Spotify User Classification

Alex Somer, Jordan Klein, Ryan Simpson
November 19th, 2023
CS4755 - Final Report

Table of Contents

Introduction

Our group is building a classifier model that aims to identify which of our own personal music libraries a given song comes from. The motivation behind this project is to apply the knowledge gained throughout this semester to an attainable and accessible problem that interests all members of our group. All our group members enjoy listening to music, data exploration, and have an interest in machine learning - so assembling a classification model to test our knowledge of the material taught this semester proves to be a challenging, yet attainable goal as a final project. To build this classifier, we are using a variety of songs from the Spotify music libraries of each of our 3 team members as training data. Each of the songs in our dataset has been classified with one of our group members' names to signify which library it came from. Using our names as target categories, we train the model on all the features of our dataset to learn correlations between each of our personal music tastes.

Objectives

The main goal of this project is to create a model that can predict which of our 3 team members' libraries a specific song has come from with greater than 33% accuracy (better than random guessing). We have a selection of attributes of each song to use as features to build the model on. Once trained, the model should be able to analyze completely new songs with their own attributes extracted using the Spotify API's functionality and return a classification for which of our group members would be most likely to listen to it. Another goal of ours is to create a front-end interface which will allow users to search for a song and then view its predicted classification.

The Data

We collect the data for this project from a website called Sort Your Music that displays the attributes of songs from Spotify. We are able to log in with our Spotify credentials and collect attributes of songs from our own personal playlists. We copy the data into a google sheets document and add an extra column with our name in it to signify which person had that song on their playlist. The attributes available are Title, Artist, Release, Length, Pop., A.Sep, Rnd, BPM, Energy, Dance, Loud, Valence, Acoustic. Title is the name of the song. Artist is the name of the recording artist. Release is the date the song was released. The length is how long the song is in the format m:ss. Pop is the popularity on a scale of 1 to 100. A.Sep maximizes the artist separation in the set. Rnd is a random number that can be used for the purpose of shuffling the playlist. The following attribute descriptions are provided by the "Spotify for Developer" Documentation found here.

.

- **BPM [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."
- **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."
- **Dance [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."
- **Loud [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 typically range between -60 and 0 db."
- **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)."
- **Acoustic [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."

We use the attributes from the songs to build a model that can correctly classify the songs into who listened to them. This is a model that we could potentially use to take in data on new songs and predict who out of the 3 of us might like to listen to it. This task is best achieved using machine learning methods because of the numerous attributes each song has. Machine learning gives us the ability to easily handle a model with many attributes and classify each song using computational power as opposed to completing calculations by hand.

Data Preprocessing

Data preprocessing is a crucial step in the model creation process. Raw data can be filled with missing or incorrect values, inconsistent scaling of features, and formatting errors, all of which need to be addressed before fitting a model in order to ensure accurate and effective results. Without preprocessing, a model could produce completely meaningless results, which is why it is so important to make sure all of the data is accurate, appropriately scaled, and properly formatted.

To begin preprocessing the data we load our raw CSV dataset (OriginalRawData) into a Pandas DataFrame object hosted in a Google Colab session. In its raw state we have 333 songs classified as “Jordan”, 128 songs classified as “Ryan”, and 1022 songs classified as “Alex”. Upon first look at the ranges of each attribute, it is clear that there are some misrepresented values in the “Release” category because some of the songs are listed as being released in 1905 when they were not. For these rows we have to manually adjust the values for that attribute using Pandas methods. No other attributes contain any incorrect values. We then convert our non-numerical columns to numerical values. “Release” and “Length” are both converted to integer values as they were originally stored as String data types. We truncate “Release” to just the year the song was released and convert to an integer. We convert “Length” from a minute:second format to just the total length in seconds. The DataFrame is now saved as another CSV file named FullCleanedData.csv.

Now we check to make sure that there are no duplicate songs with the same classification (for example if the same song is classified as “Jordan” twice), of which there are only a few. We remove the extra instances and keep the first instance of each duplicate. We decided to keep any songs that overlapped between classifications in order to be transparent about our overlapping music tastes. We want to ensure class balance. Since the “Ryan” class has the least amount of data points with only 128, we randomly select 128 data points from the “Jordan” and “Alex” classes so we have a dataset that is evenly distributed amongst the three classes. The final number of rows in our dataset is 384. This DataFrame is now saved as a CSV named FinalCrossBalData.csv.

