50.038 Computational Data Science

Hit Song Prediction

Group 8

Chen Yan 1003620

Dong Ke 1003713

Gladys Chua Shi Ying 1003585

Pang Luying 1003631

1. Introduction

When we listen to songs, despite the different taste in music, one can more or less deduce whether a song will be popular. However, rather than using our intuition or "gut-feeling" to predict hit songs, the purpose of the project is to see if we can use intrinsic music data to identify Hit songs by machine learning.

This project is very meaningful as it can help music producers and artists get to know their audience better and produce more popular songs by tweaking the music qualities. Additionally, it could help the agents to discover underground artists whose music resembles the popular patterns and lead them to gain a higher reputation.

The dataset was scraped from the Spotify weekly music chart from 2019 to 2020. We defined the top 15 weekly songs as hit-songs and the bottom 15 as non-hits. We processed the data using the Spotify API and extracted 13 features that we used to feed into various machine learning models.

The best prediction is obtained for the random forest model, which has an accuracy of 93%. There are also several other models that have fairly good performance, which are KNN, Adaboost whose accuracy is 70%, 80% respectively.

2. Literature Review

Hit Song Science is a term coined for the prediction of hit songs through machine learning techniques. It is an active research topic in Music Information Retrieval (MIR). There are two main trends in Hit Song Science. In the first trend, it started with capturing the intrinsic audio features that could possibly explain the popularity, and it later includes lyrics explanation. In the second trend, the social factors that influence popularity were explored. In this section, we would briefly review some of the research that has influenced our project or has influenced the field of hit song science.

For the first trend, the first audio-based hit prediction research[1] was done using Support Vector Machine (SVM) and achieved an accuracy of about 68%. Another research[2] built a combination of Convolutional Neural Network model and advanced JYnet model. There is also another research[3] that uses logistic regression and achieves an AUC of 0.81 corresponding to the test set. All these researches inspired us on the possible models that we could explore.

One of the research that we studied more thoroughly is based on early adopter data and audio features[4]. In this project, "a list of hit songs was parsed from dance charts listed on the Belgian website 'The Ultratop 50'" and non-hits as those songs that were predicted to be hit songs but eventually never became hits. According to the paper, "a total of 140 audio features, including those that capture a temporal aspect, were extracted through the EchoNest

[1] R. Dhanaraj, B. Logan, Automatic prediction of hit songs., in: ISMIR, 2005, pp. 488–491.

[2] L.-C. Yang, S.-Y. Chou, J.-Y. Liu, Y.-H. Yang, Y.-A. Chen, Revisiting the problem of audio-based hit song prediction using convolutional neural 405 networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on, IEEE, 2017, pp. 621–625.

[3] D. Herremans, D. Martens, K. S ¨orensen, Dance hit song prediction, Journal 390 of New Music Research 43 (3) (2014) 291–30 2

[4] Herremans, D., & Bergmans,T.(2017). Hit song prediction based on early adopter data and audio features. The 18th International Society for Music Information Retrieval Conference (ISMIR)- Late Breaking Demo, Suzhou, China. October, 2017.

API." Based on the early adopter model, the project is able to predict the upcoming top 20 hits with an AUC value of 0.79.

A second research[5] that influenced our project extracts the features using the Spotify Web API. The project uses a very large dataset that includes approximate 1.8 million songs from 1985 to 2018 based on the belief that a larger dataset would give a more robust model. However, we have some reservations about this hypothesis as we believe that people's taste in music would change, and scraping a large dataset over a very long time may not necessarily be the best solution due to the fluidity in taste. Thus, the dataset we selected is from 2019 to 2020, which is more recent and relevant.

For the second trend, we studied this paper[6], in which the author includes the social factors that can lead to the rise of hit-songs using classification models based on which users are currently listening to the songs. Though we have not explored the social factors within our scope of the project, this would be a future direction in which we could include the celebrity effect and promotion actions that would influence the popularity of the song track produced.

[5] Kai Middlebrook,Kian Sheik, SONG HIT PREDICTION: PREDICTING BILLBOARD HITS USING SPOTIFY DATA, September 20, 2019

[6] Dorien Herremans Hit song prediction through early adopter data from social networks, Preprint submitted to Knowledge Based Systems December 13, 2020

3. Dataset and Collection

The music dataset is scrapped from Spotify's weekly Global music chart from 2019 to 2020 and stored in CSV file format on a weekly basis. We then load the data into a pandas dataframe, and split the dataset into training data and testing data.

The dataset is originally in the data folder with CSV files. We get the URL of the track and the rank of songs from the CSV files. Then, we proceed to extract music features using the Spotify API with the URL obtained.

3.2 Data Pre-processing

Using Spotify's Audio Features & Analysis API, the following features were collected for each song:

- Mood: Danceability, Valence, Energy, Tempo
- Properties: Loudness, Speechiness, Instrumentalness
- Context: Liveness, Acousticness

Feature Name	Value Type	Description
duration_ms	int	The duration of the track in milliseconds.
key	int	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.

mode	int	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
time_signature	int	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
acousticness	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.

energy	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	float	Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.

loudness	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
speechiness	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
valence	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

tempo	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

4. Problem and Algorithm/Model

4.1 Problem

This project aims to predict hit songs to see if we can use intrinsic music data to identify Hit songs by machine learning. Hit song prediction collates a dataset of hit songs from popular charts. We defined the top 15 as hit-song and bottom 15 as non-hits and use different machine learning algorithms to train models with a select dataset within one year from Spotify API.

4.2 Models & Algorithms

To predict whether a song will be a hit or not, we tried eight different machine-learning models:

- *Logistic Regression (LR)*
- *Decision Trees (DT)*
- *Support Vector Machine (SVM)*
- *K Nearest Neighbor Classifier (KNN)*
- *Gaussian Naive Bayes*
- *AdaboostClassifier*
- *RandomForestClassfier*
- *Convolutional Neural Network (CNN)*

Logistic regression (LR) is a popular classification algorithm. It has a wide application that includes machine learning, social sciences, and most medical fields in practice when the dependent variable (target) is categorical. We use LR to find the relationship between features and output the probability that how likely is to occur given these features.

Decision Trees (DT) is one of the easiest and popular classification algorithms to understand and interpret. It's a method commonly used in data mining and belongs to the family of supervised learning algorithms to solve regression and classification problems. For our project, we assume that all of the input features have finite discrete domains and use a Categorical Variable Decision Tree to classify hit and non-hit songs. Each node in the tree

acts as a test case for features and each edge descending from the node corresponds to the possible answers to the test case. We start from the root of the tree to predict a class label for a record. Then, we compare the values of the root attribute with the record's attribute. On the basis of comparison, we will continue to go to a certain branch to that value and jump to the next node. The whole process is recursive and repeated for every subtree rooted at the new node. We set the max_depth parameter as 10 and the minimum number of samples in a leaf as 2 for training the Decision Tree model in order to solve overfitting.

