SPOTIFY HIT SONGS PREDICTOR

DATA 1030-Yihuan Hu
12 Oct. 2021
https://github.com/YihuanHu/SPOTIFY-HIT-SONG-PREDICTOR

1. Introduction

With the rapid growth in the music industry, one question has been raised more and more often: what factor contributes to a hit song? A hit song is the one that appears everywhere with a catching tune. The definition of catching is a vague concept, and no one can assure success even an experienced musician. This project will focus on specific parts of a song to dig out the characteristics which benefit both industry and musicians. Not only will this help musicians to exaggerate some parts of songs to maximize their potential popularity, but it could also be useful for the industry to decide which songs should be invested in an advertisement [1].

This project attempts to use a machine learning model to classify a song as hit or flop with 6398 tracks. The dataset is originally from Kaggle contains 18 features of tracks such as loudness, danceability and key. Since the rise of the internet in the recent decades change the way people listen to music, data are chosen within the recent decade to have an insight on the change of people's favors. All the features are well defined and describe specific characteristics of one track except its URI, which will be dropped later.

Several authors used this dataset to predict the popularity of a song based on its specific features. One author used this dataset with 3-layer Deep learning to predict which one song would be a hit with 75% accuracy [2]. In Predicting Hit Songs Using Spotify, another author used the random forest classifier combined with a confusion matrix, which illustrates three features play an important role in prediction. And it answered the question: how to classify hit songs well with a higher score of 84% [3]. These two projects give an insight on which features should be focused on and questions on how to improve the previous deep learning model. Even these two cases had high accuracy, the model could be more precise combined with more methods besides these two.

2. Exploratory Data Analysis

Firstly, based on the value counts of each feature, histograms were drawn for 15 numeric features excluding those categorical features such as artists. All the features except mode were continuous features based on these histograms.

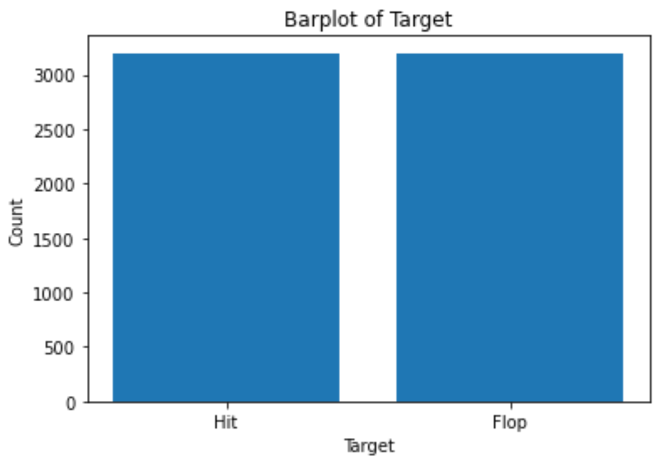

Figure 1 This bar plot shows how many tracks are hit or flop. Combined with value counts, it’s wired but highly confident that exactly 50% tracks are hits and 50% tracks are flops. In other words, this dataset is balanced, which means stratifyKfold is unnecessary in this case.

