Predicting Song Popularity Using Machine Learning Algorithms

Bihag Dave*, Koushiki Bhattacharyya†, Prayag Savsani‡ and Yashraj Kakkad§
Ahmedabad University
Enrollment No: *AU1949007, †AU2029005, ‡AU1841035, §AU1841036

***Abstract*—Being able to predict popularity of a song based on metadata and attributes can be of great industrial importance. We aim to achieve this using machine learning techniques. We use data obtained from Spotify Web API which contains information of over 160,000 songs from 1921 to 2020. We perform the required pre-processing to test several regression and classification algorithms. Based on obtained results, we build ensemble learning models for classification. Models are tuned to give optimal test results. We infer that tree-based algorithms give competitive results in general. Due to imbalanced classification, the models are able to predict non-popular songs more easily than popular ones, where there's a high number of false negatives.**

***Index Terms*—machine learning, music, popularity, prediction, songs, regression, classification, ensemble learning, random forests, boosting**

I. Introduction

Hundreds of songs are released every year, but very few of them make it to the top charts. Music analysis has made it possible to generate metadata of a song. Similarly, such features are available for artists and genres. We aim to answer the question - "Is it possible to predict the popularity of a song using these attributes in Machine Learning algorithms?". Along with that, we learn more about the features involved, how important (or unimportant) they are and what are the possible limitations.

Our key contributions are the following -

1) We explore a suitable dataset which contains the requisite features and tune it to our needs.
2) We perform Exploratory Data Analysis (EDA) on the data.
3) We apply several machine learning algorithms to predict the popularity of a song. We use both regression, classification and ensemble learning algorithms. Inferences are gained at every step and further work is decided based on that.
4) The model hyperparameters are tuned to fetch the best possible results. Different models are compared.

II. Literature Survey

The ability to predict whether a song will be a hit or not is of commercial importance, due to which there has been a keen interest in using Machine Learning techniques in predicting the popularity of a song. This is often called Hit Song Science, a term coined by Mike McCready. There are several papers that quantify certain lyrical and acoustical characteristics of a song and use those to predict popularity. An early attempt in using such features is by B. Logan et. al [1]. Another paper by R. Nijkamp [2] uses the same Spotify dataset that this project aims to use and uses the attributes provided by Spotify as features to build a linear regression model.

There have also been attempts to predict song popularity from just lyrical properties. One such attempt is by A. Shinghi et. al [3] where they use rhyme, syllable and meter features to predict popularity. They found that lyrical features worked better than audio features in popularity predictions. Another approach to hit song prediction is to ignore intrinsic characteristics of a song altogether (lyrical or acoustical) and use social media metrics of the artist or buzz created by users. Such attempts were made as early as 2008 by Bischoff et. al [4]. In a similar vein E. Zangerle et. al investigates correlations between twitter hashtag #nowplaying and billboard top 100 charts [5].

III. Implementation

A. Introduction to Dataset

Spotify has a public API which serves information about tracks, albums, artists, genres etc. [6] We obtained our dataset from Kaggle which is a collection of information of more than 175,000 songs collected from Spotify Web API. [7] Features such as acousticness, danceability, energy etc. are available for tracks, artists, genres and years.

B. Problem Statement - Song Popularity Prediction

Our aim is to build and test models which can predict the song popularity score (regression) and whether or not the song is popular (classification).

C. Pre-processing Steps

We apply the following pre-processing steps to our data, to suit our analysis better -

1) We remove the features not useful for our analysis such as 'id', 'duration' etc.
2) We convert the categorical variables 'key' and 'mode' using One-Hot Encoding to make them more suitable for machine learning algorithms.
3) Track and artist tables are (left-outer) joined over artist, whose aggregation operations are performed as summarized in Table I. Similarly, track and year tables are also joined.

As a result, we obtain 72 columns which can be used as features.

TABLE I: Aggregate Operations for Joining Track Artist Data

Feature	Operation	Feature	Operation
Acousticness	Mean	Danceability	Mean
Energy	Max	Instrumentalness	Max
Liveness	Max	Loudness	Mean
Speechiness	Mean	Tempo	Mean
Valence	Mean	Popularity	Max
Count	Max	Mode	Max

D. *Exploratory Data Analysis*

We analysed the base data to obtain a correlation matrix between columns, box-plots, and histograms. While a few strong correlations were observed between features - for instance, Energy and Danceability - we found only one strong correlation between a feature and popularity.

The subsequent sections deal with algorithms used. We used scikit-learn, a popular machine learning library in Python [8], for the required tools.

