# What Makes a Song a Top Hit? A Deep Dive into Music

Clay Wang (cw654) and Medhavi Gandhi (mg2252)

**Spotify provides our dataset through the Spotify Web API and the spotipy Python package. We employ various scalable big data techniques to train various models to identify how different features affect popularity of songs. We explore loss functions such as Quadratic Loss, Ordinal Hinge Loss, and BvSLoss with an underlying Logistic Loss Classification. To limit model complexity, we used the $L2$ regularizer. We also experimented with a general model on all music and an artist-specific model. Our results showed that levels of predictability varied with each artist. For example, our Random Forest Classifier performed exceptionally well when trained for Taylor Swift with 100% accuracy.**

## 1 Preliminary Data Analytics

### 1.1 Background

The purpose of this paper is to explore the various possibilities of training models to predict the popularity of a song based on musical data and features generated by Spotify. Being able to predict song popularity is important in many ways. For example, artists would be able to determine various features that are important to success in terms of releasing songs, which could help improve many artists' careers. Furthermore, we can also analyze how the importance of various features might change throughout the years. The characteristics of top hits from the early 2000s definitely differ from the current top hits, and it would be very interesting to explore what the change in music taste visually looks like.

In fact, currently there is a lot of research into machine learning being applied to the raw music itself (Widmer, 2000). Using Machine Learning Algorithms, one can generate various features derived from this music, which is what Spotify has provided in its Web API. Furthermore, previous research has indicated that lyrics may influence the general consensus on a song (Fried, 1996). Lot of analysis has been done regarding characteristics of top performing songs and yearly trends, for instance noting that top songs are becoming less and less happy (Interiano et al., 2018). In addition, there are various algorithms that are being used to predict whether or not a specific person will like a song (Pichl, Zangerle, and Specht, 2014).

| Feature Name | Value Type |
|---|---|
| duration_ms | int |
| key | int |
| mode | int |
| acousticness | float |
| danceability | float |
| energy | float |
| instrumentalness | float |
| loudness | float |
| speechiness | float |
| valence | float |
| tempo | float |
| release_year | int |
| artist | string |
| popularity | int |

**Figure 1:** *Raw Features from Spotify*

One of these algorithms is very well known to people, the "Discover Weekly" playlist, which employs a variant of K-NearestNeighbors to generate a weekly playlist for users.

## 1.2 Data Description

Our musical dataset and features were obtained through the Spotify Web API. The Web API supports querying for music by playlist, artist, and track ID. With a track ID, a user can also use the audio-features endpoint to retrieve the features associated with the song. Figure 1 is illustrates the raw features from the endpoint used in our model.

## 1.3 Data Selection

In our paper, we explore a variety of different datasets for different styles of models. In our first model, we initially explore one general model used to predict the popularity of all songs. We further iterate on this model by training different models per year, as well as different models per artist. The motivation for this is to provide more specific models such that we can ignore more noise, and to avoid under-fitting our model. In our general model, we have a dataset of 10,000 songs that represent a plethora of music from over the years. In our yearly model, we split our dataset of 10,000 songs by year, such that we have different dataframes for each year. In our artist model, we consider all songs from Drake, Taylor Swift, and Ariana Grande. Songs are scraped from various playlists on Spotify, as well as querying for all albums from artists and querying all tracks from all artists for the specified artists.

## 1.4 Feature Transformations

### 1.4.1 Categorical Variables as One Hot Encoded Vectors

Lots of the features retrieved from the Spotify Web API are already normalized for us. For example, danceability is already a float from 0 to 1. However, we also have some data that would benefit us in other formats. For example, release_year might be better used as a categorical variable, such that we are able to develop a non-linear model based off the feature. Furthermore, because we are primarily dealing with songs from the 2000s, we would expect to only have 19 vector columns to represent this feature, which is very scalable. Another feature that is categorical is artist, as we can draw a conclusion that there are certain artists that are already quite famous, and any song that they produce be more likely to be popular (i.e. Drake). We express these categorical variables using one-hot encoding, where a vector column exists for each categorical type.

### 1.4.2 Polynomial Transformations

In addition, as we iterate on our models, we will incorporate polynomial feature transformations to explore non-linearity trends in our data with our linear models. We achieve this by generating a new feature matrix which consists of all polynomial combinations of features with degree less than or equal to the specified degree. For example, if we have two vector columns $x_1$ and $x_2$, then all possible polynomial combinations would be