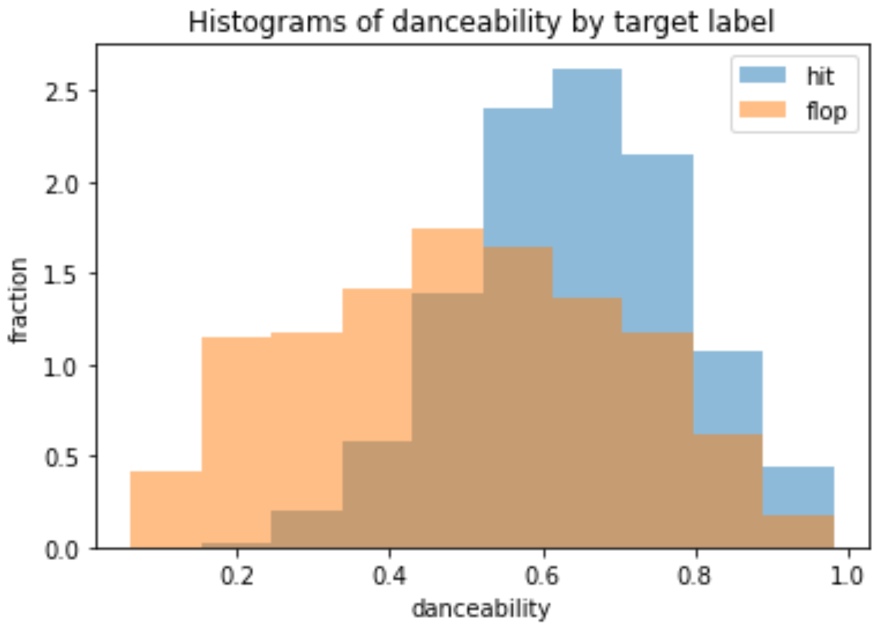

Figure 2 This histogram demonstrates the distribution of danceability labeled by target: hit or flop. Both distributions have a roughly bell shape and follow a Gaussian distribution. However, these two distributions seem different at the first glance since they have a different mean and variation: flop songs tend to have lower danceability around 0.5 compared with 0.7 average danceability in hit songs. Flop songs are more spread out from 0 to 1 while hit songs are more concentrated within one standard deviation. With such a modest overlap between two distributions, it is believed that this feature might have an important role in classification.

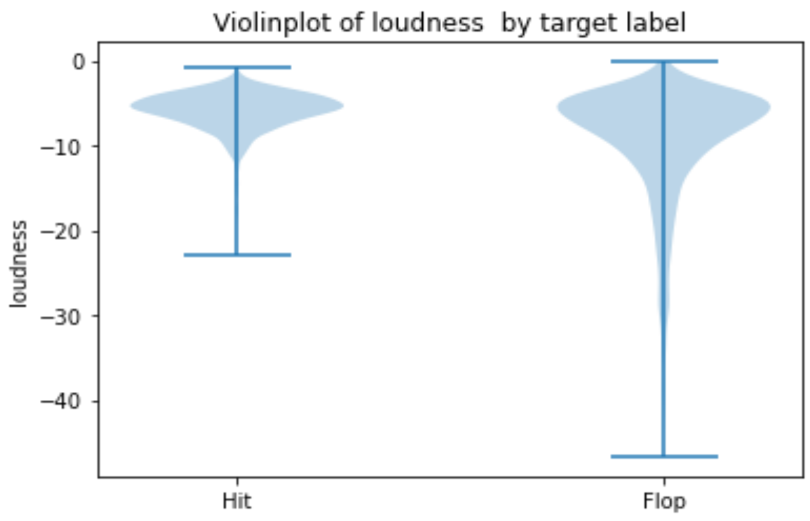

Figure 3 This violin plot shows the difference in kernel density plots divided by hit or flop songs. Both categories have similar mean and variant with unimodal. Nevertheless, it's clear that these two groups have distinct probability densities: hit songs have a narrower range while flop songs include loudness from -40 to 0. Even though flop songs include all the loudness hit songs, flop songs have a long tail on the other side. Based on these facts, a musician might avoid using loudness from -45 to -20 to increase their potential popularity.

3. Methods

3.1 Data Splitting and Preprocessing

This dataset is complete and didn’t have any null values. At the beginning of the data preprocessing step, 20% of tracks were split as test data using shuffle. And 80% of tracks were left for 5-fold cross-validation, which is to compensate for the small size of the original dataset and avoid the variability of random splits. For every fold of the cross-validation, the preprocessor fitted and transformed train data. Then, validation and test data were transformed based on train data. The resulting train validation and test data are 4094 1024 1280, which is around a proportion of 64-16-20. Since the popularity of every song is defined by its characteristics and every track has only one data point, it’s reasonable to assume the dataset is independent identical distribution without group structure or time series. The features mode was kept because it's a category that stands for where the track belongs to major genre or minor genre. Features 'key', 'mode','time_signature' and 'sections' were applied OneHotEncoder since they were categorical. Therefore, StandardScaler was applied for all the features except those four. Currently, this dataset had 69 features with a binary target.