Support Vector Machine (SVM) aims to find the most optimal hyperplane in an N-dimensional space that distinctly classifies the data points. To separate two classes of data points, we find a plane that has the maximum margin (the maximum distance between data points of both classes), so that it can provide reinforcement for future points to be classified with more confidence. We used Gaussian Radial Basis Function(RBF), $K(x, x') = exp(-\frac{\|x-x'\|^2}{2\sigma^2})$, as our kernel.

K Nearest Neighbor Classifier *(KNN)* is a simple, easy-to-implement supervised machine learning algorithm to solve both classification and regression problems. It assumes that similar things exist in close proximity. We load the data and initialize the value of k. Iterated training data from 1 to the total number of data to get the predicted class. We calculate the Euclidean distance between points to evaluate and sort the calculated distances from smallest to largest (in ascending order) by distance values. Pick the top k entries from the sorted array and get the labels of the selected K entries and return the predicted class. For the K value, we got the result $k = 39$ with the general formula, $k = \sqrt{N}$, where N is the number of song samples in our training dataset. In addition, k should be an odd number to avoid confusion between two classes of data.

Gaussian Naive Bayes is the most straightforward and fast algorithm to do classification for a large chunk of data. It is the simplest supervised learning algorithm and commonly used in sentiment analysis, spam filtering, text classification, and recommender systems to predict unknown classes with high accuracy. We assumed that the effect of a particular feature in a class is independent of other features. For our project, we calculated the prior probability for given class labels (hit and non-hit) and found the likelihood probability with each attribute for the two classes (hit and non-hit). Then, we use the Bayes formula with these values to calculate posterior probability and distribute the input to the higher probability class.

Generally, ***Adaboost Classifier*** has high accuracy. It combines multiple weak classifiers to increase the accuracy of classifiers by setting the weights of classifiers and training the data set for every iteration. It trains the model iteratively on a training dataset with a previous accurate prediction of training. In each iteration, the model will be assigned a high weight to wrong classified observations to get a higher probability of classification in the next iteration. When training data fits our model completely with zero error or reaches the max number of estimators, the iteration will be finished.

Random Forest is a simple algorithm with a great result at most times without hyper-parameter tuning. It's a supervised learning algorithm and an ensemble of decision trees with a "bagging" method. Random Forest can get a more accurate and stable prediction by merging multiple decision trees together.

Convolutional Neural Network (CNN) is a Deep Learning algorithm. Various features of songs are assigned learnable weights and biases for getting a higher accuracy training model. Compared with other models, CNN has less pre-process requirements and a good ability to learn filters or characteristics in hit songs. In our CNN model, each layer has a 1D convolutional layer, a pooling layer, and a dropout layer. We use relu and softmax as our activation function. We set the number of epochs to 20 for the model with normalized x and binary matrices y.

5. Evaluation Methodology

We evaluate all algorithms that we choose and get a series of results of these algorithms. We introduce the confusion matrix, accuracy, precision, recall, F1-score, Roc, and AUC to analyze our model. The confusion matrix shows the value of true positive, true negative, false positive, false negative of the dataset. Accuracy works best if false positives and false negatives have similar costs. Precision shows how precise the model is out of those predicted positives. Recall is the ratio of correctly predicted positive (True Positives) to actual positive. It shows the proportion of actual positives that are identified correctly. F1-score is the weighted average of Precision and Recall. Receiver Operating Characteristics (ROC) is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR). Area Under the Curve (AUC) represents degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0's as 0's and 1's as 1's.

We get the AUC, ROC, accuracy, precision, recall, f1-score and confusion matrix for each model. The classes that we defined are balanced. We use f1-score to judge models with the help of accuracy and AOC. We obtain these results for 2019.csv without feature selection as follows. All features that we obtained from songs are standardized for training.

The ***Logistic Regression*** model yielded **64**% accuracy on the test data. This model doesn't generalize well from our training data to testing data with low f1-score (67%) and accuracy (64%). The precision and recall were acceptable at **63**% and **71**%. AUC of this model is 64%. The confusion matrix on the validation set shows the numbers of false positives (Table 1.1). *Table 1.3* is the ROC graph of our model. *Table 1.2* is a graph to visualize the prediction results of our model.

```
AUC:
0.6394913303246789
report:
              precision    recall  f1-score   support

         0.0       0.65      0.57      0.61       229
         1.0       0.63      0.71      0.67       239

    accuracy                           0.64       468
   macro avg       0.64      0.64      0.64       468
weighted avg       0.64      0.64      0.64       468

ConMatrix:
[[130  99]
 [ 69 170]]
```

