Introduction

In this project, the objective is to predict if a music track will be in list of Billboard Top 100 Hits based on the previous achievement.

About Dataset

The project has training and test datasets. The training data (MusicHitTrainData.csv) consists of around 1900 songs from the 2000s taken from the Million Songs Dataset created by Columbia University cross-referenced with the list of Billboard Top 100 Hits. Each song is characterized by audio features (Artist, Track, Year, and features: PreviousHit, Danceability, Energy, Key, Loudness, Mode, Speechiness, Acousticness, Instrumentalness, Liveness, Valence and Tempo) extracted from the Spotify API. There is no missing data. The feature Previous Hit is 1 if the artist previously had a Billboard Top 100 Hit between 1986 and 2010, and 0 otherwise. Details of all other features can be found in the Spotify API documentation. The tracks are labeled 1 or 0: 1 indicating that the song was featured in the Billboard Hot 100 (between 1991 and 2010) and 0 indicating otherwise.

Using the four classifier models with the optimum parameters selected, predictions are made on the test set (MusicHitTestData.csv). In the test data, identifying information such as the names of artists, songs and song years are removed.

Preprocessing

The following preprocessing steps are applied on the datasets before any implementation.

1) StandardScaler() is used to make feature values in the same scale.
2) PCA analysis is implemented as feature reduction and finally top 10 most important features out of 12 is selected in order to save more than 95% of data variance percentage.
3) Train dataset is split into train and test sets by a fraction of 0.2.
4) For the final test data imported from 'bb_2000s_test.csv', before any prediction, the same preprocessing steps are applied as train dataset.

Algorithm Implementation

After preprocessing, four classifiers including SVM, Random Forest (RF), KNeighbors (KNN) and Linear Regression (LR) are applied on the dataset and their capabilities in prediction is analyzed for different values of hyper parameters. In SVC log(Gamma) value is chosen from -3.5 to 1 in steps of 0.25, in LR, log(C) value is chosen from -4 to 2 in steps of 0.5, in KNN the n_neighbors is chosen from {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25} set and in RF n_estimators is chosen

from {1, 2, 3, 4, 5, 7, 9, 11, 13, 15, 20, 30, 40, 50} set. When the optimum hyper parameter of each classifier is selected based on train and test validation curves, the classifier is implemented on train dataset to predict the classes. Finally, the results of prediction are save into a .csv file.

Project Outline

The report is outlined in 6 parts:

- Part 1: Feature reduction
- Part 2: Random Forest results
- Part 3: KNeighbors results
- Part 4: Logistic regression results
- Part 5: SVM results
- Part 6: Conclusion
- Appendix A: Modules used in coding

Part 1: Feature reduction

The importance of features is analyzed by Extra Trees method for different numbers of trees.

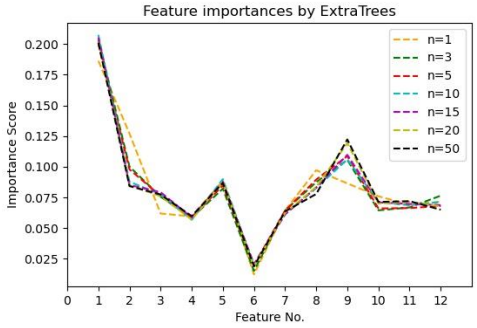

Fig 2: Feature importance by Extra Trees.

The following plot illustrates the importance of each feature in terms of variance percentage by use of PCA before and after selection feature reduction. Variance percentage of more than 95% is saved after elimination of features of “Acousticness” and “Valence”.

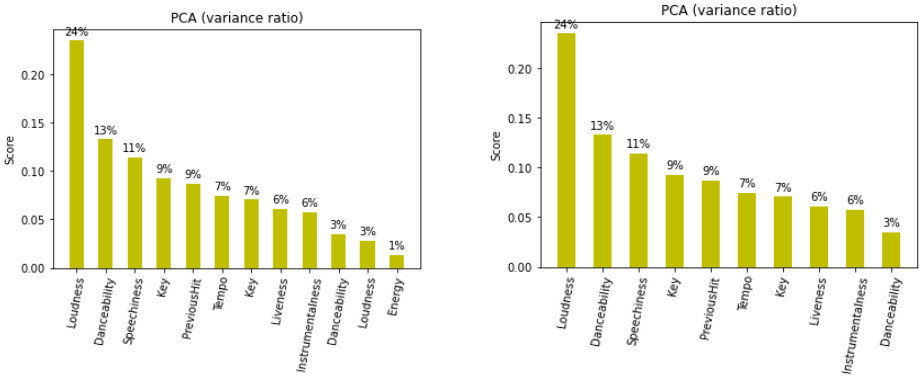

Fig 2: Feature importance plots before and after feature reduction.