3.2 Tune Models

Using the splitting and preprocessing strategies above, seven different machine learning models were used and compared to have detailed mining on the data including a logistic regression with L1 regularization, a logistic regression with L2 regularization, a logistic regression with ElasticNet regularization, a random forest classifier, a support vector machine classifier, a K-nearest neighbors classifier, and XGBoost classifier. All the models were tuned based on the validation scores using GridSearchCV to try to find the optimal combination among all the parameters. In the pipeline, every model firstly ran ten times to consider the non-deterministic of machine learning methods such as random forests. During the process, data were split using 10 different random seeds in the splitting strategy. Thus, the uncertainties could be calculated such as the mean and standard deviation of these results. Here were the parameters tuned for each model:

Model	**Parameters**
L1	**C**: 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4
L2	**C**: 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4
ElasticNet	**C**: 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4 **l1_ratio**: 0.01, 0.25, 0.5, 0.75
RF	**max_features**: 1, 3, 5, 10, 12, None **max_depth**: 1, 3, 5, 10, None
SVC	**gamma**: 1e-2, 1e-1, 1e0, 1e1, 1e2, 'auto', 'scale' **C**: 0.1, 0.32, 1, 3.2, 10
KNN	**n_neighbors**: 1, 5, 10, 30, 100 **weights**: 'uniform', 'distance'
XGBoost	**max_depth**: 1, 3, 5, 10, 30

4. Results

4.1 Model Comparison and Selection

Based on the built-in validation score, the best model parameters were found for each random state for later comparison. This project used an accuracy score as a metric since there was no preference over false negative and false positive. The plot was designed for comparing different models:

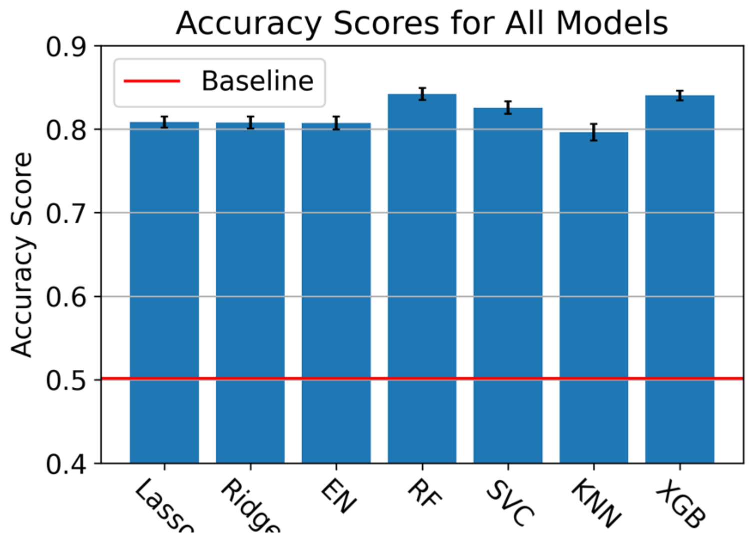

Figure 4 Bar plot of accuracy scores for the best model among different ML models with different random states

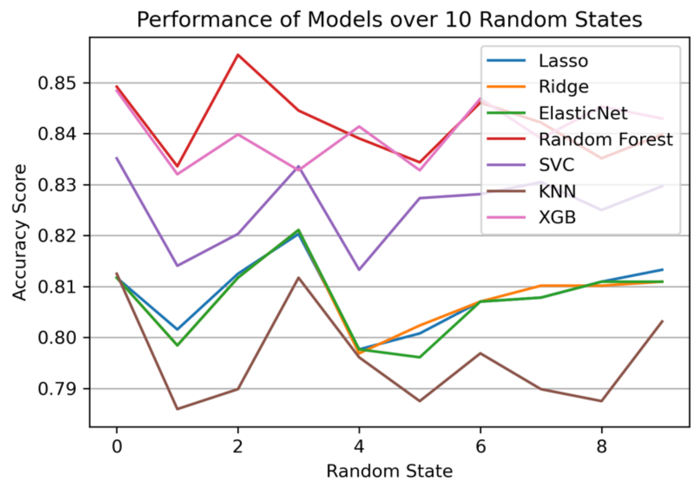

Figure5 Chart plot of accuracy scores for all ML models including 10 random states

All the models were much better than the baseline scores. According to the bar plot, Random forests and XGBoost were candidates for the best model. As the chart plot shown, we could assure random forest had better performance than XGBoost for most of the time. Thus, random forests would be the best model for later er interpretation. After checking 10 random states, paraments were chosen max_features = null and max_depth = 5 since 7 out of 10 random states have the same parameter combination.