Table1.1

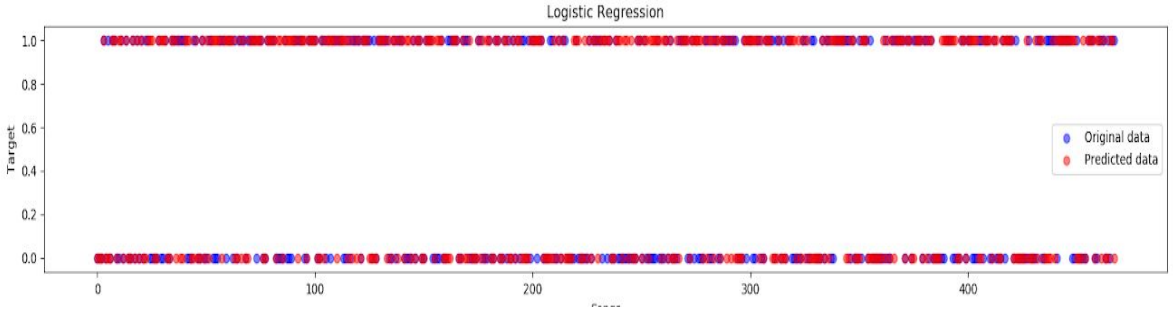


Table1.2

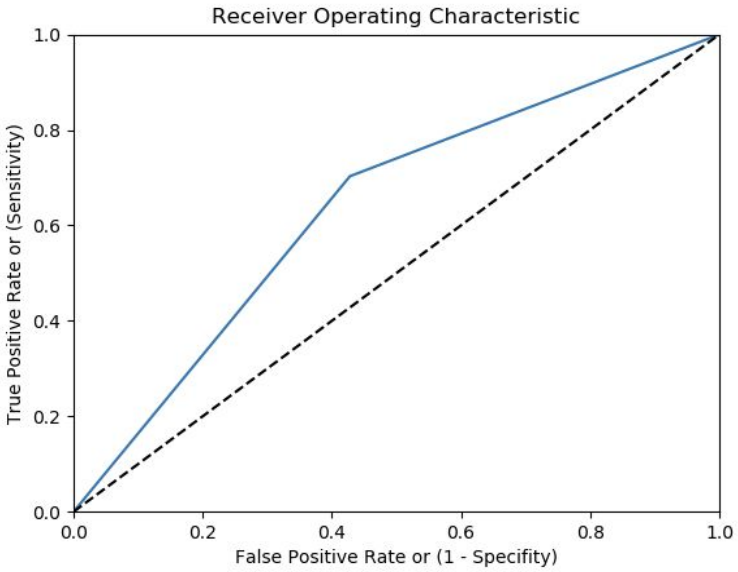


Table 1.3

The ***Decision Tree*** model yielded **60**% accuracy on the test data. Our model doesn't generalize well from our training data to testing data with low f1-score (62%) and accuracy (60%).The precision and recall were acceptable at **60**% and **65**%. AUC of this model is 60% (Table 2.3). The confusion matrix on the validation set shows the numbers of false positives (Table 2.1). *Table 2.2* is a graph to visualize the accuracy of our model.

```
AUC:
0.5993769527324551
report:
              precision    recall  f1-score   support

         0.0       0.60      0.55      0.57       229
         1.0       0.60      0.65      0.62       239

    accuracy                           0.60       468
   macro avg       0.60      0.60      0.60       468
weighted avg       0.60      0.60      0.60       468

ConMatrix:
[[126 103]
 [ 84 155]]
training score:
0.9166666666666666
```

Table 2.1

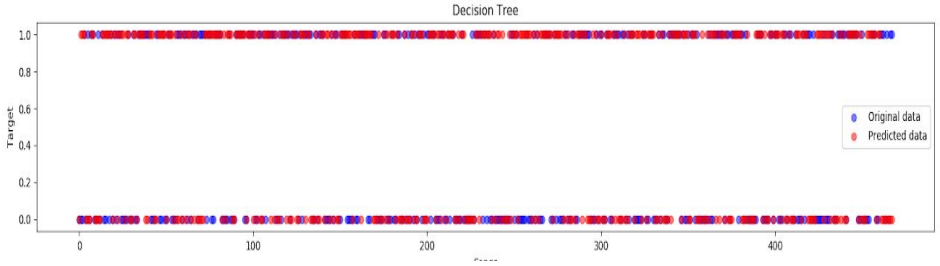


Table 2.2

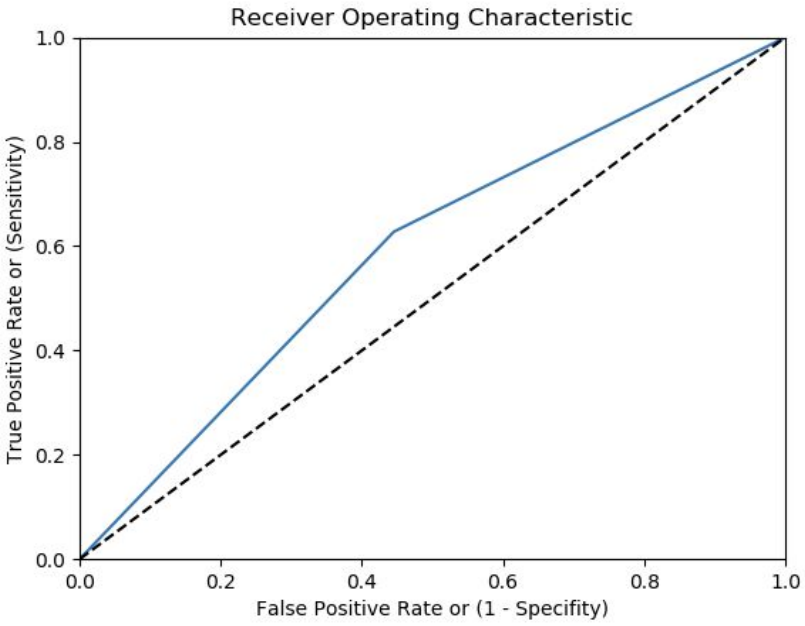


Table 2.3

The ***Support Vector Machine*** model yielded **60**% accuracy on the test data. Our model doesn't generalize well from our training data to testing data with low f1-score (64%) and accuracy (60%). The precision and recall were acceptable at **60**% and **69**%. AUC of this model is 60% (Table 3.3). The confusion matrix on the validation set shows the numbers of false positives (Table 3.1). *Table 3.2* is a graph to visualize the accuracy of our model.

```
AUC:
0.6029215618205405
report:
              precision    recall  f1-score   support

         0.0       0.61      0.52      0.56       229
         1.0       0.60      0.69      0.64       239

    accuracy                           0.60       468
   macro avg       0.61      0.60      0.60       468
weighted avg       0.61      0.60      0.60       468

ConMatrix:
[[119 110]
 [ 75 164]]
```

