# ISYE6740 - Computational Data Analytics
# Michael Crist (mcrist6)
# Final Report

## Contents

## 1 Problem Statement

Beginning with Napster in 1999, online music streaming services have gained popularity with astounding speed. The introduction of Napster directly caused the first ever global decrease in album sales, as a new technology disrupted an age-old industry (Yassin, 2019). By the mid-2010s, most music listeners, globally, utilized one streaming platform or another for their music consumption. Spotify has established itself as one of the most popular streaming services. One of the most important services that Spotify offers (as well as most other streaming platforms) is the curation of playlists and song recommendations for its listeners. The goal of my project is to classify the genre of a song, given input feature data for a particular song. This provides value by automating the process of classifying song genre, thus allowing for automatic playlist creation and song recommendation based upon a listener's preferred genre.

## 2 Methodology

The overall goal of my project is to build a supervised learning algorithm to classify a song into a particular genre. As such, genre is the target variable, and all other variables serve as predictor variables. To achieve the best performing classification model, I will break the project into four distinct steps. The first step is data collection and pre-processing. This step involves obtaining the data and handling missing values/outliers. Additionally, this step involves using PCA to project the data onto two dimensions to visualize the data. The second step is to split the data into a training set and a test set. The training set is used to build/train the models, and the test set is used to evaluate the models. The third step is to build several initial models and tune the model parameters to identify the best-performing models. The fourth and final step is to select the best performing model(s), evaluate the models, and draw conclusions.

## 3 Data Collection and Pre-processing

### 3.1 Data Source

I utilized Kaggle to obtain the dataset necessary for my project. Kaggle is an open-source database containing datasets provided by a wide range of companies, government agencies, and individuals from across the globe.

The dataset I chose contains a set of audio features, provided by Spotify, for a large sampling of songs. Each row corresponds to a song, and each column corresponds to a feature of the song. One of the columns is the target variable, which is genre. There are 10 total genres provided in the dataset. The feature columns contain various attributes such as artist, song title, duration, key, tempo, loudness, etc. The total memory size of the dataset is 7.5MB; it contains 50K rows and 18 columns. The dataset I used can be found at the following link: https://www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre

## 3.2 Data Processing

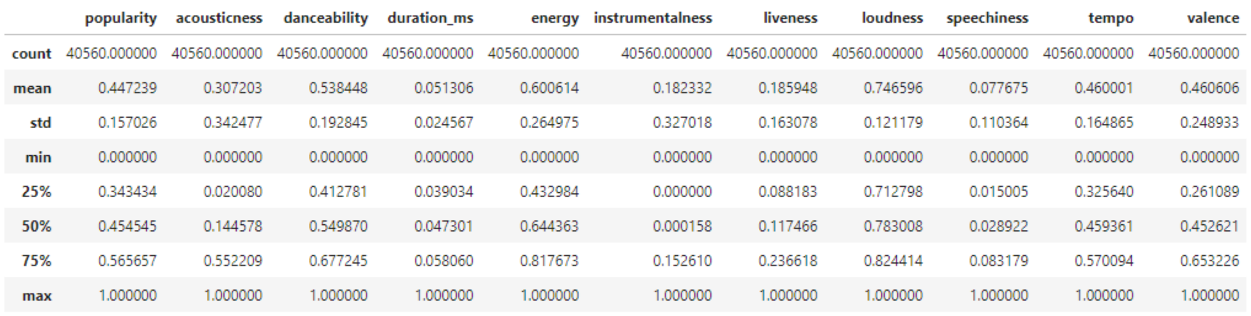
### 3.2.1 Data Cleaning and Outlier Detection

Upon initial investigation of the data, I found that the dataset contained some n/a values for some features. To address this, I dropped all songs containing n/a values from the dataset. After that, I normalized the data by normalizing the values for each feature to fall within the range [0,1]. To do so, I performed the feature-wise operation to subtract the minimum feature value for the dataset from each entry, and divide by the range for that feature. This calculation is shown below.

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

Equation 1. Normalization of feature data

Next, I summarized the normalized feature data:



| | popularity | acousticness | danceability | duration_ms | energy | instrumentalness | liveness | loudness | speechiness | tempo | valence |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 | 40560.000000 |
| **mean** | 0.447239 | 0.307203 | 0.538448 | 0.051306 | 0.600614 | 0.182332 | 0.185948 | 0.746596 | 0.077675 | 0.460001 | 0.460606 |
| **std** | 0.157026 | 0.342477 | 0.192845 | 0.024567 | 0.264975 | 0.327018 | 0.163078 | 0.121179 | 0.110364 | 0.164865 | 0.248933 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.343434 | 0.020080 | 0.412781 | 0.039034 | 0.432984 | 0.000000 | 0.088183 | 0.712798 | 0.015005 | 0.325640 | 0.261089 |
| **50%** | 0.454545 | 0.144578 | 0.549870 | 0.047301 | 0.644363 | 0.000158 | 0.117466 | 0.783008 | 0.028922 | 0.459361 | 0.452621 |
| **75%** | 0.565657 | 0.552209 | 0.677245 | 0.058060 | 0.817673 | 0.152610 | 0.236618 | 0.824414 | 0.083179 | 0.570094 | 0.653226 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