4.2 Best Model Performance

To have a concise understanding of how the random forests model was superior to the baseline model, the model was split using new random states. This whole preprocessing and splitting process was repeated 100 times due to the computation time. For each split, the same proportion was applied that 80% used for train data while the left 20% for test data. The baseline scores and test scores for all random states were recorded. The baseline model has an accuracy of 0.50 with a standard deviation of 0.01 while the best model has a higher accuracy of 0.84 with a lower standard deviation of 0.007. Therefore, the best model was 34 standard deviations above the baseline model. The baseline model was 49 standard deviations below the baseline model.

Based on the scores calculated above, the ROC curve and confusion matrix were plotted:

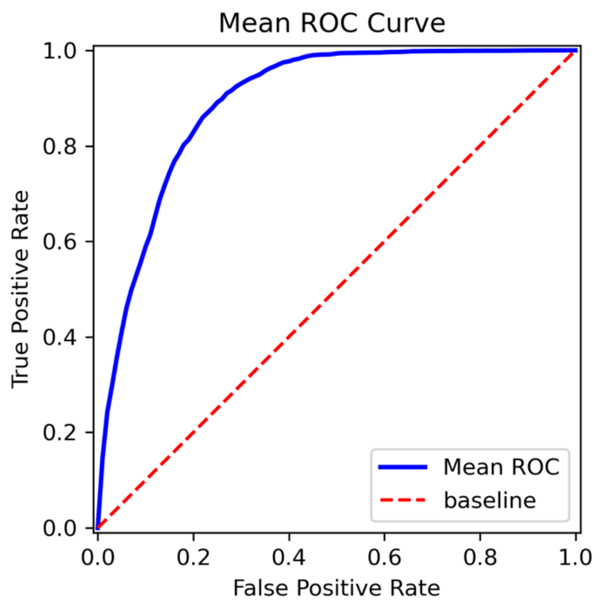

Figure 5 ROC curve for best model over 100 random states

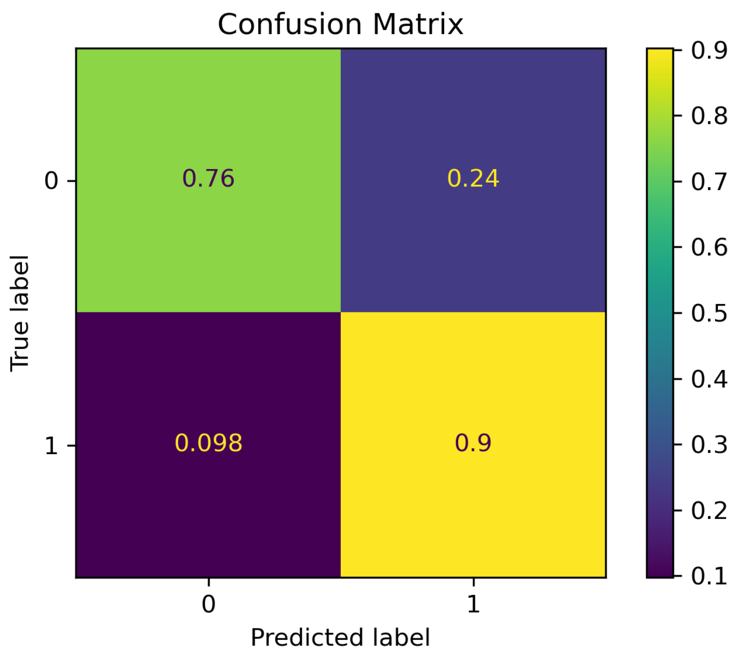

Figure 6 Confusion Matrix for best model over 100 random states

Those two plots showed that the model was highly accurate and much better than the baseline model. While from the confusion matrix, false positive was two times false negative.