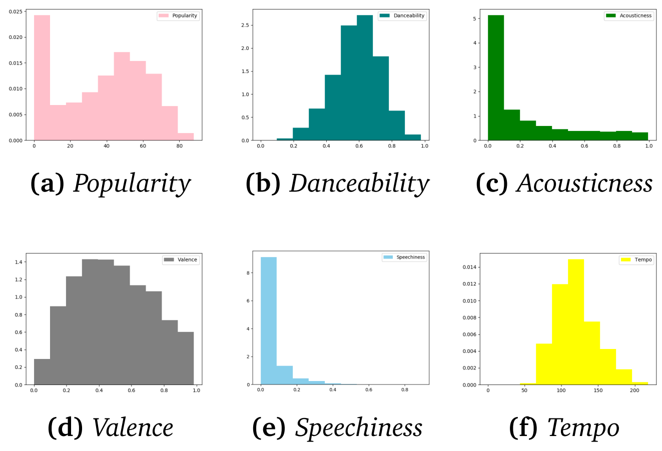

**Figure 2:** *Distributions of features*

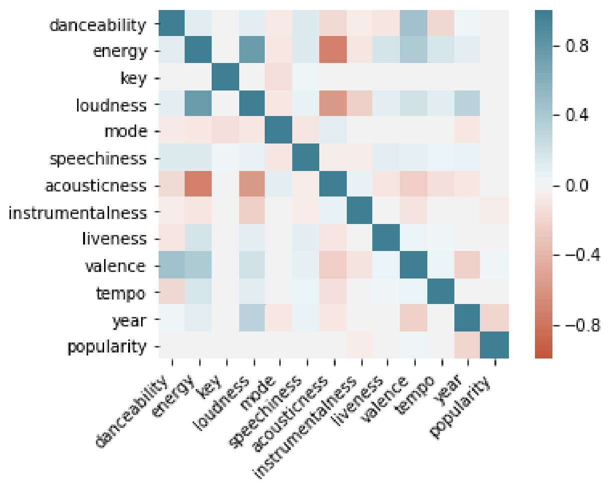

**Figure 3:** *Correlation Heatmap*

$1, x_1, x_2, x_1x_2, x_1^2, x_2^2$. As the degree of polynomial transformation increases, we can see that the number of features will grow exponentially, so the degree must be carefully chosen such that model training will terminate in a sufficient amount of time. Moreover, increasing the degree has risk of over-fitting, further reinforcing the sensitivity of degree.

#### 1.4.3 Derived Features using Principal Component Analysis

From our current set of features, we can derive various features from them. For example, we can convert some of the features into quantiles, and we can also take a linear combination of various vector columns. In particular, what we will be exploring is Principal Component Analysis, a method of summarizing our data in less columns. Principal Component Analysis helps to convert our observed variables into a set of linearly uncorrelated components, which will most likely help as an additional column for our feature matrix.

### 1.5 Data Visualization

Data visualization helps us explore our data visually such that we can derive some meaningful insight from the data. We start by providing some distributions from our data set.

Figure 2 illustrates some of the distributions of the various features we are considering, as well as the distribution of the popularity label that we are trying to predict. Figure 3 illustrates how the various features we have correlate with each other. In particular, some things that we see that are expected, such as danceability and energy having some correlation. Furthermore, it's interesting to see that accousticness and energy also have a negative correlation. However, one thing we see that is that none of these variables have significant correlation with popularity. This implies that no single feature alone would help us predict popularity, but perhaps a combination of the features will help us.

## 2 Model Selection

### 2.1 Model Pipeline

Our popularity label is an ordinal value from 1-100. From this, we derive a new label, that represents if a song is popular is the popularity label $> 50$. The number 50 was chosen to provide a balanced classification dataset. Thus, we convert the dataset into a classification problem, where we try to predict whether or not a song is "above average." We consider the following models.

To prevent overfitting, we use a train-test split of 80% training and 20% test. Furthermore, we also treat our data as a time series of sorts, where all songs in the training set are released before the song in the test set. This is to mimic how we would actually use the model in a live environment. In addition, we also don't want to test too many models, as it's possible that we eventually end up fitting to the test data set.

## 2.2 Logistic Regression

For our initial model, we choose to explore the idea of logistic regression. Logistic regression is a classic method of initially approaching binary classification problems, because the output of the loss function is able to be interpreted probabilistically.
Assuming that $y \in \{-1, 1\}$, then for vector $x$ and weight vector $w$, the loss function of logistic regression is defined as the following:

$$l(y, x, w) = log(1 + exp(-yw^T x))$$

## 2.3 L2 Regularization

One of the problems commonly seen in machine learning is the problem of overfitting. Typically, this happens when too many features are incorporated and the model incorporates random noise into the trained weights, thus making the model too complex. One way of combating this is to use regularization, which applies a penalty on complexity of the model. This penalty is typically added to the loss function as a function of the weights. Therefore, the revised general loss function is

$$l(y, x, w) + r(w)$$

In our project, we will be exploring $L2$ Regularization which is defined as

$$r(w) = \lambda \sum_{i=1}^{n} |w_i|^2$$

Furthermore, we set $\lambda = .01$ for our $L2$ Regularization parameter. We decided against testing various values of $\lambda$, because it might lead to over fitting the parameter on the test-data set.