Figure 1. Statistical summary of normalized feature data

At this point, I recognized that the dataset contained some incomplete information. Some of the songs had a 'duration' of zero. As such, I removed the datapoints that had a duration of zero. The next step was to identify and remove outliers. To do so, I first plotted boxplots of the overall distributions for all the feature data:

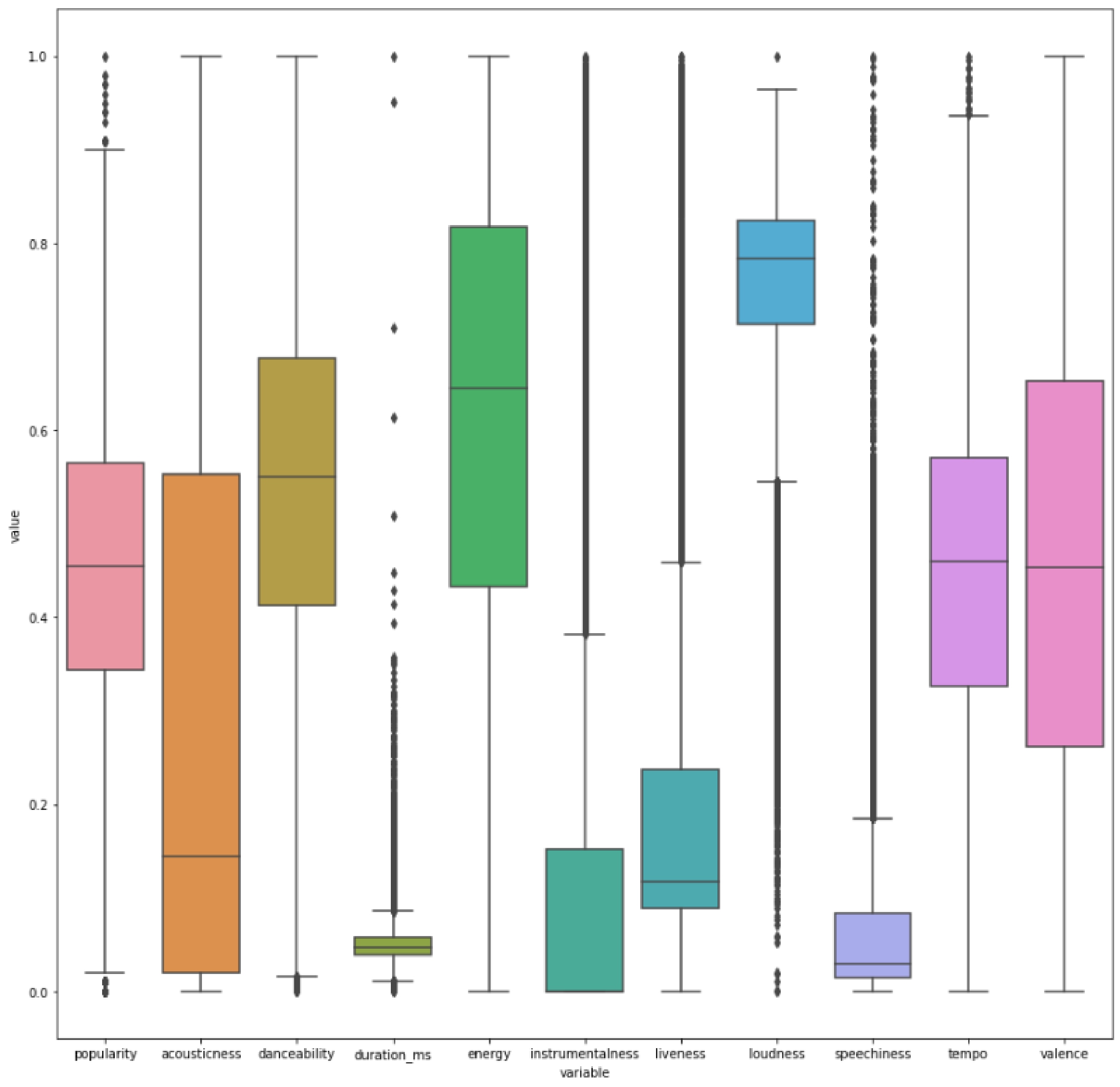


Figure 2. Feature data boxplots

As can be seen in the boxplots above, it appears likely that there are quite a few outliers in some of the variables, namely: duration, instrumentalness, liveness, loudness, and speechiness. To address this, I detected and removed outliers using the inter-quartile range (IQR) method. The IQR method involves first calculating the Q3 quartile and Q1 quartile, which correspond to the 75th and 25th percentile of the distribution for each feature. The IQR is then calculated as the difference between Q3 and Q1. Then, outliers are considered any data points that fall above the upper bound of Q3 + 1.5*IQR, or fall below the lower bound of Q1 – 1.5*IQR (Chaudhary, 2020). However, I could not detect outliers based upon whether a feature value is an outlier based upon the distribution of the entire dataset, because there could be a reason for higher/lower feature values dependent upon the genre label. So, I performed the outlier calculation uniquely for each genre label, and removed all datapoints containing outliers that fell outside the bounds determined by the IQR method.