Table 3.1

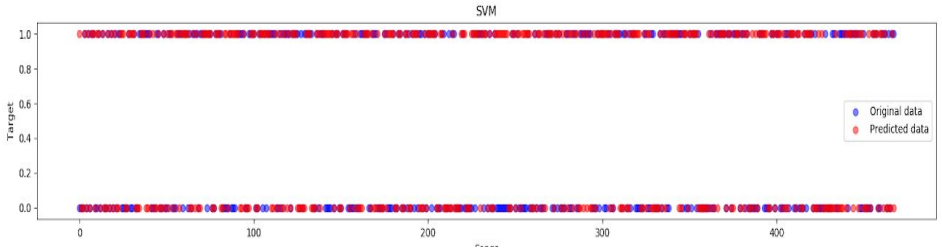


Table 3.2

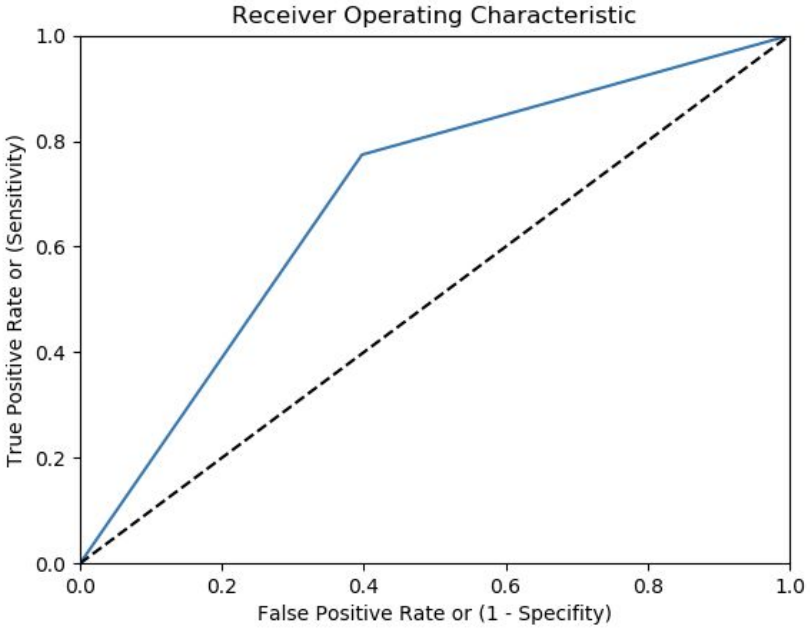


Table 3.3

The ***K Nearest Neighbor Classifier*** *(KNN)* model yielded **70**% accuracy on the test data when we set k = 39. For KNN mode, it is necessary and important to choose a suitable k value for classify. Therefore, we perform a validation set to help us choose a good k value for this model. We set k = 39 as the k parameter of the KNN model by comparing the training accuracy and validation accuracy for different K values as follows. (Table 4.0).

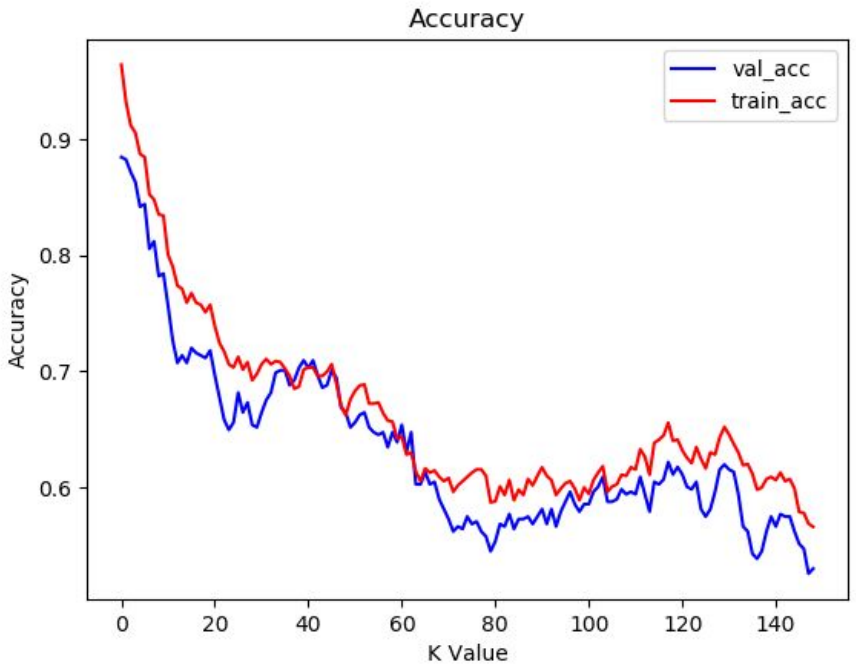


Table 4.0

From the graph, we found the value of K around 40 is suitable for our model as the model gives similar accuracy for both validation and training set. Hence, we build our KNN model with k = 39, which is approximately similar to the result of the general function that we used to define the value of k. ($k = \sqrt{N}$, where N = 1560)

This model generalizes well from our training data to testing data with high F1-score (74%) and accuracy (70%). The precision and recall were acceptable at **67**% and **81**%. AUC of this model is 70% (Table 4.3). The confusion matrix on the validation set shows the numbers of false positives (Table 4.1). *Table 4.2* is a graph to visualize the accuracy of our model.

```
AUC:
0.7007089218176171
report:
              precision    recall  f1-score   support

         0.0       0.75      0.59      0.66       229
         1.0       0.67      0.81      0.74       239

    accuracy                           0.70       468
   macro avg       0.71      0.70      0.70       468
weighted avg       0.71      0.70      0.70       468

ConMatrix:
[[136  93]
 [ 46 193]]
training score:
0.5045787545787546
```

Table 4.1

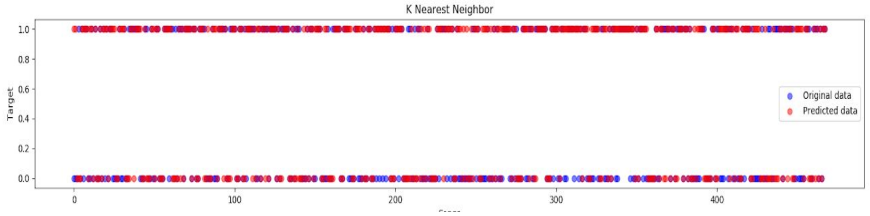


Table 4.2

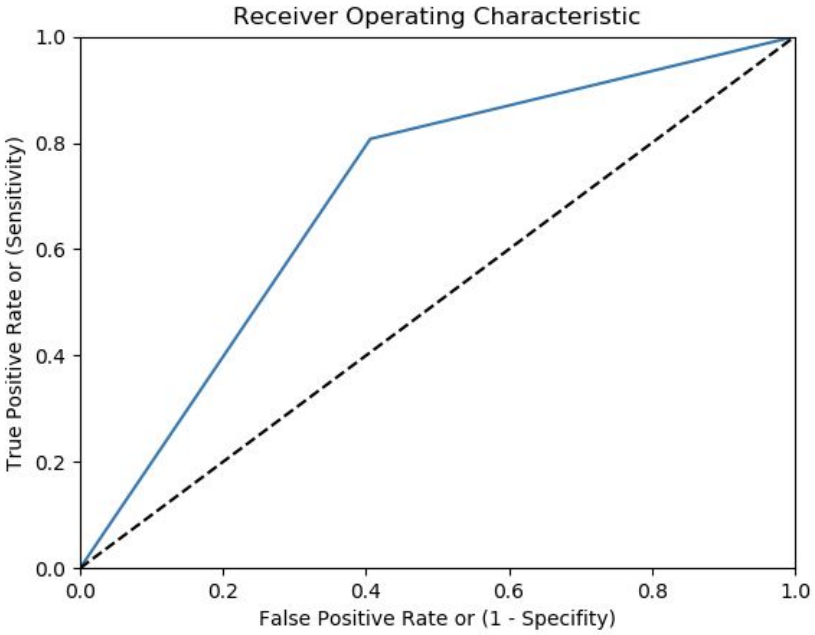


Table 4.3

The ***Gaussian Naive Bayes*** model yielded **56**% accuracy on the test data. This model doesn't generalize well from our training data to testing data with low f1-score (42%) and accuracy (56%). The precision and recall were acceptable at **63**% and **32**%. AUC of this model is 56% (Table 5.3). The confusion matrix on the validation set shows the numbers of false positives (Table 5.1). *Table 5.2* is a graph to visualize the accuracy of our model.

```
AUC:
0.5629259469039484
report:
              precision    recall  f1-score   support

         0.0       0.53      0.81      0.64       229
         1.0       0.63      0.32      0.42       239

    accuracy                           0.56       468
   macro avg       0.58      0.56      0.53       468
weighted avg       0.58      0.56      0.53       468

ConMatrix:
[[185  44]
 [163  76]]
training score:
0.5045787545787546
```