Part 2: Random Forest results

After using Random Forest with criterion of 'entropy' and selecting n_estimators from the set mentioned before, it is seen that estimators of almost 20 offers an acceptable accuracy regarding train and test error. Test accuracy of 76% is the best result for this classifier.

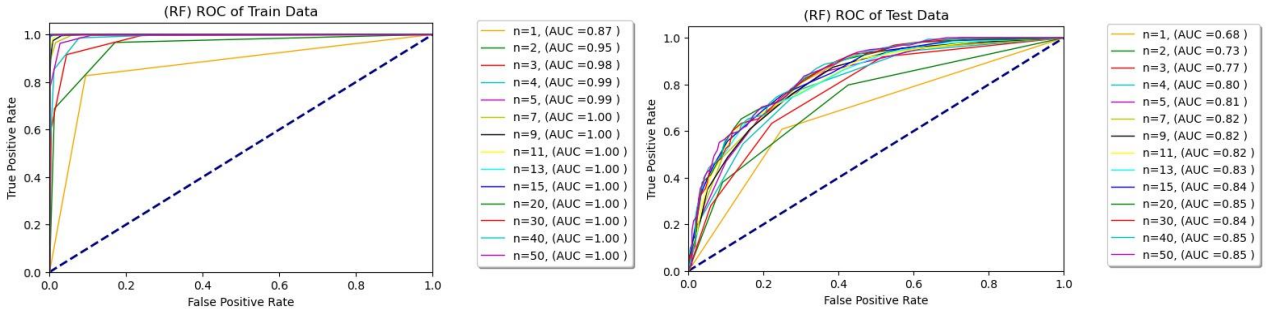

Fig 2: ROC curves for train and test data for different n_estimators of RF.

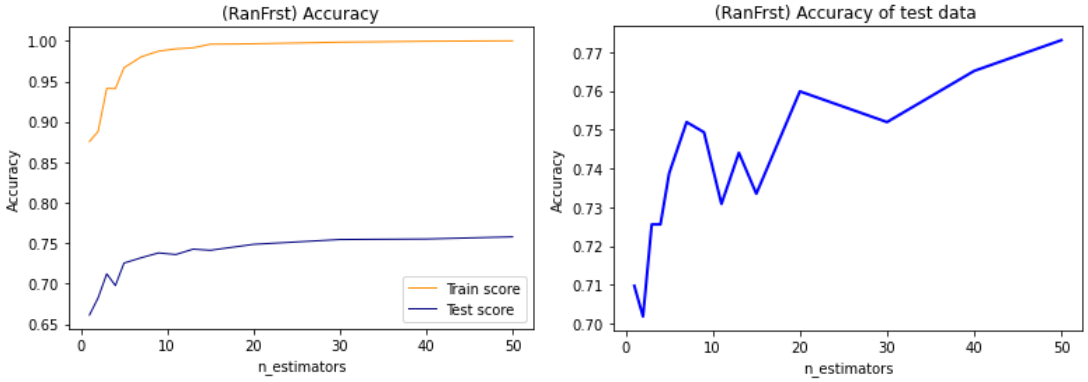

Fig 2: Accuracy score for RF classifier.

Part 3: KNeighbors results

The first two plots of following figures illustrate ROC for train and test data and it is seen that in a specific number of neighbors, ROC accuracy reaches 1.00 which shows the ability of this classifier. Also, increase in number of neighbors doesn't result in underfitting. In this analysis, n_neighbors is selected from the set of numbers mentioned before and distance metric of 'minkowski' with p=2 is used. Changing p=1 didn't affect the results. The optimum classifier can be chosen with n_ neighbors of almost 20 with test accuracy of 76%.