### 3.2.2 Principal Component Analysis

The next step was to perform principal component analysis (PCA) on the data to project it into two dimensions. In my case, this was done for visualization purposes only, to visualize any patterns in the data in two dimensions. However, when building the models, I decided to utilize the complete feature set to maximize the information obtained from the data.

The intent of PCA is to reduce the dimensionality of a dataset, while minimizing the loss of information. PCA works by evaluating correlation between variables, and combining correlated variables to reduce dimensionality while preserving information. This is done by taking the mean and covariance matrix from the data, then solving for the eigenvectors corresponding to the largest eigenvalues of the covariance matrix. Because my goal was to produce a 2D projection, I only needed to calculate the top two principal components, corresponding to the two largest eigenvalues/eigenvectors. The plot of my 2D PCA-projected data can be found below.

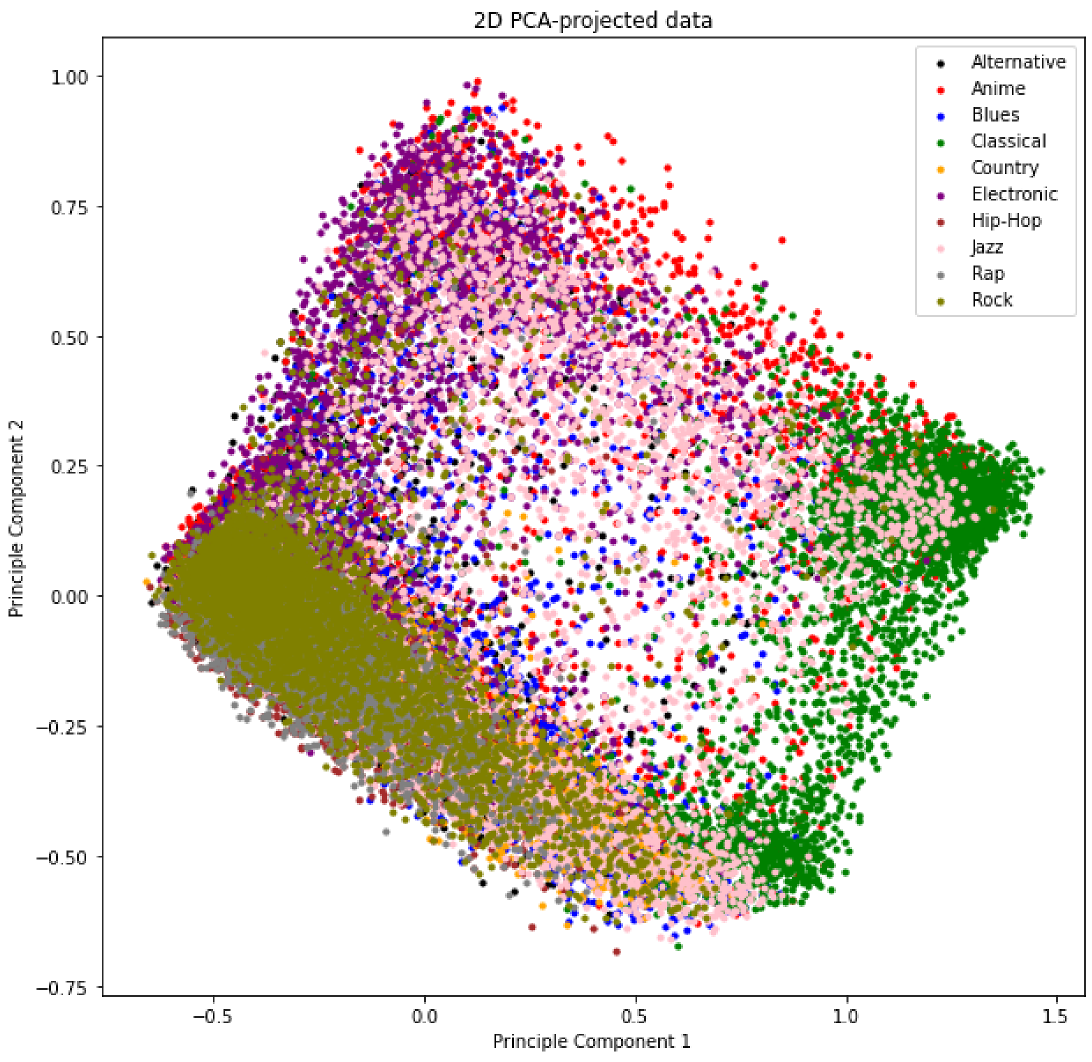


Figure 3. 2D PCA-projection of dataset

As can be seen in the projection above, there are some clear patterns in the data dependent upon the genre label. This is what we would expect to see, and it provides confidence that we can build a successful model to classify the data.

### 3.3 Train/Test Split

The next step was to divide the data into training and test sets. The training set is used to build/train the classification models, and the test set is used to evaluate the models. This is necessary because a model's estimated performance would be far too optimistic if the data used to train the model was also used to evaluate the model. To divide the data into a training set and a test set, I utilized the train_test_split function from the scikit-learn package to randomly assign datapoints into training and test sets. Using this function, I split the data into 80% training and 20% test.