Table 5.1

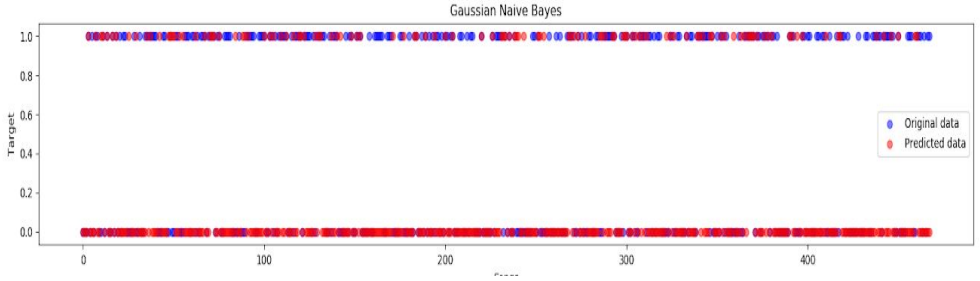


Table 5.2

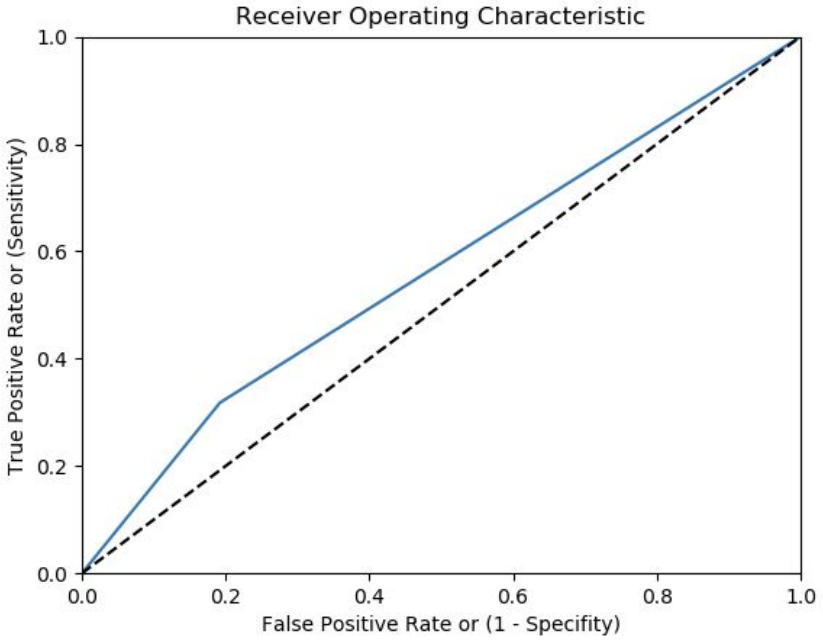


Table 5.3

The ***AdaboostClassifier*** model yielded **80**% accuracy on the test data. This model generalizes well from our training data to testing data with high f1-score (82%) and accuracy (80%). The precision and recall were acceptable at **77**% and **89**%. AUC of this model is 80% (Table 6.3). The confusion matrix on the validation set shows the numbers of false positives (Table 6.1). *Table 6.2* is a graph to visualize the accuracy of our model.

AUC:
0.8015932469715518
report:

	precision	recall	f1-score	support
0.0	0.86	0.72	0.78	229
1.0	0.77	0.89	0.82	239
accuracy			0.80	468
macro avg	0.81	0.80	0.80	468
weighted avg	0.81	0.80	0.80	468

ConMatrix:
[[164 65]
[27 212]]

