Song Hit Prediction: Predicting Billboard Hits Using Spotify Data

Kai Middlebrook, Kian Sheik

***Abstract*—In this work, we attempt to solve the Hit Song Science problem, which aims to predict which songs will become chart-topping hits. We constructed a dataset with approximately 1.8 million hit and non-hit songs and extracted each songs audio features from the Spotify Web API. We test four models on our dataset. Our best model was random forest, which was able to predict Billboard song success with 88% accuracy.**

***Index Terms*—Machine Learning, Hit Song Science, Classification, Data Mining, Data Collection.**

I. Introduction

HIT song science (HSS) aims to predict whether a given song will become a chart-topping hit. The underlying assumption in HSS is that hit songs are similar with respect to their features. Hit Song Science is an active research topic in Music Information Retrieval (MIR).

Hit prediction is useful to musicians, labels, and music vendors because popular songs generate larger revenues and allow artists to share their message with a broad audience. For example, if a label would like to increase profits, they may choose to invest their limited resource (ad campaigns, studio equipment, etc.) on tracks that are likely to become popular. On the other hand, if an artist wants to embody an aesthetic that is devoid of mainstream musical characteristics, they may choose to release tracks that are unlikely to become popular. We describe our approach to solve the HSS problem in the proceeding sections.

II. Methods

A. Dataset and Features

Previous work on HSS have used relatively small datasets [1]. We extend previous work by creating a larger dataset. We believe this larger dataset will allow for more robust model architectures than previous datasets. We used the Spotify API to create a dataset with approximately 1.8 million songs. We reduced the size of the dataset by considering only the songs released between the years 1985 and 2018. We then collected a dataset of all unique songs on the Billboard Hot 100 chart between 1985 and 2018 using the Billboard API (∼16k songs). Finally, we merged the Spotify and Billboard datasets together by matching tracks on their title and artist. This combined dataset contained approximately 1.8 million tracks, with approximately 12,000 tracks being Billboard Hot 100 hits.

In order to balance our data, we randomly sampled 12,000 non-hits from the Spotify data and created a new dataset. This dataset contained approximately 12k non-hits and 12k hits (∼24k tracks total). We refer to this dataset as SpotifyBillboard.

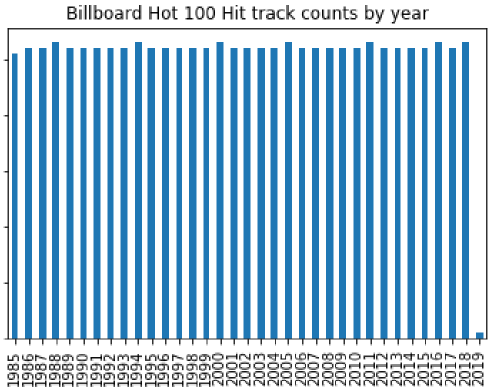


Fig. 1. Billboard track counts by year. Years with few than 5k songs were excluded from our dataset (e.g. 2019).

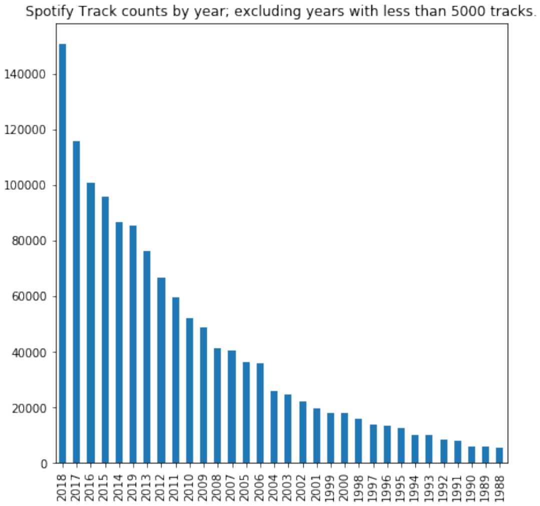


Fig. 2. Spotify track counts by year. Years with few than 5k songs were excluded from our dataset.

