separate covariance of LDA toward scalar covariance, which introduces hyperparameter α, representing the shrinkage factor.

$$\hat{\Sigma}(\alpha)=\hat{\Sigma}(\alpha)+(1-\alpha)\hat{\Sigma}\hat{\sigma}^2 I$$

SVM with kernel tricks SVM is a generalized classification method in finding optimal separating hyperplanes. The kernel trick helps produce nonlinear boundaries by constructing a linear boundary in a large transformed version of the feature space. In this work, we used three different types of kernels, linear kernel, Radial Basis Function (RBF) kernel, and polynomial kernel. We also used elastic net regularization to control the complicity of the model.

RF Besides using regularization terms, we also explored a few ensemble learning techniques to control the complexity of the model and avoid overfitting. Ensemble learning builds a committee of weak learners and then combines them to form a composite predictor. RF uses de-correlated decision trees as the weak learner and predicts based on the average of the committees.

GBM Boosting is another popular ensemble learning technique. Unlike RF, the committee of weak learners in boosting evolves and the members cast weighted votes to make predictions.

NN with regularization NN is probably the most popular model developed in modern machine learning. Its key idea is to extract linear combinations of the inputs as new features and then output as a nonlinear function of these features to the next layer. Over time, a few regularization techniques were devised for NN. The most explicit method is to use weight decay, which is to add a penalty to the error function (like ridge regression). The hyperparameter λ represents the shrinkage factor.

$$R(\theta)+\lambda\Big(\sum_{k,m}\beta_{km}^2+\sum_{k,l}\alpha_{ml}^2\Big)$$

5. EVALUATIONS AND DISCUSSIONS

5.1 Results

For evaluation, we split the datainto the training and test set using a 72/25 split. Since we do not have a specific application, we evaluated four metrics (f1, accuracy, precision, and recall) for the CV during hyperparameter tuning during training and for model comparison duration testing and model comparison in the next section. The score functions are all classically defined as follows.

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

$$precision = \frac{TP}{TP+FP}$$

$$recall = \frac{TP}{TP+FN}$$

$$Fscore = \frac{2}{recall^{-1}+precision^{-1}}$$

where TP =True positive; FP =False positive; TN =True negative; FN =False negative.

Overall, despite the best efforts, all models perform worse than we expected. In a close exam, NN is the best model for this task. The Accuracy, Precision, and recall are all better than 50% on the test set, meaning the model is better than random guess.

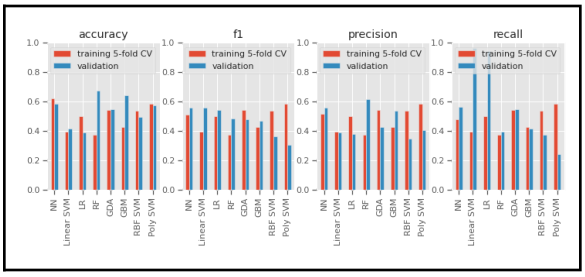

Figure 4 Analysis results of the ML models

All other models also yield similar scores on the test set as on the 5-fold CV, indicating no overfitting. Despite the limited data set, our strategy on focus regularization worked.

Table 2 Analysis results of the ML models								
	Accuracy		F1		Precision		Recall	
	CV	Test	CV	Test	CV	Test	CV	Test
NN	0.62	0.58	0.51	0.56	0.51	0.56	0.48	0.56
Linear SVM	0.39	0.42	0.39	0.56	0.39	0.39	0.39	0.96
LR	0.50	0.39	0.50	0.54	0.50	0.38	0.50	0.94
RF	0.37	0.68	0.37	0.48	0.37	0.62	0.37	0.40
LDA	0.54	0.55	0.54	0.48	0.54	0.43	0.54	0.55
GBM	0.43	0.64	0.43	0.47	0.43	0.54	0.43	0.42
RBF SVM	0.54	0.50	0.54	0.36	0.54	0.35	0.54	0.38
Poly SVM	0.58	0.58	0.58	0.31	0.58	0.41	0.58	0.25

5.2 Neural network deep dive and error analysis

We performed extensive tuning on all models, but due to the page limit, we will only dive deep into the most successful algorithm, NN, as an example.

We implemented the NN in Keras. We choose Karas for its better performance over Sklearn and ease of use over TensorFlow and PyTorch. Firstly, we experimented with the architectures of the Neural Network: single layer, two layers, two layers, and two dropout layers to simplify the estimators, three hidden layers, etc. We settled on using two hidden layers with dropout as the choice. We then fine-tune the size of each hidden layer and the regularization parameters in the model.

For error analysis, we performed permutation tests to inspect the feature importance test and build up the intuition. The weights of the top 1 most important are quite low, which match our observations in the previous sections.

Figure 5 Error analysis for NN via feature importance test		
Importance	Weight	Feature
1	0.0198 ± 0.0116	mean_rmse.2
2	0.0198 ± 0.0103	median_rmse.3
3	0.0188 ± 0.0131	max_rmse.1
4	0.0183 ± 0.0116	kurtosis_mfcc.2
5	0.0149 ± 0.0056	median_mfcc.62
6	0.0149 ± 0.0103	kurtosis_mfcc.15
7	0.0145 ± 0.0081	std_chroma_stft.72
8	0.0145 ± 0.0053	min_chroma_cqt.54
9	0.0140 ± 0.0083	kurtosis_mfcc.5
10	0.0135 ± 0.0099	kurtosis_chroma_cens.1

1.2 Computational efficiency

Beside the challenges of controlling the complicity of models, our limited computation resources posed an unexpected challenge. Our hardware setup is an Intel 6th Generation Intel Core i7-6770HQ with 16GB RAM in an Intel NUC form factor without GPUs. For the milestone report, we created tiny data (154×518) and trained all of the models using sklearn. To our surprise, when moved to a slightly larger data set of 554 observations, all algorithms but LDA failed to scale up.

For LR and SVM, we had to switch from the analytics solver LogisticRegression() and SVC() to SGDClassifier(), a stochastic gradient descent solver, which speeds up the computation an order of magnitude and allows for a grid search for the hyperparameters in minutes.

For RF and GBM, we had to go one step further to use the HalvingRandomSearchCV() estimator for the hyperparameters tuning. The routine implements the Successive Halving (SH) concept, which enables a tournament-like elimination method. SH started with a large number of parameters but on a small scale of data. In each iteration, it eliminates worst performance candidates and reevaluates the model with additional resources until it gets the one with the best CV score on the full data. This new method greatly speeds up our evaluation of RF and GBM since both share a large number of hyperparameter spaces.

6. CONCLUSION

Since the model performs better than luck, we can conclude that the myth that a good chorus makes a song popular is plausible. Nonetheless, due to the low scores of metrics among all models, we tend to believe that many more likely external factors contribute to the success of a song.

Due to the time and resources limitation, the size of the data set and the complexity of the models are still limited by modern standards. So for the future, we would like to generate more data and explore advanced NN algorithms, such as convolutional neural networks (CNNs).