## 4 Model Building and Tuning

### 4.1 Initial Model Building

My approach to model building was to build/train an initial model for several of the algorithms that were covered in this course. The models that I attempted were Naïve Bayes, Logistic Regression, KNN, linear SVM, kernel SVM, neural network, adaboost, classification tree, and random forest classification. Based upon the 2D PCA projection of the data, the decision boundaries for each genre label appeared to be relatively linear, so I felt that any of these models had potential to perform well. However, some of the genre labels had significant overlap, so I knew that the classification would be challenging. Due to the overlap my initial thought was that, while the patterns in the label data appeared to be linear, the most successful model would likely be a complex non-linear algorithm, such as Naïve Bayes, Neural Networks, or Decision Tree.

To build the models, I leveraged the algorithms available in the sci-kit learn package in Python. I trained each model on the training dataset, then computed genre label predictions using the test set, and evaluated the misclassification rate of the predicted labels versus the actual labels to compare model performance. For the initial model building, I did not tune any hyperparameters.

| **Model** | **Misclassification Rate** |
|---|---:|
| Naive Bayes | 48.1% |
| Logistic Regression | 40.1% |
| KNN | 46.6% |
| Linear SVM | 54.1% |
| Kernel SVM | 66.7% |
| Neural Network | 38.0% |
| AdaBoost | 50.9% |
| Decision Tree | 46.7% |
| Random Forest | 37.5% |

Table 1. Misclassification rates of initial models

As can be seen in Table 1, some models performed much better than others. The best performing initial models were logistic regression, neural network, and random forest. This was in line with my expectation because I knew that a complex non-linear algorithm was most likely required, based upon the patterns in the data. However, logistic regression is a linear model, so I was pleasantly surprised with its initial performance. I believe logistic regression performed better than the other linear models because it classifies the datapoints based upon a likelihood for each label, rather than a "hard" prediction.

The worst performing initial models were linear SVM and kernel SVM. Linear SVM's poor performance was not surprising to me, because of the label overlap in the 2D projection of the data. It was clear to me that it would be very difficult to correctly classify the data using a strictly linear decision boundary. However, I was surprised that kernel SVM performed even worse than linear SVM. I believe this is due to the fact that kernel SVM may have a non-linear decision boundary, but it is still a "hard" classifier that is unable to adapt to overlapping labels in the data pattern.

## 4.2 Model Tuning

The next step in the modeling process was to tune the hyperparameters for some of the best-performing initial models. I decided to move forward with only four of the best-performing models: logistic regression, neural network, decision tree, and random forest.

### 4.2.1 Logistic Regression Tuning

To tune the logistic regression model, I plotted the cross-validation score versus the inverse regularization strength. The inverse regularization strength is equivalent to 1/lambda, where lambda is the regularization penalty term. If lambda is too large, the model will underfit the data, whereas if lambda is too small it may overfit the data.

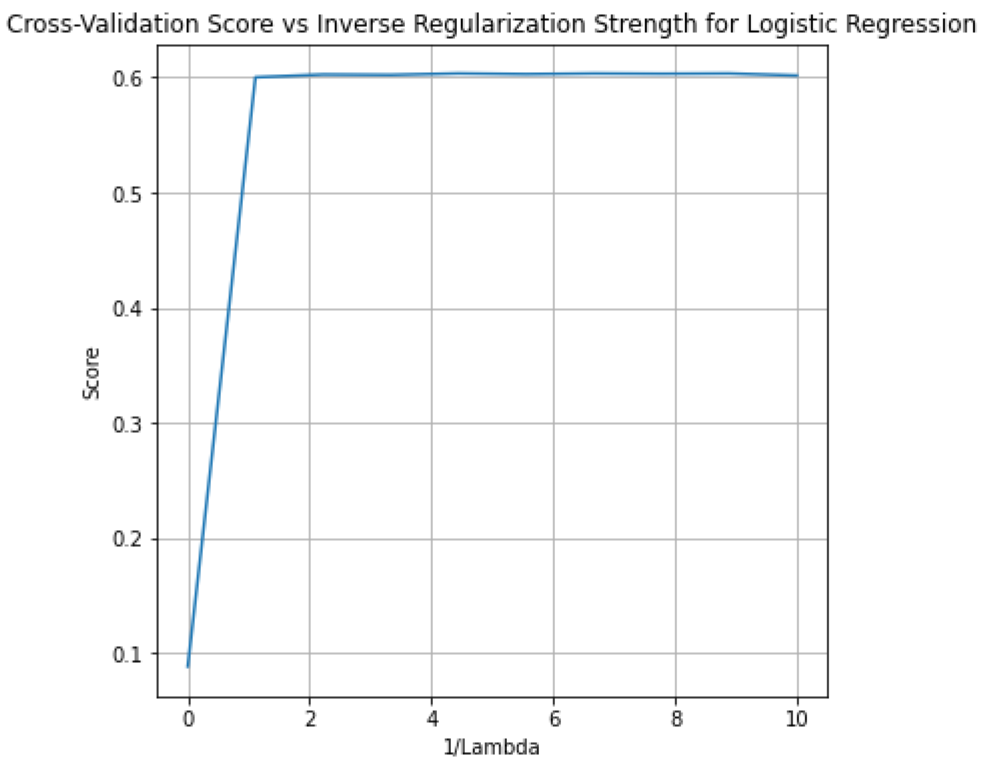


Figure 4. CV score versus inverse regularization strength for logistic regression