Each track contains 27 features categorized by track information, artist information, album information, and audio analysis features. We describe these feature in detail below:

- *track_id*: the song's unique Spotify track ID
- *track_title*: the track title
- *artist_title*: the artist's title
- *artist_id*: the artist's unique Spotify ID
- *popularity*: a value between 0 and 100, with 100 being the most popular. Popularity is calculated by Spotify, and is based, "in the most part, on the total number of plays the track has had and how recent those plays are" [2].
- *explicit*: a value indicated whether a track has explicit

lyrics (1 = explicit, 0 = not explicit)

- *duration_ms*: the duration of the track in milliseconds.
- *preview_url*: a link to a 30 second preview of the track.
- *album_id*: the unique Spotify album ID
- *album_type*: the type of the album: one of album, single, or compilation
- *album_release_date*: the date the album was first released (date format: YYYY-MM-DD)
- *acousticness*: a value from 0.0 to 1.0 predicting whether the track is acoustic.
- *danceability*: a value from 0.0 to 1.0 describing how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. Values closer to 1.0 indicate that the track is more danceable.
- *energy*: a value from 0.0 to 1.0 that represents a perceptual measure of intensity and activity
- *instrumentalness*: a value from 0.0 to 1.0 predicting whether the track is instrumental or contains vocals. Values closer to 1.0 represent more instrumental track.
- *key*: the key the track is in
- *liveness*: a value from 0.0 to 1.0 that describes the presence of an audience in the track. Values closer to 1.0 represent tracks that were performed live.
- *loudness*: the overall loudness of a track in decibels (dB)
- *mode*: indicates the modality (major=1 or minor=0) of a track.
- *speechiness*: a value from 0.0 to 1.0 describing the amount of spoken words present in the track. Values close to 1.0 indicate exclusively speech-like tracks (e.g. podcast, audio book, poetry).
- *tempo*: the overall estimated tempo of a track in beats per minute (BPM)
- *time_signature*: an estimated overall time signature of a track.
- *valence*: a value from 0.0 to 1.0 describing the musical positiveness conveyed by a track. A value close to 1.0 suggests that the track sounds more positive and upbeat.
- *weeks*: a value indicating the total number of weeks the track was on the Billboard Hot 100 chart
- *rank*: a value between 0 and 100 indicating a track's position on the Billboard Hot 100 chart. A value of 0 indicates that the track never appeared on the Billboard Hot 100 chart.
- *score*: a weighted rank value from 0.0 to 1.0 indicating the popularity of a track. The score is a custom made value. It is a weighted value indicating the most popular tracks on the Billboard Hot 100 chart. A value of 0.0 indicates the track never appeared on the chart. A value close to 1.0 indicates that the track appeared frequently at the of the chart. Note, this value was not given to us, we used a data mining method, which was described above, to calculate this value.
- *billboard_hit*: indicates whether the track appeared on the Billboard Hot 100 chart (1=hit, 0=non-hit).

After processing SpotifyBillboard features we created the train, validation, and test sets. No tracks in the training set appeared in the validation and test set. Each set contained a balanced distribution between hits and non-hits (Figure 3). Note, the majority of HSS work has only considered audio analysis features (acousticness, instrumentalness, etc.) [1], [3]. To extend previous work, in addition to audio analysis features, we consider song duration and mine an additional artist past-performance feature. Artist past-performance for a given song represents how many prior Billboard hits the artist has released before that track's release date.

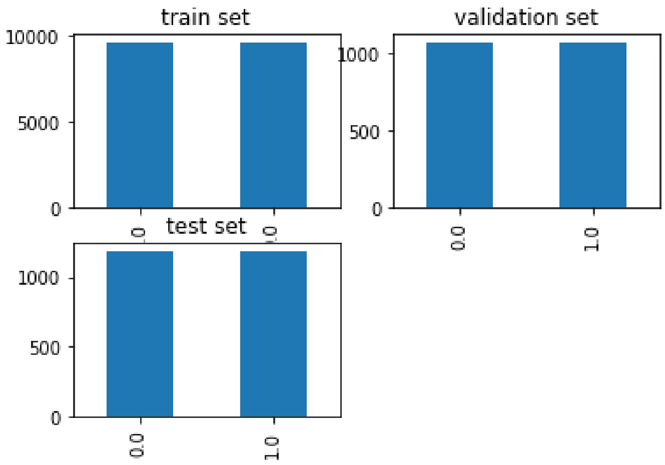


Fig. 3. Distribution of Hits and Non-Hits in the train (20k tracks), validation (2k tracks), and test (2k tracks) sets

B. *Training Environment*

We utilized the entire fleet of computers available to University of San Francisco's Computer Science Department ($\sim$40) in order to run first a randomized search and then a grid search on each of the classification algorithms available in the scikit-learn package. The average training time was about one day and the following models have been reported for notable performance.