E. *Regression algorithms*

1) Linear Regression and Polynomial Regression: Performing iterative feature selection based on correlation with popularity, we noticed a plateau at around 27 features after which the co-efficient of determination did not increase appreciably and sacrificed computational time.

For polynomial regression, we went up to degree 2, beyond which we were computationally limited, owing to the size of the dataset.

2) Lasso Regression: Lasso Regression has the property of acting has a feature selector. [9] It reduced all our feature weights to zero except for two - artist popularity and year popularity, while giving a reasonable accuracy.

3) Decision Tree Regression: We consider only binary trees for Decision Tree algorithms. The algorithm leads to severe overfitting (99% accuracy on cross validation vs 75% accuracy on test set), and therefore we tune the model hyperparameters, whose optimum values are summarized in Table II.

TABLE II: Hyperparameters of Decision Tree Algorithms

Hyperparameter	Regression	Classification
Maximum depth	10	10
Minimum samples to Split	6	12
Minimum samples required at leaf	7	1

F. *Classification algorithms*

The popularity values between 0-100 is converted to labels based on a decided threshold value. The threshold was determined by looking at the popularity distribution of the songs which appeared in Billboard Hot 100 at least once using their API. The positive class is in minority in comparison to the negative class i.e. the non-popular class. Therefore, we used stratified Kfolds technique instead of normal Kfolds during splitting our dataset and also during cross validation.

In this case, classification accuracy alone cannot be used as a reliable performance metric [9]. Therefore, we use other metrics like precision, recall and F1-score. More on this will be explained in the results section.

1) Support Vector Machines (SVM): During our experimentation, we observed that a non-linear SVM classifier gives better results. We applied the so-called *kernel trick* which is just exploiting math to map the data to a higher dimensional space. We use the default linear kernel during training. It has a regularization parameters called C, which was tuned using grid search technique to give us the best possible recall and F1 score.

2) Decision Tree Classification: We used “Entropy” impurity (versus the conventional “Gini”) as our loss function because it yields slightly better results. The model hyperparameters and their optimum values are summarized in Table II.

3) Perceptron: We tuned the learning rate and the number of training epochs. After a point, increasing the number of training epochs was not improving results. We chose a feasible value which converged. The values of these hyperparameters are given in Table III.

TABLE III: Hyperparameters of Perceptron Classifier

Hyperparameter	Value
Learning rate	1
Number of training epochs	10000

G. *Ensemble learning*

Ensemble learning involves the use of multiple learning algorithms to improve performance. We limit our scope to classification problem using tree-based algorithms, except for voting classifiers where multiple algorithms need to be used.

1) Voting classifiers: Voting classifiers take a few classifier’s decisions to aggregate them and predict a final decision. Often a classifiers’ individual classification is much weaker than a voting classifier’s that is the aggregate of all of them. We use both hard and soft voting classifiers and use all the classifiers with the hyperparameters that we have tuned before. We use pipelines to assemble several steps that can be cross-validated together while setting different parameters for each of the different classifiers.

a) Hard Voting Classifier: We use SVM, Logistic Regressor, Decision Tree and Perceptron as the voters of this classifier. The class with majority votes from all these classifiers are predicted by the voting classifier.

b) Soft Voting Classifier: We use all the classifiers except Perceptron as the voters in this, as Perceptron does not give class probabilities. These voters’ decisions are pipelined into the voting classifier weighted by the predicted class probabilities. This classifier is expected to give better results as the more confident votes are given higher weightage.

2) Random Forests: Random forests are an ensemble of Decision Trees trained via the bagging method. They use the same hyperparameters as Decision Trees and can be used for classification as well as regression. We used Random Forest Classification to train our model. There was marginal change in performance when we used default hyperparameters. We used Randomized Grid Search, which optimizes values for hyperparameters by iterating through random combinations from the parameter grid. We set the Grid Search to go through 100 iterations with 3-fold cross-validation and arrived at a marginal improvement in performance with the optimal hyperparameters. The optimal hyperparatmeters are shown below.