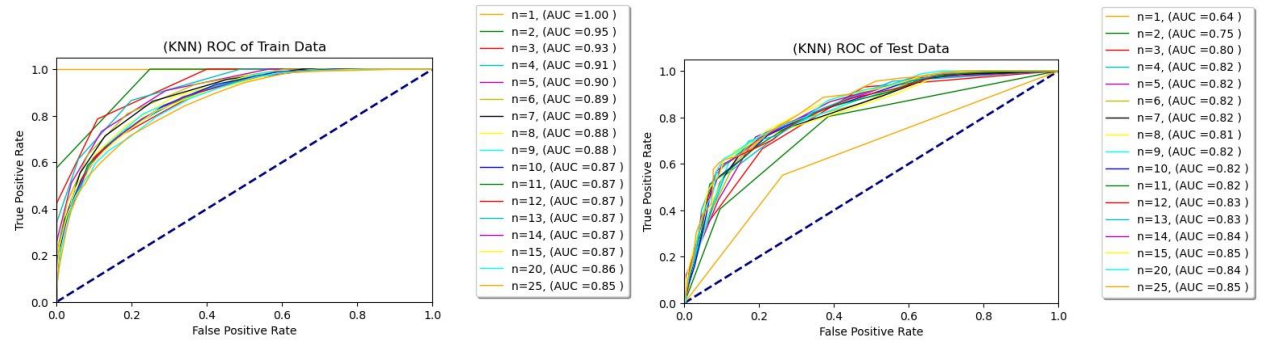

Fig 2: ROC curves for train and test data for different n_neighbors of KNN.

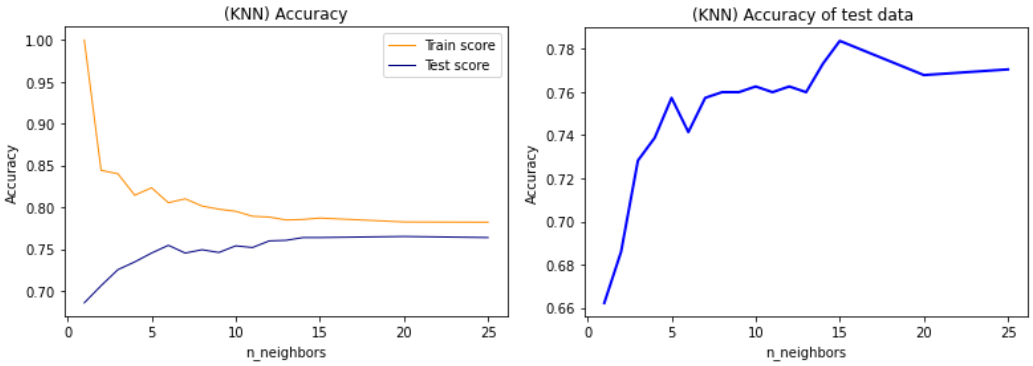

Fig 2: Accuracy score for KNN classifier.

Part 3: Logistic Regression results

By selecting log(C) from set of -4 to 2 in steps of 0.5, the optimum accuracy is gained at log(C) = 0. The following plots illustrate validation and test data accuracy of this classifier. Test data accuracy of 77% is the best that it can do.

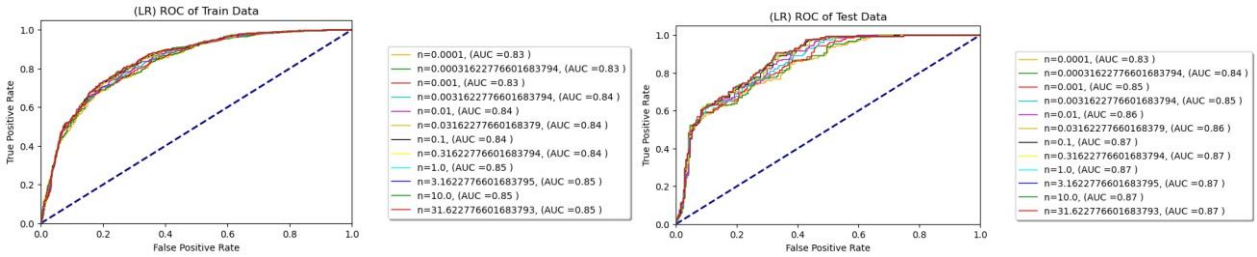

Fig 2: ROC curves for train and test data for different log(C) of LR.