C. *Models & Algorithms*

To predict whether a song will be a Billboard hit or not, we use four different models:

- *Logistic Regression (LR)*
- *Neural Network (NN)*
- *Random Forest (RF)*
- *Support Vector Machine (SVM)*

1) Logistic Regression: Logistic regression (LR) is a popular classification algorithm. It is used when the dependent (target) variable is categorical. The idea in LR is to find a relationship between features and the probability of a particular outcome. There are two types LR problems binary logistic regression and multi-class logistic regression. We used binary logistic regression because our dependent variable has two possible values 0 (non-hit) and 1 (hit). We use the sigmoid activation function to constrain our probability estimate between 0 and 1.

$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{1}$$

We use Maximum Likelihood Estimation (MLE) to estimate the feature coefficients and RMSEprop to back-propagate the gradients over 1000 epochs. We define the cost function below.

$$
\begin{aligned}
L(\beta; y) &= \prod_{i=1}^{n} P\big(Y_i = y_i | X_i = x_i\big) \\
&= \prod_{i=1}^{n} \sigma\big(x_i^t \beta\big)^{y_i} \big(1 - \sigma\big(x_i^t \beta\big)\big)^{1-y_i}
\end{aligned} \tag{2}
$$

Where $\sigma\big(x_i^t\beta\big)$ is the probability of a hit and $\big(1-\sigma\big(x_i^t\beta\big)\big)$ is the probability of a non-hit and y_i = 1 (hit) or 0 (non-hit).

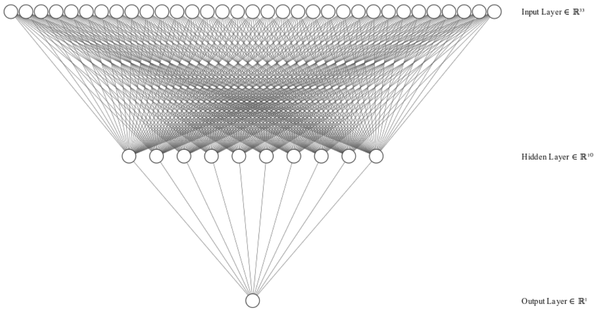


Fig. 4. Diagram of our neural network architecture. We use 1 hidden layer with 10 filters.

2) Neural Network: Neural Networks (NNs) have become popular to solve classification tasks after the rise of deep learning. We use a simple neural network with one hidden layer to solve HSS (Figure 4). We use RMSprop-an unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton [4], and sigmoid function in the final layer to constrain the output between 0 and 1. In the hidden layer, we use ten filters and rectified linear unit (ReLU) activation. We set the batch size to 32 and stopped training after 1000 epochs.

3) Random Forest: Random Forest (RF) models are one of the most popular ensemble methods used in classification. These models aim to correct for the problem of over-fitting in traditional decision trees. This will not be covered in depth, but decision trees tend to learn on irregular paths of data. RF models train multiple deep decision trees on different aspects of the dataset with the aim of reducing the overall variance.

Not only was RF the most accurate model overall, but it was the quickest to train. We used a maximum number of features of eight with 80 estimators and a minimum split condition of two samples under the Gini criterion.

4) Support Vector Machine: Support Vector Machine (SVM) aims to find the most optimal hyper-plane that separates the data into two distinct classes. We used the Gaussian Radial Basis Function (RBF) as our kernel: $\exp(-\gamma\|x - x'\|^2)$. Our model uses $\gamma = 0.1$ and $C = 10$.

III. Results

We focused mainly on the accuracy of results, but we report the precision and recall as well since false positive predictions may be costly when a music label invests in a song that is actually unlikely to become a hit (Table I).

TABLE I
MODEL RESULTS

Models	Accuracy		Precision		Recall	
	Test	Val	Test	Val	Test	Val
Logistic Regression	0.8151	0.8065	0.7526	0.7457	0.9391	0.9298
Neural Network	0.8214	0.8305	0.8235	0.8233	0.7913	0.7671
Random Forest	**0.877**	**0.887**	**0.86**	**0.87**	**0.9**	**0.89**
SVM	0.839	0.828	0.81	0.79	0.89	0.89

TABLE II
NN CONFUSION MATRIX ON THE VALIDATION SET

		Actual	
		Hit	Non-Hit
Predicted	Hit	1027	363
	Non-Hit	42	707

The NN model with one hidden layer gave 82.14% and 83.05% accuracy on the validation and test data, with similar results on the training data indicating no over-fitting. The final cross-entropy loss after 1000 epochs was 0.4261. The precision and recall on the validation set were 82.33% and 76.71%. The confusion matrix on the validation set shows that there are some false positives (Table II).