As can be seen in Figure 4, the logistic regression model is optimized with an inverse regularization strength >= 2, which corresponds to a lambda value <= 0.5. As such, I reran the logistic regression model with inverse regularization strength set equal to 2.0, and reran the model.

| **Model** | **Misclassification Rate** |
|---|---:|
| Logistic Regression (default) | 40.1% |
| Logistic Regression (inv. reg. strength = 2) | 40.1% |

Table 2. Logistic regression performance with and without tuning

After setting the inverse regularization strength to 2, the results were essentially unchanged, meaning the default logistic regression model was sufficiently tuned.

### 4.2.2 Neural Network Tuning

To tune the neural network model, I plotted the cross validation score versus various configurations of hidden layers in the model. The hidden layer configurations tested were (20,10), (20,20), (30,15), (30,30), (50,50), (100,100), and (100,). The results can be found below.

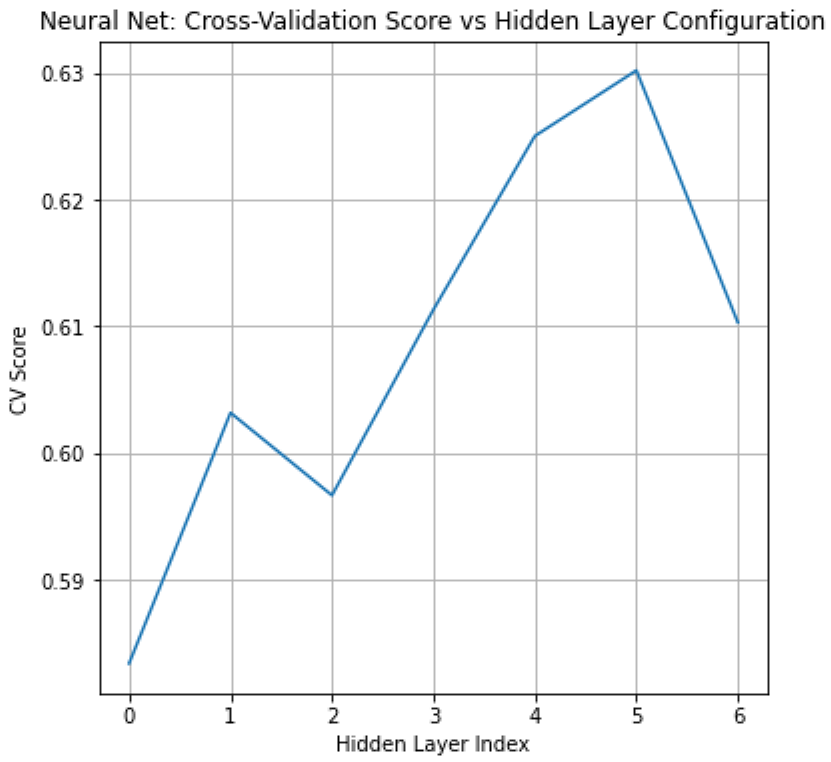


Figure 5. Neural network CV score vs hidden layer configuration

As can be seen in the plot above, the index=5 configuration performed best, which corresponds to the hidden layer configuration (100,100). This is not surprising, because it was the maximum number of neurons tested. In general, we would expect the neural network to perform better as more neurons/hidden layers are added. However, as we add hidden layers, we trade model performance for model efficiency. The model improvement is shown below:

| **Model** | **Misclassification Rate** |
|---|---:|
| Neural Network (default) | 38.0% |
| Neural Network (hidden layers = (100,100)) | 35.1% |

Table 3. Neural network performance with and without tuning

### 4.2.3 Decision Tree Tuning

To tune the decision tree, I plotted the cross validation score versus the “max depth” of the tree. The max depth of the tree corresponds to the length of the longest path from the base of the tree to a leaf.