At this point, the columns “Rnd”, “A.Sep”, and “Title” are all dropped from the DataFrame as they do not provide any meaningful contribution to classification. “Title” was considered for one hot encoding but was ultimately decided against because if title was a meaningful attribute for classification, it would be because of its sentiment. Assessing a title’s sentiment would be a whole project on its own so we decided not to do it, but it is a topic for further research. To properly preprocess the remaining categorical variables, “Artist” and “Classification”, one hot encoding is applied using Pandas “get_dummies” function. Principal component analysis (PCA) with two components is performed between the one hot encoding of “Artist” and “Classification” to allow for graphing with the categories still labeled. The resulting PCA graph is displayed below (Fig. 1). The final, fully preprocessed data is now saved to a CSV named FinalData.csv.

The Keras artificial neural network also requires additional preprocessing before data can be loaded into the model for training. After being loaded into a Pandas DataFrame object from the saved CSV file, the data is converted into a Numpy array of dimensions 384 by 196. This array is then sliced into its constituent ‘x’ and ‘y’ components at the correct column index. The ‘x’ component of the data contains all the columns of the CSV data file containing numeric values pertaining to each metric and is of shape 384 by 192. The ‘y’ component consists of the three columns containing the true classifications of each song in our dataset. This data is then split in test and training sets and fed into the model.

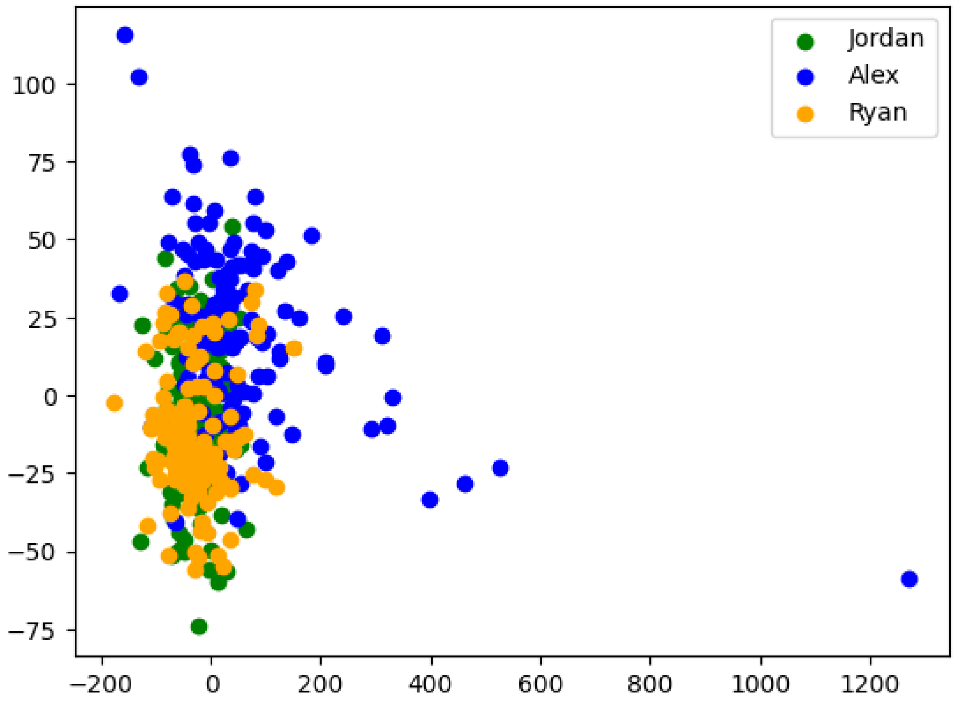


Figure 1 - Principal Component Analysis of the preprocessed dataset using two components and colored according to classification.

Additional CSV files are saved before and after one hot encoding classifications as FinalDimReducedData.csv and FinalDimReduceClassificationdData.csv respectively. Further figures illustrating the distributions of the data after each step are provided in the Google Colab file available at: DataPreprocessingFinal.ipynb . All datasets are also accessible at the following link: Data Folder.

Data Visualizations

Here we have provided some visualizations to help better show the distributions of song attributes by classification.