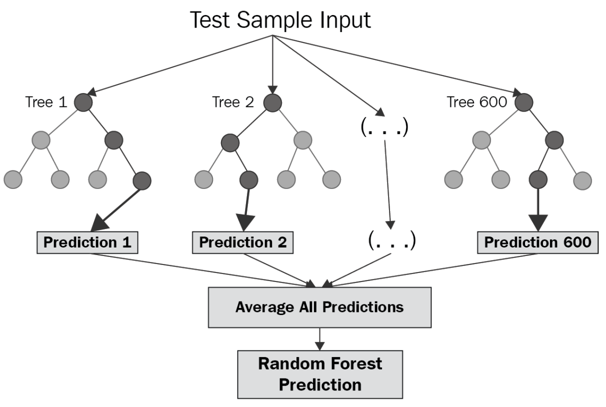

**Figure 4:** *Random Forest Illustration*

## 2.4 Random Forest Classification

We want to further add complexity to our model by exploring random forest. Intuitively, a random forest is a collection of decision trees that output some decision for a set of features from the original dataset. The random forest uses decisions from all the decision trees to make a final classification, typically through majority vote for classifications.

Typically, random forest models are very robust and prevent underfitting due to usage of multiple decision trees. We would expect random forest trees to avoid overfitting because majority vote would offset various overfitted decision trees.

For our parameters, we use 100 decision trees, each with a maximum depth of 9 (square root of our 81 features). By limiting the depth of the decision trees, we are aiming to prevent overfitted decision trees. Because we have overfitted trees, we can use a large number of decision trees to prevent underfitting.

# 3 Model Results

We primarily use the *sklearn* python package for our data transformations and modelling. We train our data on our training split, and to evaluate our model, we observe the accuracy and precision of our binary classifier on our test data split. Our data is split as a time series, with 80% of the songs in the training data, and the rest in the test data.

We define accuracy and precision as the following

- Accuracy is the percentage of correct classifications
- Precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives.

## 3.1 Logistic Regression with $L2$ Regularization

In our initial model, we first explore the usage of a Logistic Regression with $L2$ Regularization. Our objective function is

$$l(y,x,w) = log(1+exp(-yw^Tx)) + \lambda \sum_{i=1}^{n} |w_i|^2$$

### 3.1.1 General Model

In our general model of all songs, we found that our Logistic Regression model was able to classify with an accuracy of 70.15% in the test data set. Furthermore, the precision of the prediction was 74.57%. Therefore, we can see that our model generally does not label an unpopular song as popular.

### 3.1.2 Artist Model

For our artist model, we train models on the songs of Drake, Taylor Swift, and Ariana Grande individually. The performance of the model is illustrated in figure 5. In particular, what we notice is that for Drake and Taylor Swift, the model performs significantly better than the model for Ariana Grande. Most likely, this is due to the amount of data for Drake and Taylor Swift, because they have been in the music industry longer than Ariana Grande has.

| Artist | Accuracy | Precision |
|---|---|---|
| Drake | 77.14% | 76.66% |
| Taylor Swift | 76.92% | 82.35% |
| Ariana Grande | 52.63% | 52.63% |

**Figure 5:** *Artist Model Results using Logistic Regression*

| Artist | Accuracy | Precision |
|---|---|---|
| Drake | 64.28% | 69.49% |
| Taylor Swift | 100% | 100% |
| Ariana Grande | 52.63% | 60.00% |

**Figure 6:** *Artist Model Results using Random Forest Classification*

## 3.2 Random Forest Classification

For our random forest classifier, we are using $100$ decision trees with a max depth of $9$ per decision tree. We use the majority voting scheme to aggregate the result of the decision trees into one final classification.

### 3.2.1 General Model

In our general model of all songs, we found that our Random Forest model was able to classify with an accuracy of 47.01% in the test data set. Furthermore, the precision of the prediction was 78.72%. Therefore, we can see that despite the fact that the accuracy is significantly poorer when compared to logistic regression, our model is even better at mislabeling an unpopular song.

### 3.2.2 Artist Model

We repeat the process illustrated in the Logistic Regression model and train individual models for each artist. The performance of the model is illustrated in figure 6. In particular, the model for Taylor Swift performed exceptionally well, while the other models performed roughly the same as their Logistic Regression counterparts.

## 4 Discussion

### 4.1 Result Overview

In general, we can see that the general model typically does not have great accuracy, but has good precision. This indicates that in a real life scenario, an artist should not take the model's result as true, but that if the model thinks that the song will be popular, then it has a higher chance of being a popular song after release. Thus, the general model is more of an aid for an artist.