4.3 Interpretation of Global Importance

To have a basic idea of which features played a more important role in the whole model, three global importance methods were applied including permutation test, SHAP value and the built-in function Mean Decrease Impurity(MDI) for the best model:

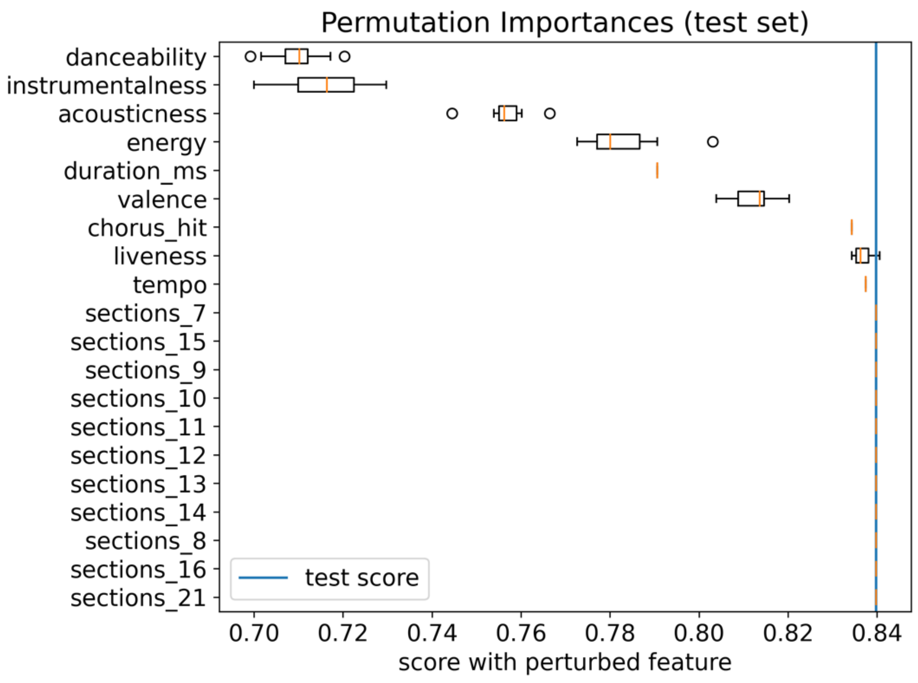

Figure 7 Permutation test of top 20 important features for the best model

From this permutation test plot, the large difference between the test score and permutation importance, the more important those features were such as danceability, instrumentalness, acousticness, and energy.