TABLE IV: Hyperparameters of Random Forest Classifier

Hyperparameter	Value
Max depth	20
Minimum samples required at leaf	4
Minimum samples required to split	2
Number of Trees	400
Max features	sqrt

3) Boosting: Boosting is a general term for an Ensemble method which combines several weak learners to a strong learner. [9] We consider the two most popular methods - *AdaBoost* (*Adaptive Boosting*) and *Gradient Boosting*. A common disadvantage in boosting algorithms is that they are sequential, unlike bagging algorithms like Random Forests.

a) AdaBoost: AdaBoost involves the use of consecutive predictors, Decision Stumps (Decision Trees of unit depth) in our case. After every prediction, the weights of training instances are updated such that the instances which the model underfitted get a larger relative weight. The predicted class is the one which receives majority of the votes from all the predictors, considering the prediction probabilities as well. The tuned hyperparameters are summarized in Table V.

TABLE V: Hyperparameters of AdaBoost Classifier

Hyperparameter	Value
Number of estimators	250
Learning rate	0.68

b) Gradient Boosting: Gradient Boosting tries to correct its predecessor by fitting the new predictor to the residual errors made. We used Scikit-learn's new experimental Histogram-based Gradient Boosting Classification (*HistGradientBoostingClassifier*) which is orders of magnitude faster than the conventional classifier [8]. It discretizes the features to ordinal bins which makes decision trees run faster and the cost on performance is minor. The hyperparameters considered are summarized in Table VI. In addition to the number of trees, we also regulate the size by controlling the maximum number of leaf nodes in a tree.

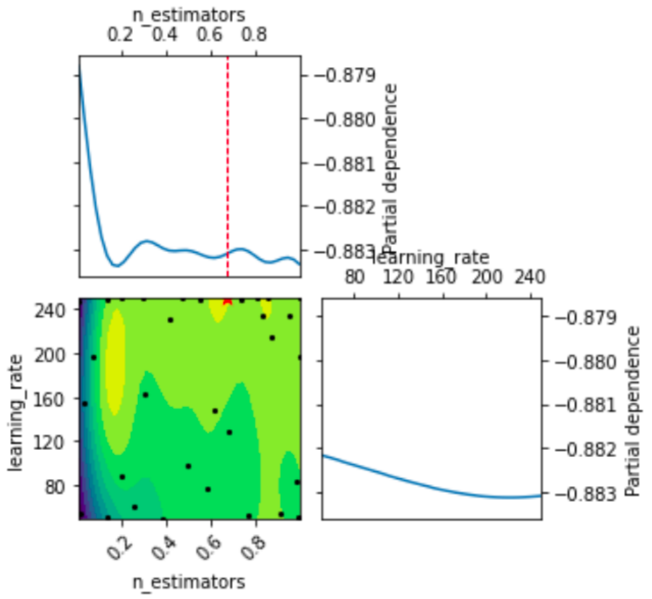
TABLE VI: Hyperparameters of Gradient Boosting Classifier

Hyperparameter	Regression
Maximum number of iterations (estimators)	200
Learning rate	0.06
Maximum number of leaf nodes	100
Minimum samples required at leaf	78

A general observation is that on increasing the size of the tree, a decreased learning rate is obtained while tuning the parameters.

c) Bayesian Optimization in hyperparameter tuning: Boosting algorithms being relatively slower, naive hyperparameter iteration (scikit-learn's *GridSearchCV*) is not feasible option. We use the Bayesian optimization technique for iterating through the hyperparameter space. The technique is helpful when we have an computationally intensive and cost function with noisy evaluations whose closed form solution is not known. This is exactly the case for these algorithms. It does not iterate through the entire space and uses a "surrogate" function to model the same. An acquisition function subsequently chooses the next sample to test the function. [10] We use the implementation from scikit-optimize library [11].

Fig. 1: Bayesian Optimization for AdaBoost Classifier



IV. Results

A. Regression algorithms

Decision Tree yields very competitive results compared to the simpler algorithms. In Linear Regression, the results don't improve significantly with increased number of features.

B. Classification algorithms

As discussed before, it can be seen that the accuracy is not a suitable metric for this problem. Our recall scores may have

TABLE VII: Linear Models

Regression	Metrics			
	$R2_{\text{Train}}$	$R2_{\text{Test}}$	$RMSE_{\text{Train}}$	$RMSE_{\text{Test}}$
Linear Regression	0.82907	0.82920	9.03576	9.05644
Polynomial Regression	0.84282	0.84208	8.66458	8.70819
LASSO Regression	0.79677	0.83086	8.98626	9.01212
Decision Tree	0.86986	0.85649	7.88407	8.30137