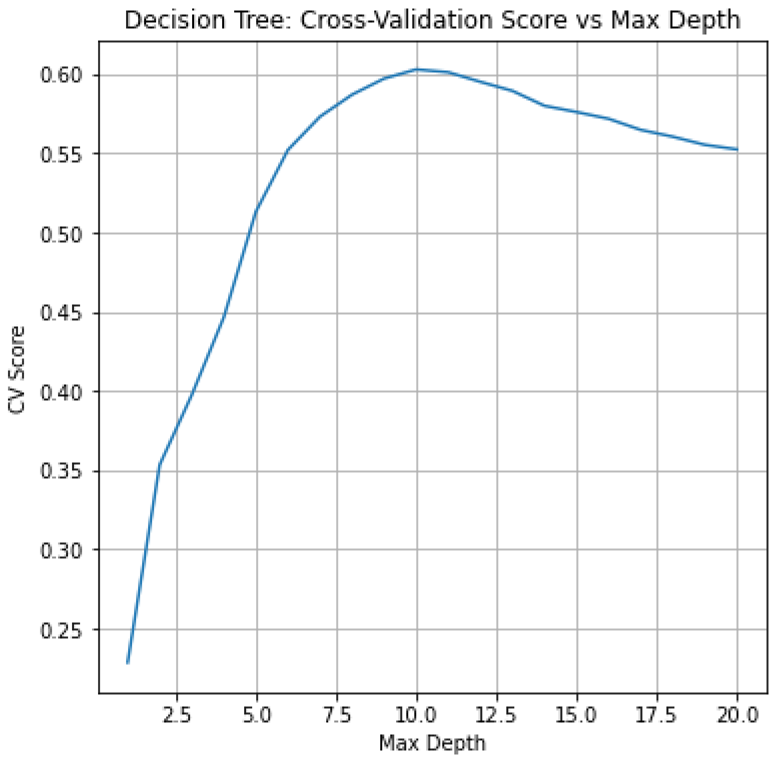


Figure 6. Decision Tree CV score vs max depth

As can be seen above, the decision tree is optimized for a max depth of 10. After tuning, the model improvement can be found below.

| **Model** | **Misclassification Rate** |
|---|---:|
| Decision Tree (default) | 46.7% |
| Decision Tree (Max Depth = 10) | 39.8% |

Table 4. Decision tree performance with and without tuning

### 4.2.4 Random Forest Tuning

For the random forest model, I tuned three different hyperparameters: number of trees, max depth, and minimum samples for splitting. The number of trees corresponds to the number of trees used in the "forest", max depth is the length of the longest path from the base of the tree to a leaf, and minimum samples for splitting is the minimum samples required to split a node in the tree. To tune these hyperparameters, I plotted the out-of-bag (OOB) error versus hyperparameter values. The OOB error is the average error for each observation, among all the trees in the forest.