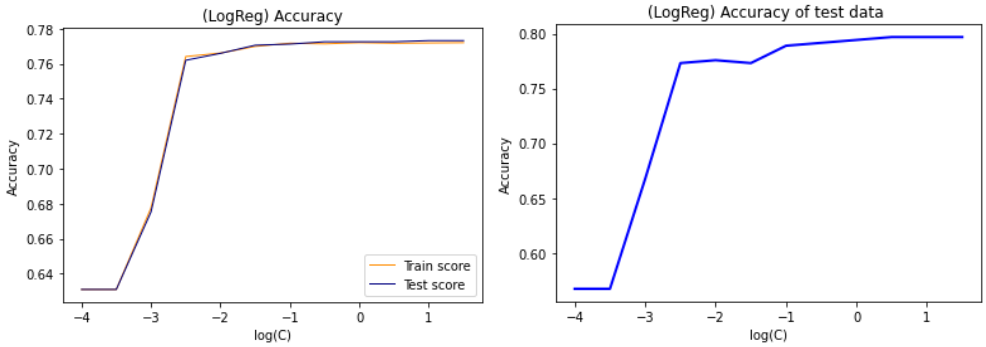

Fig 2: Accuracy score for LR classifier.

Part 3: SVM results

By selecting log(Gamma) from set of -3.5 to 1 in steps of 0.25 with C=1, the optimum accuracy is can be obtained at Gamma = 0.1. The following plots illustrate validation and test data accuracy of this classifier. Test data accuracy of 77% is the best that SVM can do. Only in this classifier, overfitting is observed.

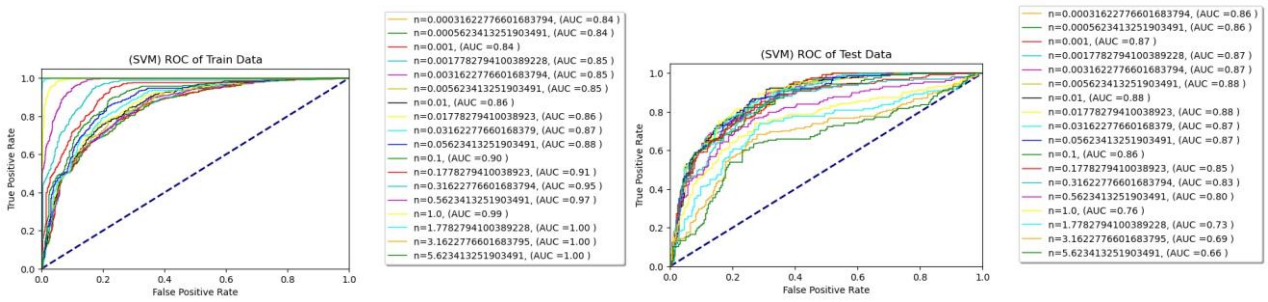

Fig 2: ROC curves for train and test data for different log(Gamma) of LR.

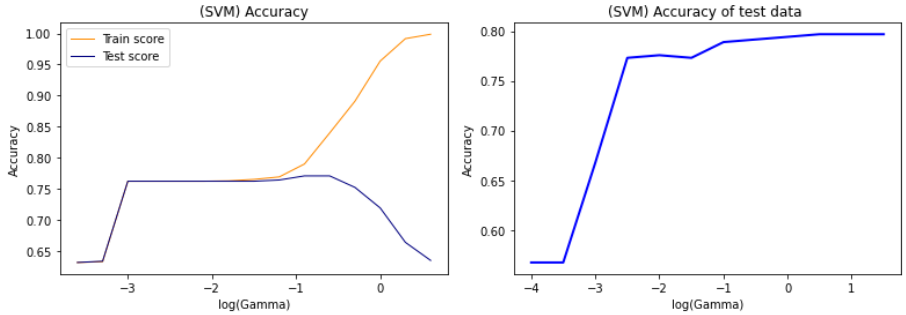

Fig 2: Accuracy score for SVM classifier.

Part 3: Conclusion

By analyzing the results, following points are observed:

1. SVM is the only classifier that encountered overfitting.
2. Random Forest was the fastest classifier among them.

Appendix A: Modules used in coding

The following Modules are used in python codes:

- **Auto Report File:** Automatic Docx report file is created by use of python-docx module.
- **Class:** is created and used for better programming quality.
- **Logging:** an enhanced logger file is created to monitor the progression of codes and log any messages emitted from process.
- **Argparse:** to read the data input by user.
- **Plots** are saved in a separate specific folder for ease of access and analysis.