Furthermore, we see that for some artists, models trained for their music specifically do not have significant improvement. For example, Ariana Grande's model does not perform significantly better than the general model. This is most likely becuase Ariana Grande's career is relatively young, thus leading to less data for the model to train. On the other hand, Taylor Swift has a very mature musical career, and thus there is a plethora of data to train the model on, and thus the model's trained for her specifically perform significantly better than the general model.

In addition, we noticed that for Taylor Swift, the random forest classifier performed significantly better than for all other artists. This result indicates to us that different artists have a different level of predictability in their music, as some artist's styles change throughout their careers. Thus if a model is trained during a different period of their career, the model might not be applicable for their new songs, and thus a general model would be better.

### 4.2 Future Improvements

Our project as room for many improvements. For example, we could try to find other data sources to generate more features. Our data doesn't include anything about the lyrics of songs, a feature that is generally very important for music listeners. Furthermore, we can test on more artists to experiment with predictability of success for various artists.

Another idea would be to change the threshold for determining the definition of a "popular" song. We set the threshold to 50 to have a balanced dataset for classification, but perhaps different measures could be used. One example would be songs that make it to the Billboard Top 100 playlist. We can also aim to increase our total music data set by trying to combine more playlists. Spotify currently has a 10,000 song limit on playlists, so we were limited to 10,000 songs in our dataset.

We can also apply different models in the future. Two examples of models that would be interesting to test are deep neural networks and gradient boosted trees. In particular, neural networks would potentially lead to finding trends in our data that aren't easily noticeable to linear models.

In the future, a yearly based model can be explored, where a model is trained per year. This would eliminate changes in yearly trends and value songs that are more recent, which typically are more indicative of current trends, and thus won't be influenced by songs from previous generations.

### 4.3 Weapons of Math Destruction

Looking at the factors that determine whether a model is a Weapon of Math Destruction or not:

- Are the outcomes hard to measure?
- Could the predictions harm anyone?
- Could our results create a feedback loop?

Our outcome is a classification, a function of number of plays of a song, and our data has been split into train and test sets, to be able to measure test error.

In terms of negative consequences, our model is intended to be an aid for an artist, not a foolproof test of whether or not a song will be popular, so ideally the chances of it being misused by labels or managers to make artists make a certain kind of music are low. Music- even for one artist- is usually changing and evolving with their creative process, so a feedback loop would most likely not form. Because our model does not include sensitive or protected data, the model most likely will cause no harm to others.

### 4.4 Fairness

Our model doesn't use any protected attributes. Our data is listening data of humans - either measurable on its own based on a song, or derived from something that is, without needing human response or inputs. Our model most likely would not discriminate against, or unfairly affect people.

## 5 Conclusion

By using techniques we learned in class such as one-hot encoding, logistic regression, regularization, PCA, and polynomial transformation, we were able to explore the idea of algorithmically determining whether or not a song would be popular. We tested Logistic Regression and Random Forests for classification of songs, and we found that there was indeed some success in predicting whether or not a song would be popular. In particular, our best results came from a model trained specifically for Taylor Swift, which performed with 100% accuracy on our data test set.

In our results, we saw that Logistic Regression performed better for certain artists, and Random Forest Classification performed better for others. We avoided testing multiple parameters, because we wanted to avoid over-fitting on our test data set.

We are interested to see the effects of applying our model to various artists and to perform more vigorous tests to explore the idea of levels of predictability for songs produced by certain artists.

## Bibliography

Fried, Carrie B. (Dec. 1996). "Bad Rap for Rap: Bias in Reactions to Music Lyrics". In: *Journal of Applied Social Psychology* 12.26, pp. 2135–2146. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1559-1816.1996.tb01791.x`.

Interiano, Myra et al. (May 2018). "Musical trends and predictability of success in contemporary songs in and out of the top charts". In: *Royal Society Open Science* 5.5, pp. 4477–4479. URL: `https://royalsocietypublishing.org/doi/full/10.1098/rsos.171274`.

Pichl, Martin, Eva Zangerle, and Günther Specht (Oct. 2014). "Combining Spotify and Twitter Data for Generating a Recent and Public Dataset for Music Recommendation". In: *Proceedings of the 26th GIWorkshop on Foundations of Databases (Grundlagen von Datenbanken), 21.10.2014 - 24.10.2014, Bozen, Italy, published at http://ceur-ws.org*. Pp. 1–6. URL: `https://dbis.uibk.ac.at/sites/default/files/2017-03/gvdb14.pdf`.

Widmer, Gerhard (2000). "On the Potential of Machine Learning for Music Research". In: *Readings in Music and Artificial Intelligence*. URL: `https://pdfs.semanticscholar.org/8dc8/be5f2fc7456897c426931769e4fad7f5962d.pdf`.