Table 6.1

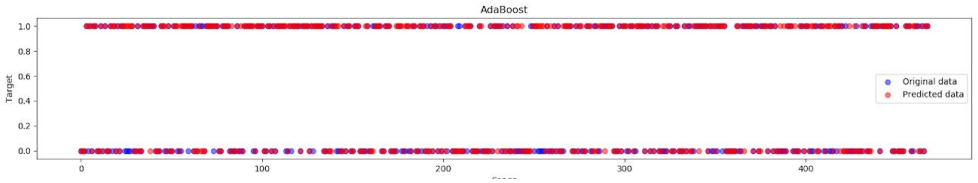


Table 6.2

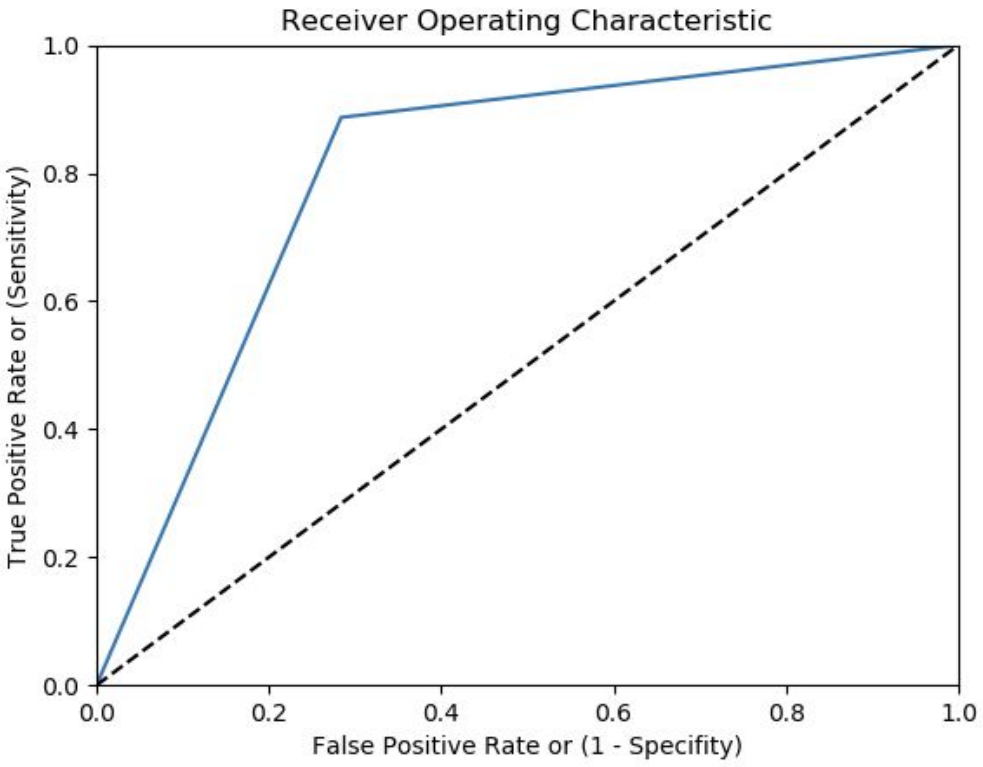


Table 6.3

The ***Random Forest*** model yielded **93**% accuracy on the test data. This model generalizes well from our training data to testing data with high f1-score (93%) and accuracy (93%). The precision and recall were acceptable at **93**% and **92**%. AUC of this model is 93% (Table 7.3). The confusion matrix on the validation set shows the numbers of false positives (Table 7.1). *Table 7.2* is a graph to visualize the accuracy of our model.

```
AUC:
0.9274085984177157
report:
              precision    recall  f1-score   support

         0.0       0.92      0.93      0.93       229
         1.0       0.93      0.92      0.93       239

    accuracy                           0.93       468
   macro avg       0.93      0.93      0.93       468
weighted avg       0.93      0.93      0.93       468

ConMatrix:
[[213  16]
 [ 18 221]]
```

Table 7.1

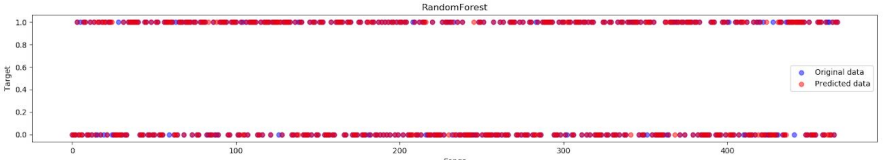


Table 7.2

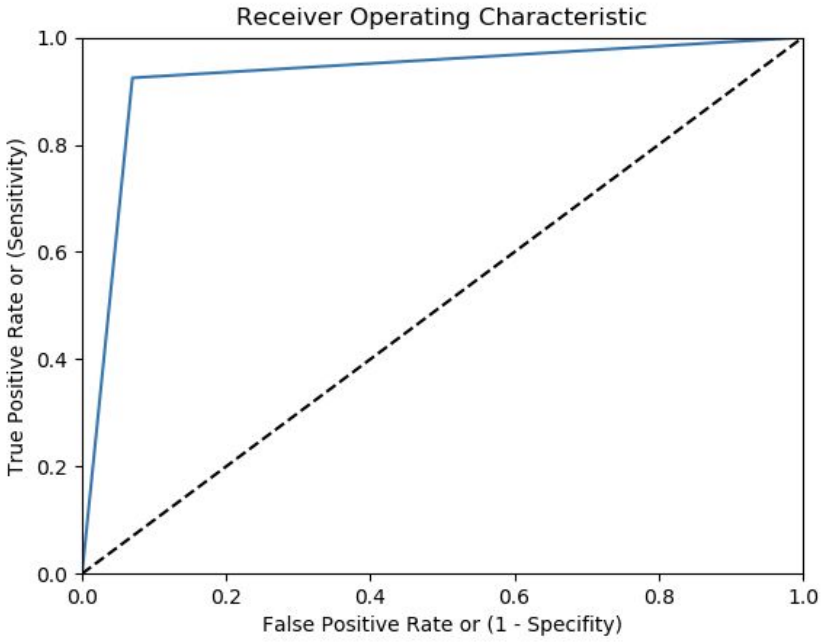


Table 7.3

The ***Convolutional Neural Network (CNN)*** model yielded **53**% accuracy on the test data. This model doesn't generalize well from our training data to testing data with low f1-score (62%) and accuracy (53%). The precision and recall were acceptable at **51**% and **79**%. AUC of this model is 54%. The confusion matrix on the validation set shows the numbers of false positives (Table 8.1). *Table 8.2* is a graph to visualize the accuracy of our model.

AUC:
0.5425656403130257
report:

	precision	recall	f1-score	support
0.0	0.60	0.30	0.40	242
1.0	0.51	0.79	0.62	226
accuracy			0.53	468
macro avg	0.56	0.54	0.51	468
weighted avg	0.56	0.53	0.51	468

ConMatrix:
[[72 170]
 [48 178]]

Table 8.1

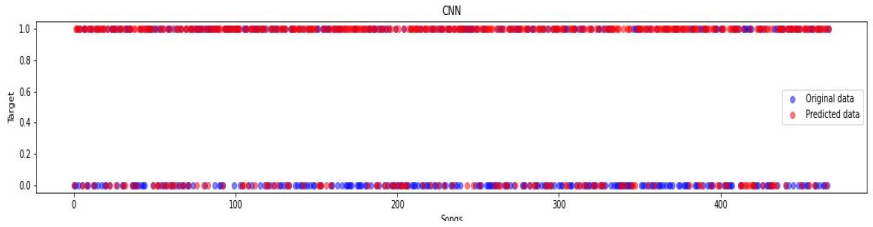


Table 8.2

For the model of CNN, we did further analysis to train the model. We want to check if deep learning will work better than others. Therefore, we compared training loss and validation loss as follows.(Table 8.3)