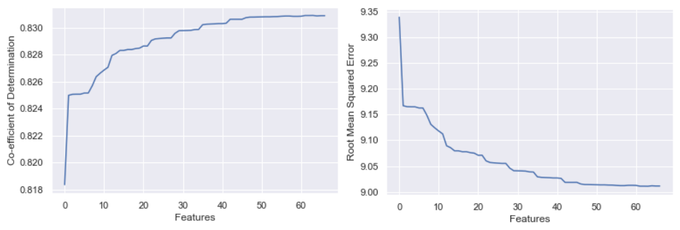


Fig. 2: Plots for Linear Regression

(a) R2 score vs. Number of features (b) RMSE vs. Number of features

TABLE VIII: Performance metrics of classifiers

Classifier	Metrics				
	Train accuracy	Test accuracy	Precision	Recall	F1 score
Logistic Regression	0.9105	0.9110	0.79	0.64	0.70
SVM	0.9144	0.9142	0.81	0.69	0.74
Decision tree	0.8924	0.8868	0.84	0.71	0.77
Perceptron	0.8116	0.8133	0.69	0.50	0.58

some scope for improvement. Number of false negatives is considerably high.

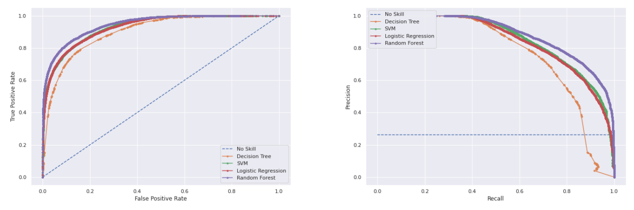


Fig. 3: Curves for classification algorithms

(a) ROC curve (b) Precision-Recall curve

We plot the standard ROC curve and the Precision-Recall curve, the latter being more suited given our imbalanced class distribution. Here, the Decision Tree Classifier's curve is closest to the perfect curve which agrees with the metrics shown in Table VIII.

C. Ensemble Learning

Ensemble learning techniques show quite some improvement, specifically in the F1 scores. Recall scores are still around the best performing non-ensemble classifier. Something to note is that these techniques are computationally expensive. This is the same reason as to why we couldn't generate the ROC and Precision-Recall curves on time for the submission.

TABLE IX: Performance metrics of tree-based methods

	Metrics				
	Train accuracy	Test accuracy	Precision	Recall	F1 score
Random Forest	0.9545	0.8954	0.86	0.69	0.76
Adaptive Boosting	0.8837	0.8820	0.83	0.70	0.76
Gradient Boosting	0.8918	0.8911	0.86	0.70	0.77

TABLE X: Performance metrics of voting classifiers

Classifier	Metrics				
	Train accuracy	Test accuracy	Precision	Recall	F1 score
Soft voting	0.9198	0.9211	0.84	0.67	0.75
Hard voting	0.9144	0.9142	0.89	0.60	0.72

V. Conclusion

We conclude that popularity can be predicted reasonably well using machine learning techniques. In both regression and classification, Decision Trees worked reasonably well. Ensemble algorithms give a slight improvement over traditional classification algorithms, very similar results with a common shortcoming of mis-classifying popular songs as unpopular. These algorithms show similar performance, with voting classifiers and Random Forest giving marginally better accuracy and boosting algorithms giving better recall and F1 score. The differences are negligible and parallelizable algorithms can be preferred.

References

[1] R. Dhanaraj and B. Logan, "Automatic prediction of hit songs," pp. 488–491, 2005.

[2] R. Nijkamp, "Prediction of product success: explaining song popularity by audio features from spotify data," in *11th IBA Bachelor Thesis Conference*, 2018.

[3] A. Singhi and D. Brown, "Can song lyrics predict hits?" 2015.

[4] K. Bischoff, C. S. Firan, M. Georgescu, W. Nejdl, and R. Paiu, "Social knowledge-driven music hit prediction," in *Proceedings of the 5th International Conference on Advanced Data Mining and Applications*, ser. ADMA '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 43–54.

[5] E. Zangerle, M. Pichl, B. Hupfauf, and G. Specht, "Can microblogs predict music charts? an analysis of the relationship between nowplaying tweets and music charts," in *Proceedings of the 17th International Society for Music Information Retrieval Conference 2016 (ISMIR 2016)*. ISMIR, 2016.

[6] "Web api reference." [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/

[7] Y. E. Ay, "Spotify dataset 1921-2020, 160k tracks (version 10n," Jan 2021. [Online]. Available: https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks/version/10

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[11] T. Head, M. Kumar, H. Nahrstaedt, G. Louppe, and I. Shcherbatyi, "scikit-optimize/scikit-optimize," 2020. [Online]. Available: https://zenodo.org/record/4014775