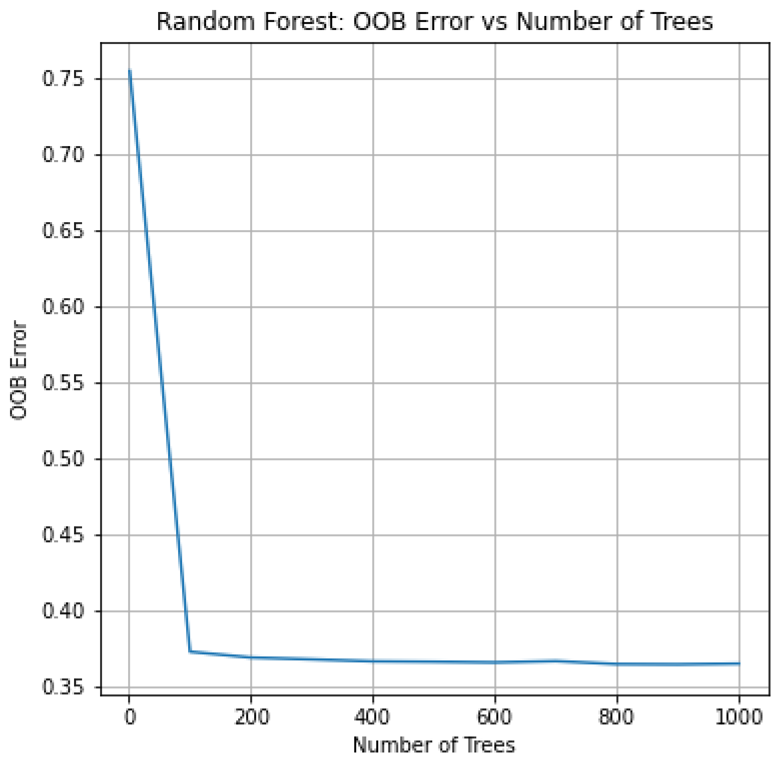


Figure 7. Random Forest OOB Error vs Number of Trees

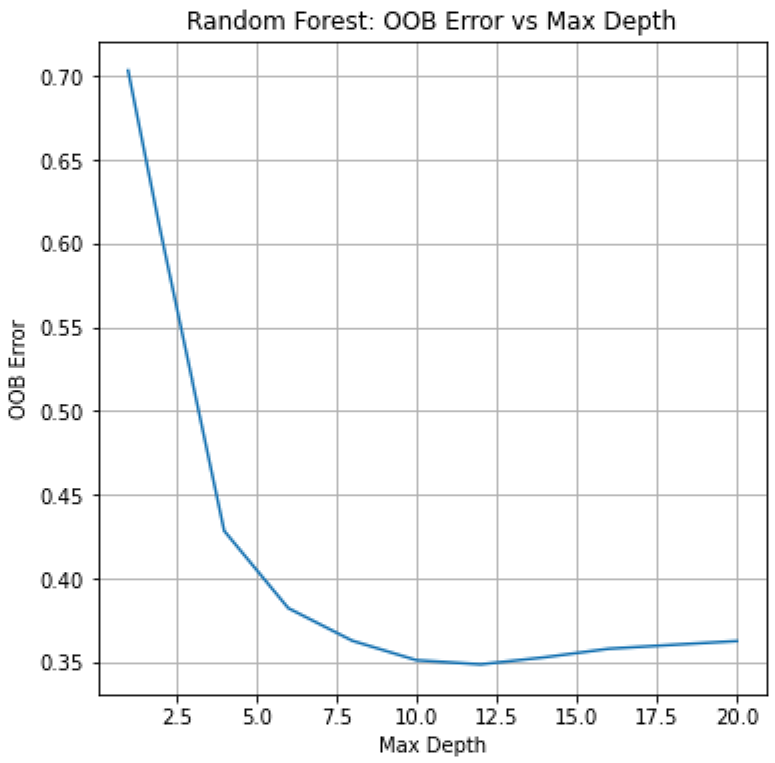


Figure 8. Random Forest OOB Error vs Max Depth

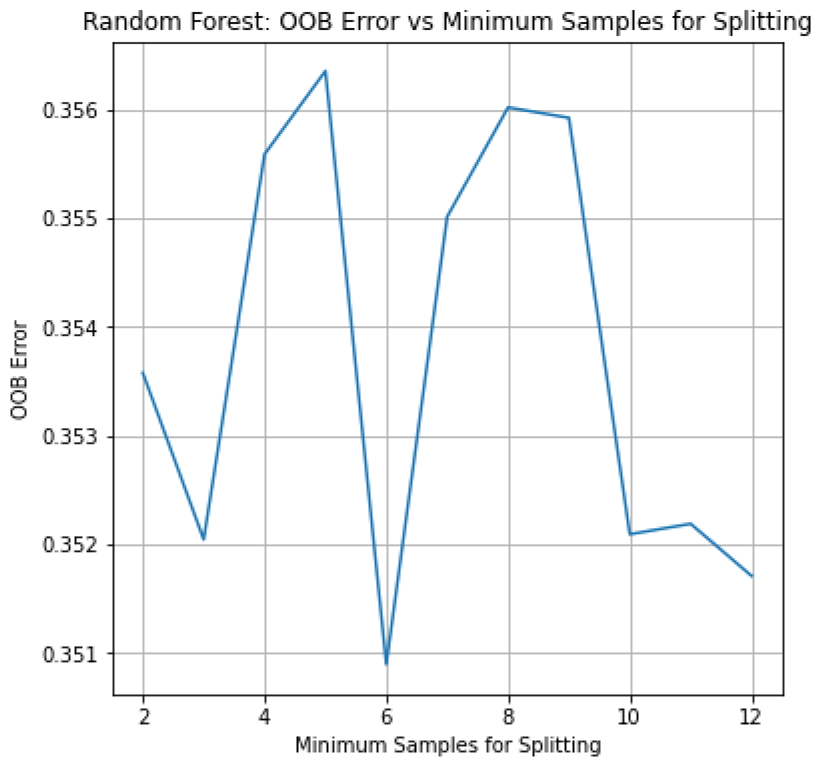


Figure 9. Random Forest OOB Error vs Minimum Samples for Splitting

As can be seen in the plots above, the random forest model is optimized for number of trees = 800, max depth = 12, and minimum samples for splitting = 6. After tuning, the improvement in model performance can be found below.

| **Model** | **Misclassification Rate** |
|---|---:|
| Random Forest (default) | 37.5% |
| Random Forest (after tuning) | 35.2% |