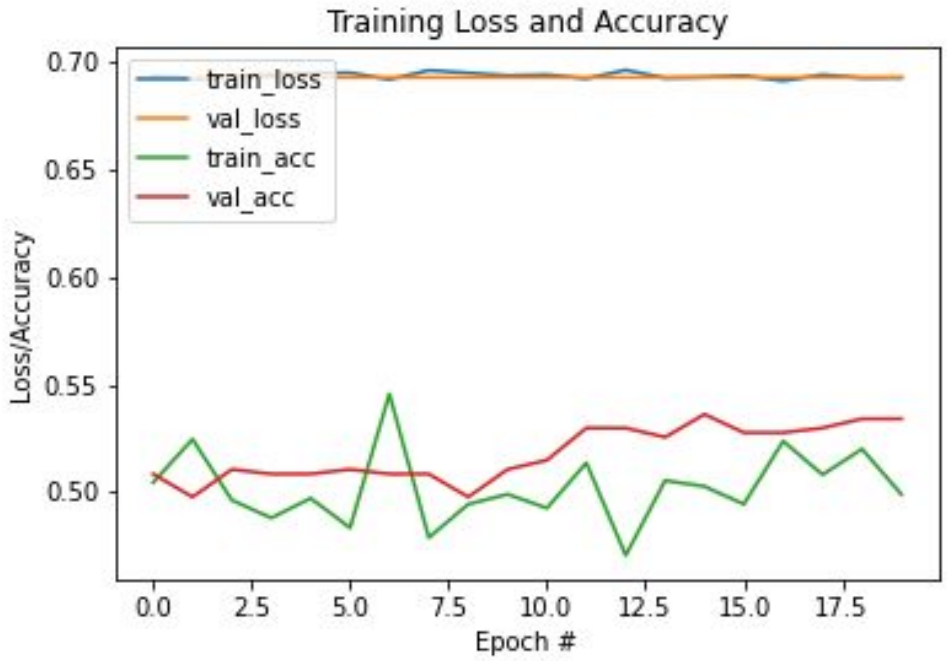


Table 8.3

From the graph, we find the training loss and validation loss of each epoch are similar, which is around 0.69. Validation accuracy is slightly higher than training accuracy.

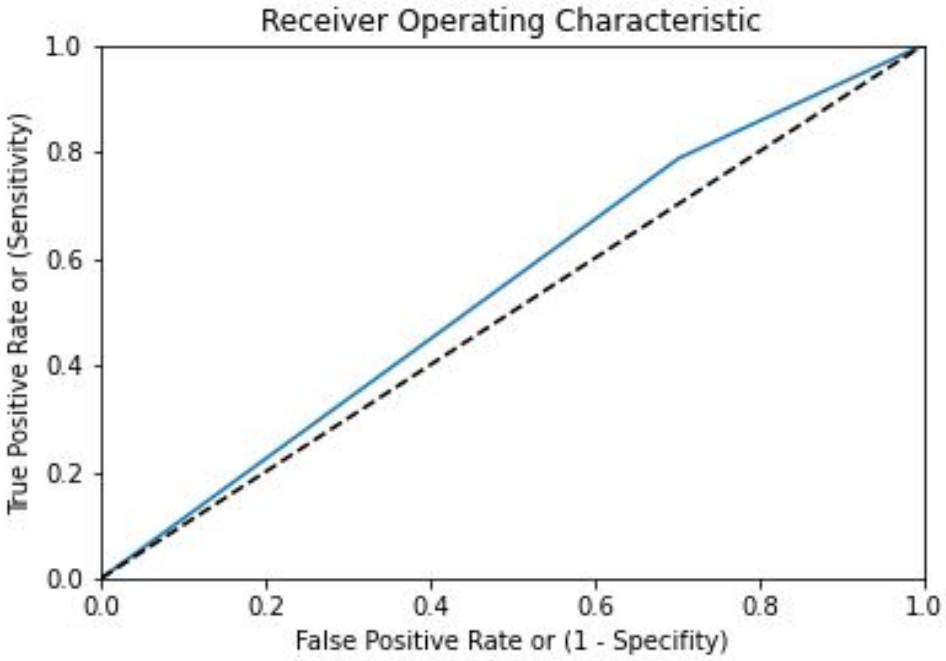


Table 8.4

However, when we look at the ROC graph (Table 8.4), we found AUC is 0.54. Which suggests that this model has bad ability (0.54) to perform classification. Considering all analysis as above, we didn't get a good training result for the deep learning model of CNN. Further work can be done to discover the principle of deep learning models to evaluate better models for hit songs prediction in the future.

6. Results and Discussion

6.1 Results

We reported the AUC,accuracy, precision and recall on table as follows for each model, but we mainly focused on the f1-score to analyse results.The compare table as follows:

Models	Logistic Regression (LR)	Random Forest	Decision Trees(DT)	Support Vector Machine (SVM)
	Test	Test	Test	Test
AUC	0.64	0.92	0.59	0.68
Precision	0.63	0.92	0.60	0.67
Recall	0.70	0.92	0.63	0.77
Accuracy	0.64	0.93	0.59	0.69
f1-score	0.67	0.92	0.61	0.72

Models	K-Nearest Neighbor Classifier	Gaussian Naive Bayes	AdaboostClassifier	CNN
	Test	Test	Test	Test
AUC	0.70	0.56	0.80	0.54
Precision	0.67	0.63	0.77	0.51
Recall	0.81	0.32	0.89	0.79

Accuracy	0.70	0.56	0.80	0.53
F1-score	0.74	0.42	0.82	0.62

We use F1-score and accuracy to compare each model and decide which one is the best model for our project. Higher F1-score, more robustic of the models. F1-Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. We use F1-score to analyze the results for each model because it is usually more useful than accuracy.Accuracy works best if false positives and false negatives have similar cost.

Looking at the table above, we select the top 3 models with the highest F1-score, Random Forest,AdaboostClassifier and K Nearest Neighbor Classifier (KNN).

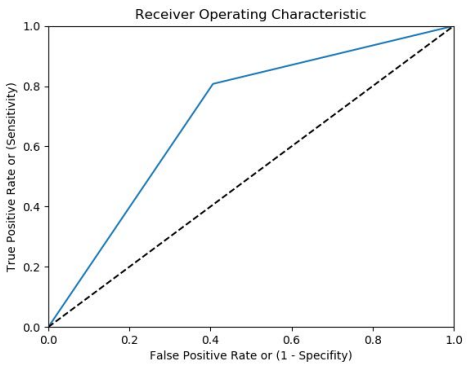
6.2 Discussion

In order to get better results for our project, we chose the top3 successful algorithms Random Forest,AdaboostClassifier and K Nearest Neighbor Classifier (KNN) for further investigation. We select the top 7 features(half of total numbers of features) for each algorithm by looking at how much the tree nodes that use that feature reduce impurity across all trees in the forest.Our evaluation process is as follows.

We compared the AUC graph and the value of the confusion matrix,accuracy, precision,recall,F1-score,Roc and to analyze these three selected models.

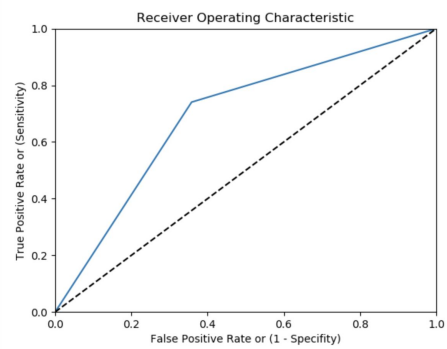
Firstly,we analyze the AUC graph for each model that we selected.