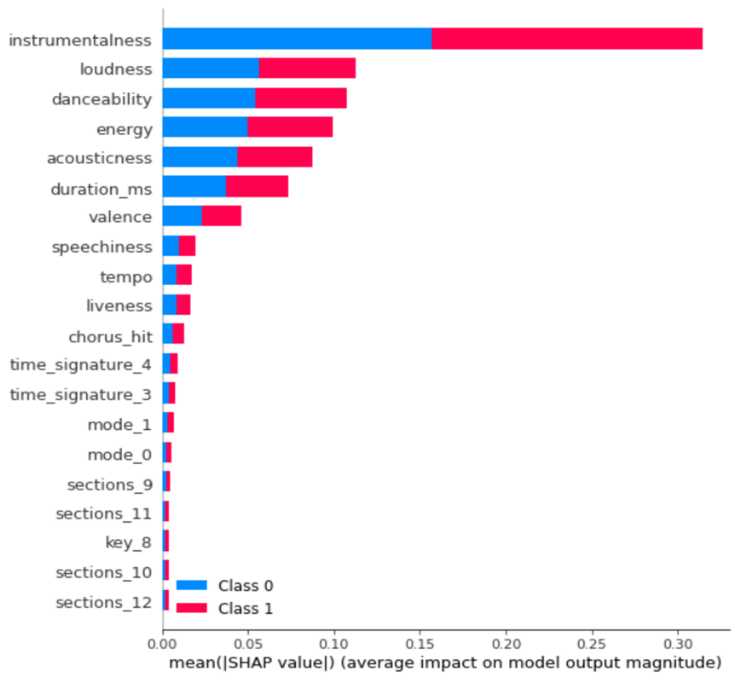

Figure 8 Top 20 SHAP values including all points for the best model

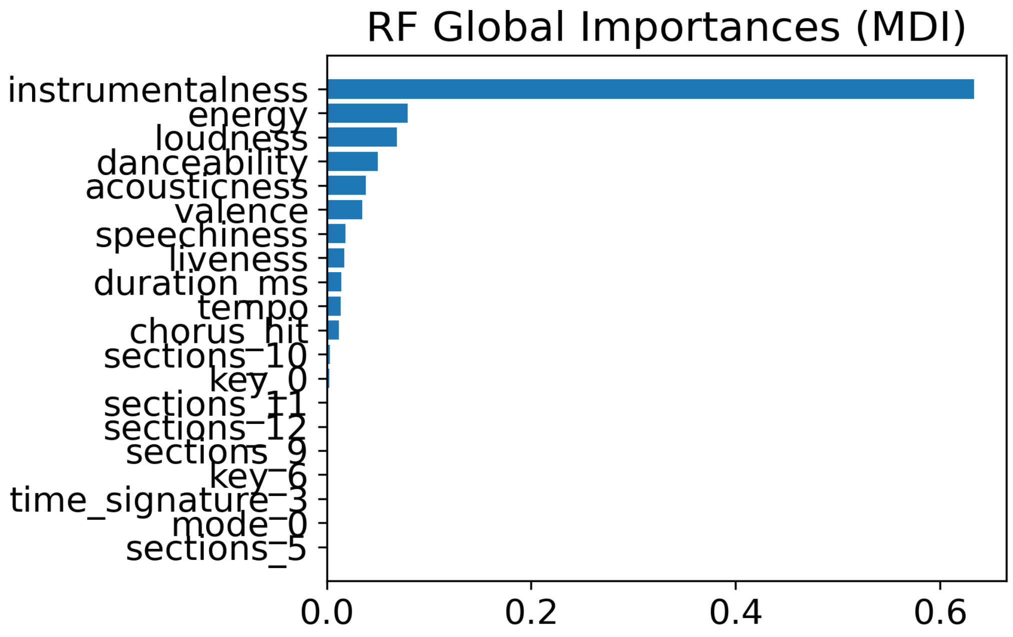

Figure 8 Top 20 Mean Decrease Impurity for the best model

For those two plots above, the large value it was, the more important this feature was. MDI used a similar method to the permutation test which was biased to high cardinality features. That was why those three ways of computing global importance had similar but slightly different results. No matter the small discrepancy, those four features were important among the model: danceability, instrumentalness, acousticness, and energy. The most important one was instrumentalness, which is defined to predict whether a track includes vocals.

And the same process repeated, the least important features were found using the same process above in the code file. Use MDI as an illustration:

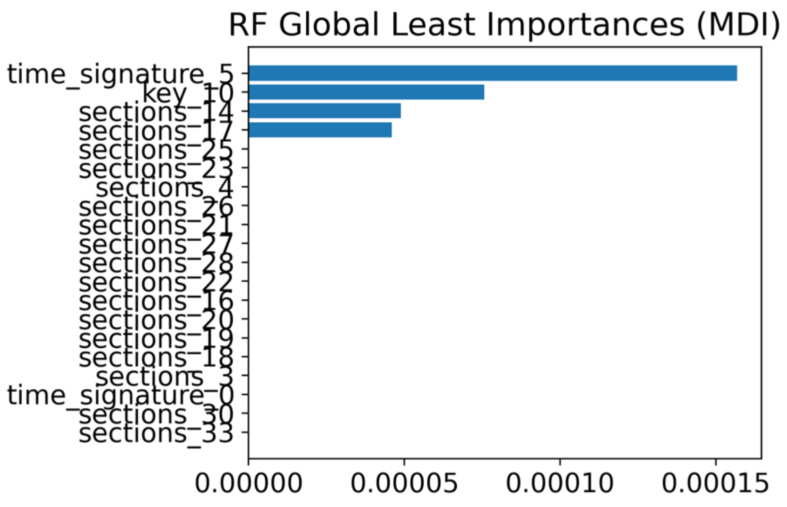

Figure 9 The least important features based on MDI for the best model

There was no doubt that the less important feature were section and time_signature, which was the number of repeat sections in one song and the length of the song.