TABLE III
LR CONFUSION MATRIX ON THE VALIDATION SET

		Actual	
		Hit	Non-Hit
Predicted	Hit	994	339
	Non-Hit	75	731

The LR model yielded 80.65% accuracy on the validation data and 81.51% accuracy on the test data, with similar result on the training data indicating no over-fitting. The precision and recall were acceptable at 74.57% and 92.98%. The confusion matrix on the validation set shows that there are some false positives (Table III).

TABLE IV
RF CONFUSION MATRIX ON THE VALIDATION SET

		Actual	
		Hit	Non-Hit
Predicted	Hit	917	153
	Non-Hit	82	993

The RF model yielded 88.7% accuracy on the validation data and 87.7% accuracy on the test data, with similar result on the training data indicating no over-fitting. The precision and recall were acceptable at 87% and 89%. The confusion matrix on the validation set shows that there are some false positives and false negatives (Table IV).

TABLE V
SVM CONFUSION MATRIX ON THE VALIDATION SET

		Actual	
		Hit	Non-Hit
Predicted	Hit	1065	5
	Non-Hit	447	628

The SVM model yielded 82.8% accuracy on the validation data and 83.9% accuracy on the test data, with similar result on the training data indicating no over-fitting. The precision and recall were acceptable at 79% and 89%. The confusion matrix on the validation set shows that there are some false negatives (Table V).

IV. CONCLUSION & FUTURE WORK

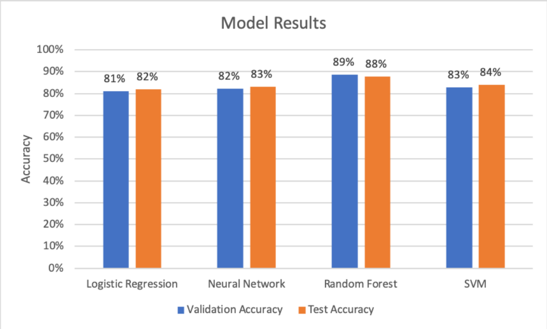


Fig. 5. Model Results on the validation and test sets. The Random Forest model was the most successful

The results showed that SVM and RF outperform LR and NN in regards to accuracy (Figure I). The most robust model is the RF. Interestingly, the SVM had the highest precision accuracy (Table I).

The false positive rate for our SVM is very low, while maintaining an average false negative rate. The truth values predicted by this model can be trusted while the false values cannot. This algorithm is greedy and will assume the least amount of risk when classifying a positive. Music labels may prefer to use the SVM since it is less likely to predict hits incorrectly.

In future experiments we would like to investigate label influence and social media presence with respect to song success. Using features of the audio itself combined with artist past-performance has managed to explain a majority of the variance in the data; we believe there are more types of features which can provide our model with a social context to make even better predictions.

REFERENCES

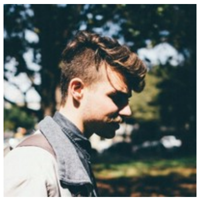
[1] E. Georgieva, M. Suta, and N. Burton, "Hitpredict: Predicting hit songs using spotify data," 2018.

[2] S. for Developers, "Song popularity," 2019.

[3] D. Herremans, D. Martens, and K. Sörensen, "Dance hit song prediction," *Journal of New Music Research*, vol. 43, no. 3, pp. 291–302, 2014.

[4] V. Bushaev, "Understanding rmsprop: faster neural network learning." Referenced on May 10th, 2019, 9 2018.

Kai Middlebrook is a Junior studying at the University of San Francisco (USF). He is pursuing a major in Data Science and a minor in Music. He can be reached at krmiddlebrook@usfca.edu.



Kian Sheik is a Senior at the University of San Francisco (USF). He is pursuing a major in Data Science. After graduation, he will be working full-time as a data engineer at ReferralExchange in San Francisco. He can be reached at kasheik@usfca.edu.