K Nearest Neighbor



Without Features selected

- Accuracy = 0.700
- AUC: 0.700 > 0.5

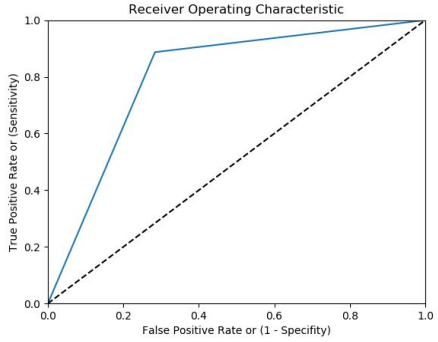


Without Features selected

- Accuracy = 0.700
- AUC: 0.691 > 0.5

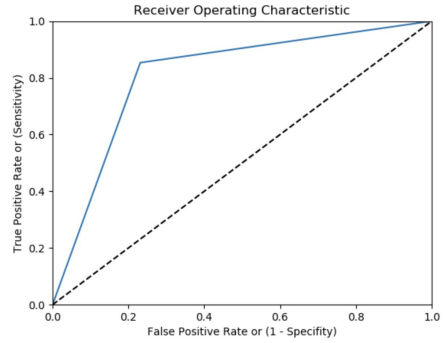
The model without features selected have better AUC value(0.700)

Adaboost Classfier



Without Features selected

- Number of estimators =1000
- Learnig rate = 0.1
- Accuracy = 0.800
- AUC: 0.800 > 0.5

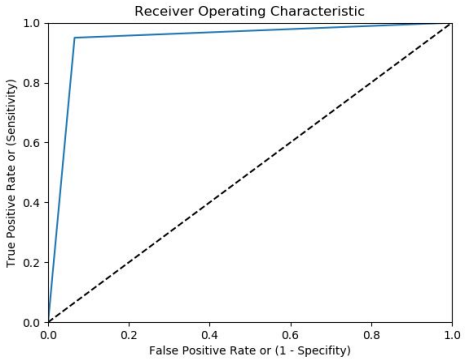


With Features selected

- Number of estimators =1000
- Learnig rate = 0.1
- Accuracy = 0.810
- AUC: 0.811 > 0.5

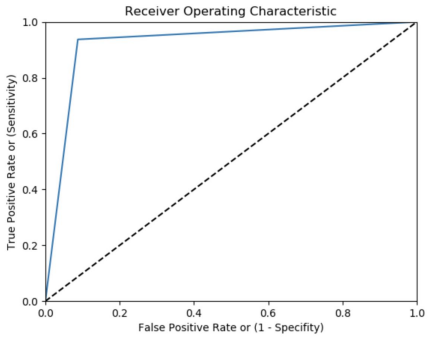
The model with features selected have better AUC value(0.811)

Random Forest



Without Features selected

- Number of estimators =100
- Accuracy = 0.93
- AUC: 0.927 > 0.5



With Features selected

- Number of estimators =100
- Learnig rate = 0.1
- Accuracy = 0.93
- AUC: 0.925 > 0.5

The model without features selected have better AUC value(0.925)

Here is the comparison results between training with features and without features selected on these models.

Models	K Nearest Neighbor Classifier (KNN) with features selected		Random Forest with features selected		AdaboostClassifierwith features selected	
	Without FS	With FS	Without FS	With FS	Without FS	With FS
AUC	0.700	0.691	0.920	0.925	0.800	0.811
Precision	0.670	0.680	0.920	0.920	0.770	0.790
Recall	0.810	0.740	0.920	0.940	0.890	0.850
Accuracy	0.700	0.690	0.930	0.930	0.800	0.810
F1-score	0.740	0.710	0.920	0.930	0.820	0.820

For the KNN algorithm, we found the F1-score of the model without features (0.74) is higher than the model without features selected(0.71). For the Random Forest algorithm, we found the F1-score of the model with features (0.73) is higher than the model without features

selected(0.72).Adaboost Classifier havs similar f1-score between with features selected and without features selected. The results of KNN and Adaboost Classifier models with features selected didn't improve because our dataset is quite general. Therefore, although these selected features are more important, it cannot predict hit or non-hit songs. In the future, we will classify the dataset by genres first and train the model with features selected later.

6.3 User Interface

After we decided on using the KNN, Adaboost and Random Forest models, we created the user interface with the Streamlit library. The trained models are saved as a .sav file that allows ease of access to it for testing in the front-end. Hence, the web page allows users to input a song of their choice and display whether our models thinks it will be a hit song. In addition, a few plots are shown in the webpage for data visualisation.

The user interface and results of out models can be downloaded in our github repository: https://github.com/chenyan1998/50.038_Computational_Data_Science_Project.git

7. Conclusion & Future work

In conclusion, we got a very high F1-score and accuracy in the Random Forest model. To some extent, it's enough to predict hit songs in general. However, we also need to do further work to predict songs better. In this project,we found that it's quite difficult to define non-hit songs due to the ever-changing trends in hit music. In addition, the current model that we have is very general. Hence, it would be better if we classify the hit and non-hit songs more thoroughly such as using more data to reduce variability of results, select good quality data and consider more factors that affect prediction results. There are two ways we thought of for better classification of hits and non-hits songs in the future: Data based and Time based.

Data based

a. Split songs to different genres

   The features between hit and non-hit from different genres may be on different ends of a spectrum, which suggests that the song features may contrast vastly from one another. Therefore, we may split hit or non-hit songs into their genre and build a model for each genre to perform error analysis for some successful algorithms to select the greatest influence features to train our model.

   It would be better to classify the songs into different genres before we model them as some genres are distinct from one another, hence modelling them together might make our model confused about the features of a hit song. Generally, different genres have different features that make them become hit songs. So, it's best to narrow down the scope to cater for each genre.

b. Get data from the audio itself

   From another perspective, we can also predict hit or non-hit songs by audio itself. For example, the tone, pitch, intonation, or music effect of the song. It is also meaningful to define hit-songs lyrics which is mentioned by many papers regarding Hit Song predictions. Applying more types of high level features for training makes even better predictions.

Time based

a. Social media trends

Trends of hit music change over time with the effect of social media trends. We may focus on finding a way or dividing the data into several subsets of year periods for training models in the future.

b. Popularity of the artist

When popular artists produce a new song or new album, their fans will support them no matter the quality of the song. Hence, a model can be made to detect if the song is a hit song after the first hype of the song. Trying to use combined audio itself with artist past-performance may help us to explain a majority of the variance of the data.