## 5 Evaluation and Final Results

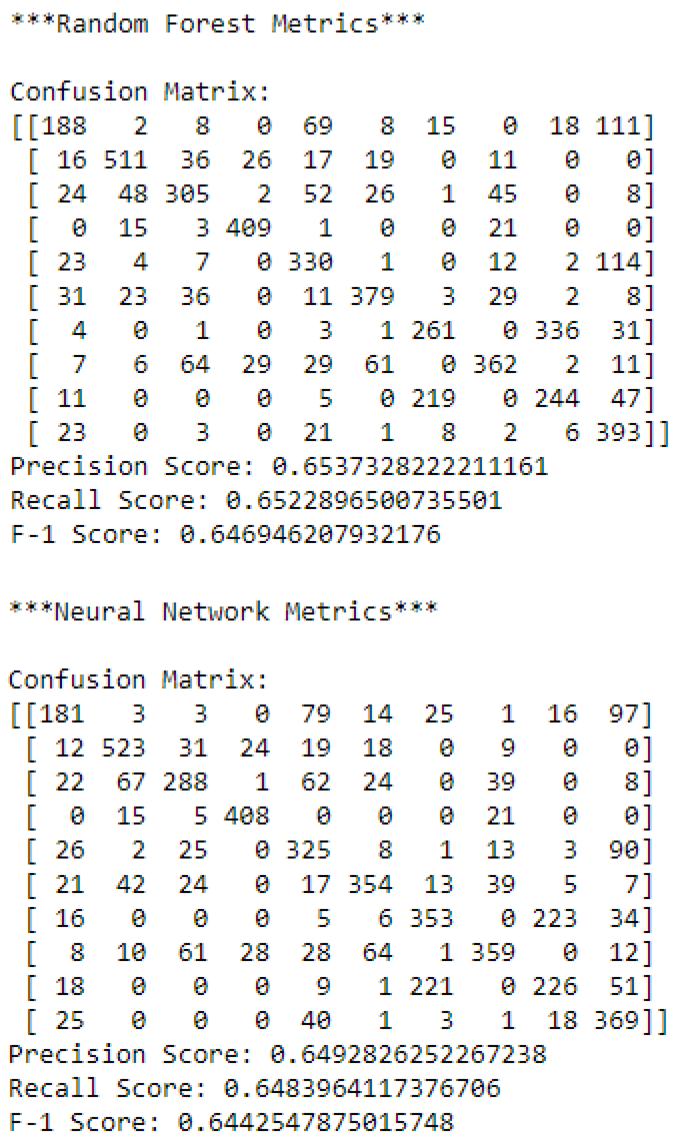
### 5.1 Evaluation

After tuning the models in section 4, I obtained the final misclassification rates below for each model.

| **Model** | **Misclassification Rate** |
|---|---:|
| Logistic Regression | 40.1% |
| Neural Network | 35.1% |
| Decision Tree | 39.8% |
| Random Forest | 35.2% |

Table 5. Final model misclassification rates

As is clear from the table above, the best performing models were neural network and random forest. Next, I obtained some more detailed metrics for those models, including: confusion matrix, precision score, recall score, and F1 score. The precision score is the ratio of TP / (TP + FP), the recall score is the ratio of TP (TP + FN), and the F1 score is the harmonic mean of precision score and recall score, calculated by (2*precision*recall) / (precision + recall).



```
***Random Forest Metrics***

Confusion Matrix:
[[188   2   8   0  69   8  15   0  18 111]
 [ 16 511  36  26  17  19   0  11   0   0]
 [ 24  48 305   2  52  26   1  45   0   8]
 [  0  15   3 409   1   0   0  21   0   0]
 [ 23   4   7   0 330   1   0  12   2 114]
 [ 31  23  36   0  11 379   3  29   2   8]
 [  4   0   1   0   3   1 261   0 336  31]
 [  7   6  64  29  29  61   0 362   2  11]
 [ 11   0   0   0   5   0 219   0 244  47]
 [ 23   0   3   0  21   1   8   2   6 393]]
Precision Score: 0.6537328222211161
Recall Score: 0.6522896500735501
F-1 Score: 0.646946207932176

***Neural Network Metrics***

Confusion Matrix:
[[181   3   3   0  79  14  25   1  16  97]
 [ 12 523  31  24  19  18   0   9   0   0]
 [ 22  67 288   1  62  24   0  39   0   8]
 [  0  15   5 408   0   0   0  21   0   0]
 [ 26   2  25   0 325   8   1  13   3  90]
 [ 21  42  24   0  17 354  13  39   5   7]
 [ 16   0   0   0   5   6 353   0 223  34]
 [  8  10  61  28  28  64   1 359   0  12]
 [ 18   0   0   0   9   1 221   0 226  51]
 [ 25   0   0   0  40   1   3   1  18 369]]
Precision Score: 0.6492826252267238
Recall Score: 0.6483964117376706
F-1 Score: 0.6442547875015748
```

As can be seen above, the random forest and neural network models performed roughly the same, with F1 scores of 64.7% and 64.4%, respectively.

## 5.2 Conclusions

As is clear from the results obtained in section 4 and 5.1, the models that performed best for music genre classification on this dataset were random forest and neural networks. This conclusion was derived by starting with a wide array of models, and eliminating models until the best performers emerged. The models were significantly improved by tuning the model hyperparameters in section 4. Ultimately, the random forest and neural network models achieved a misclassification rate of about 35%. This is still relatively high, but it is the best performance I was able to obtain. I believe the reason the misclassification rate is somewhat high is because there are 10 different genre labels, which is much more complex than classifying between only 2 labels. Additionally, I suspect that many songs, although classified as different genres, possess very similar attributes. If this is the case, even though some songs were classified as an incorrect genre, they are still classified under the same label as similar songs. So, this model would still provide significant value to a streaming platform, because it could be used to autonomously build playlists containing similar songs.

## 6 Bibliography

Bhardwaj, Ashutosh. "Silhouette Coefficient : Validating Clustering Techniques." *Towards Data Science*, 27 May 2020, https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c.

Blaufuks, William. "FAMD: How to Generalize PCA to Categorical and Numerical Data." *Towards Data Science*, 28 Mar. 2022, https://towardsdatascience.com/famd-how-to-generalize-pca-to-categorical-and-numerical-data-2ddbeb2b9210.

Bonthu, Harika. "KModes Clustering Algorithm for Categorical Data." *Analytics Vidhya*, 13 June 2021, https://www.analyticsvidhya.com/blog/2021/06/kmodes-clustering-algorithm-for-categorical-data/.

Chaudhary, Shivam. "Why '1.5' in IQR Method of Outlier Detection?" *Medium*, Towards Data Science, 26 Aug. 2020, https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097.

Yassin, Diana. "A Brief History of Streaming Services." *The Michigan Daily*, 5 Dec. 2019, https://www.michigandaily.com/music/brief-history-steaming-services/.