4.4 Interpretation of Local Importance

To double-check the validation of the global importance, several parts of the data set were randomly picked to calculate local importance. For instance, point 250 was picked to draw a force plot:

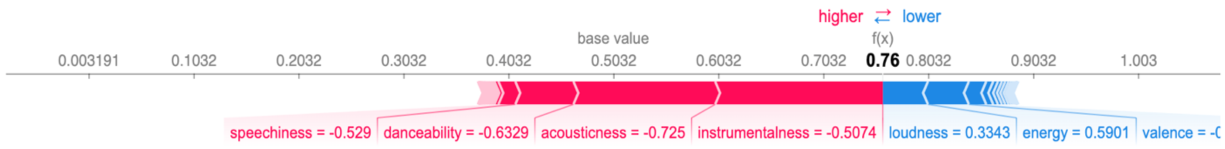

Figure 10 The force plot of point 250

This plot was aligned with the global importance that important features pushed the predicted probability to a high level. However, feature energy pushed down the predicted probability while it played an important role in global importance. This could be explained by the uniqueness of this song that it may have some certain characteristics in model to the less popularity.

Then, zoom on the first 1000 points of test data with the SHAP value. This resulting plot was perfectly matched with the global importance:

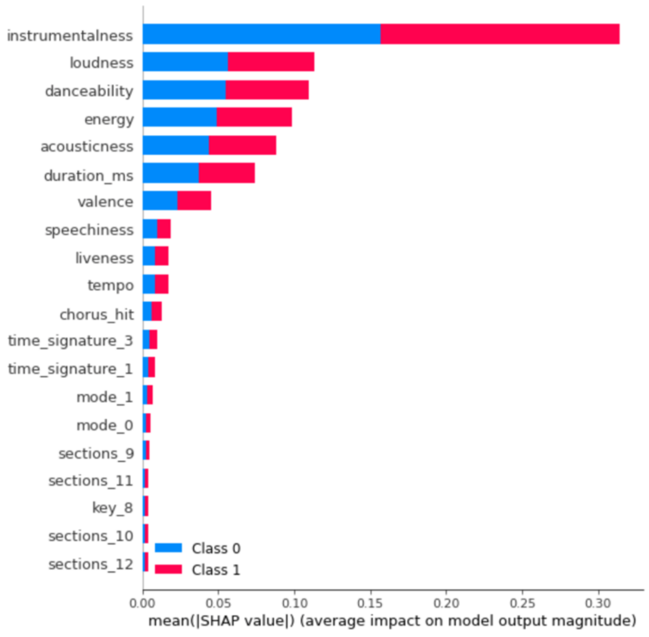

Figure 10 SHAP values of first 1000 points

5. Outlook

Since the current model is random forests, the results are not so interpretable. But it does give some hints on how to improve. The confusion matrix showed that the false positive is two times the false negative. In reality, companies prefer to have less false positive (predict the song as a hit but turns out as flop) to decrease their spending. An improvement that changing the metric from accuracy to f score less than one would be beneficial because more penalty would be put on the false negative. Besides, due to the limitation of computational power, XGBoost may be more accurate than current random forests if more parameters were tuned. The focus of this project is for music within the decade. It is of great benefit if the dataset can include music from the 60s to the recent decade.

Reference:

[1] Georgieva, E., Şuta, M., & Burton, N.S. (2018). HITPREDICT : PREDICTING HIT SONGS USING SPOTIFY DATA STANFORD COMPUTER SCIENCE 229 : MACHINE LEARNING.

[2] Abdul, Meral. Spotify-Deep Learning in 10 Steps [https://www.kaggle.com/abdulmeral/spotify-deep-learning-in-10-steps].

[3] Anneka, Osmun. Predicting Hit Songs Using Spotify [https://www.kaggle.com/annekaosmun/predicting-hit-songs-using-spotify/notebook].