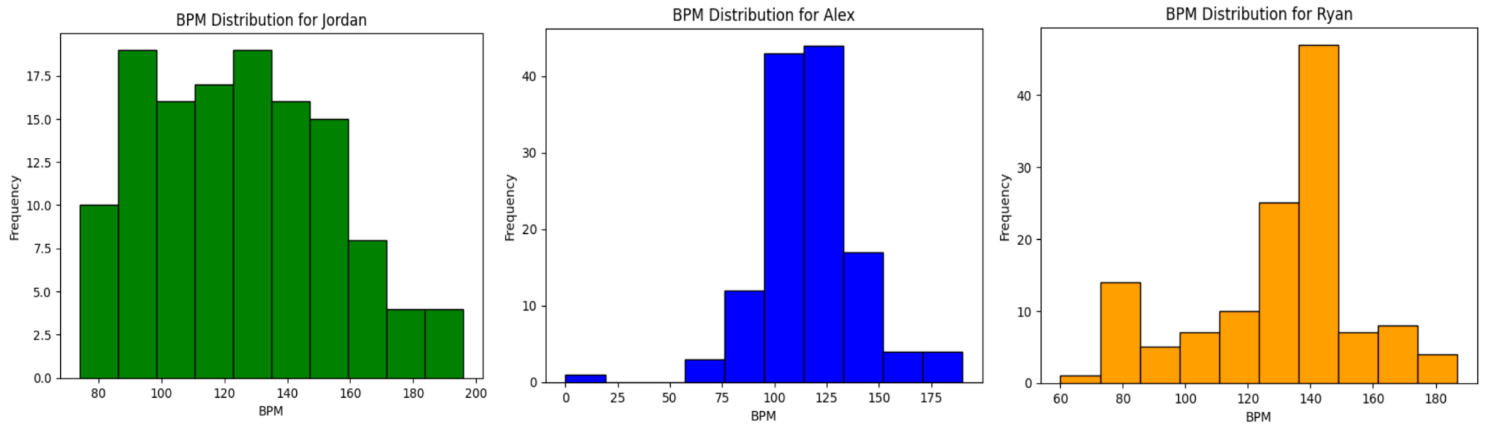


Figure 2 - Distributions of BPM by Class

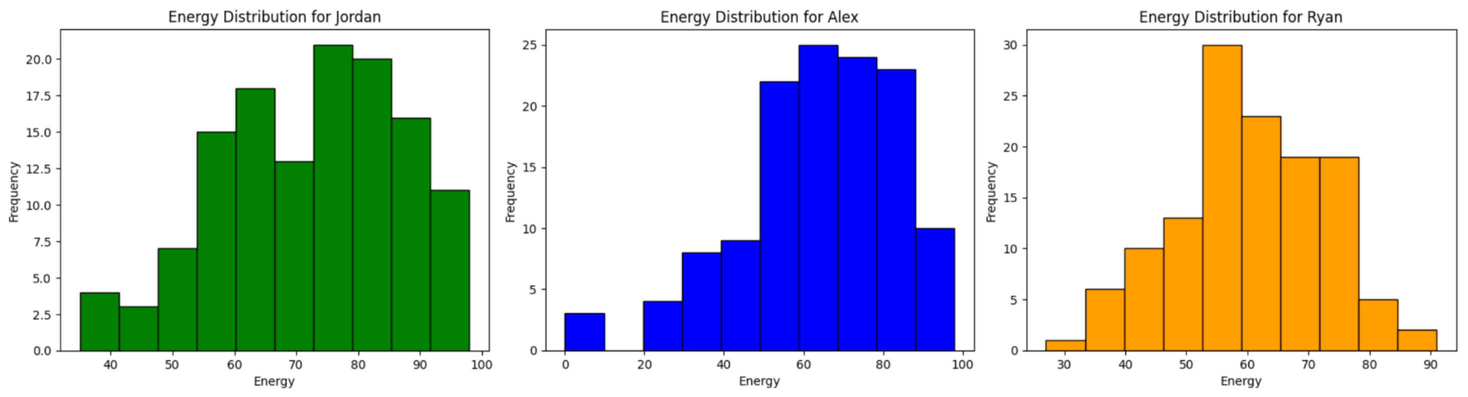


Figure 3 - Distributions of Energy by Class

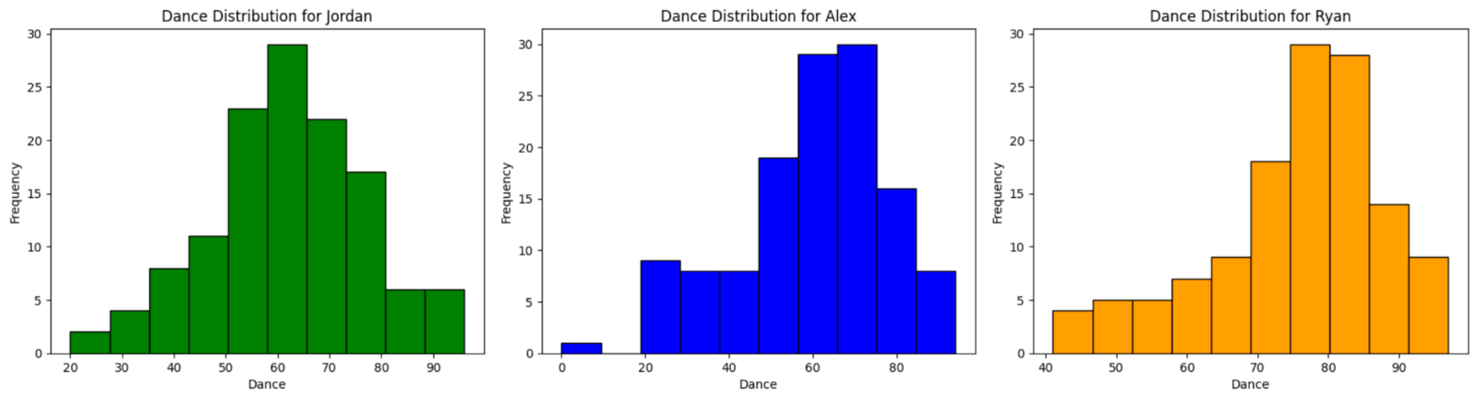


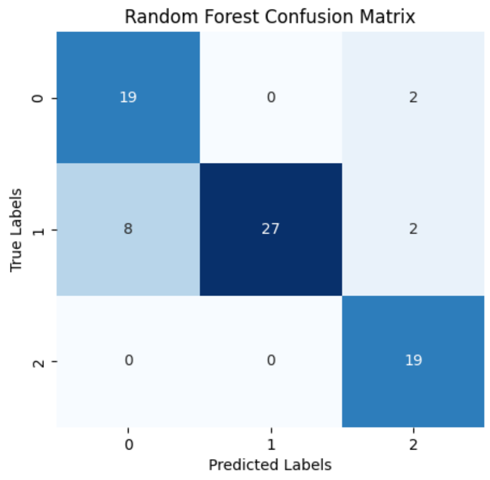
Figure 4 - Distributions of Dance by Class

Model Selection

Since our project is about a multiclass classifier (classifying songs as “Alex”, “Jordan”, or “Ryan”) we try the following models: Random Forest classification, Naive Bayes classification, K-Nearest Neighbors classification, Decision Tree classification, and classification with a neural network using Keras.

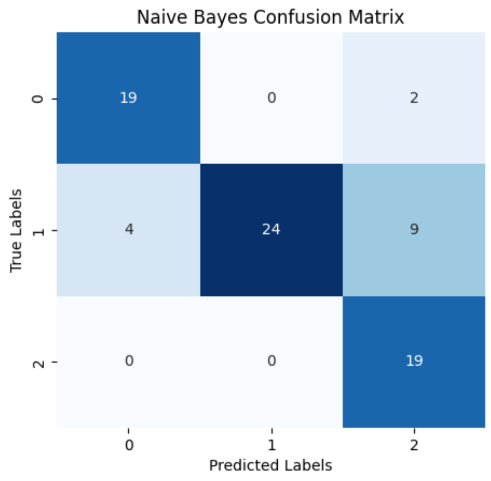
Random Forest

For Random Forest Classification we use GridSearchCV with the parameter grid: 'n_estimators': [25, 50, 100, 150, 200, 250, 500], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8], 'criterion' : ['gini', 'entropy']. The best parameters for this model from GirdSearchCV are 'criterion': 'gini', 'max_depth': None, 'max_features': 'log2', 'n_estimators': 150. This Random Forest model results in an accuracy of 0.84.



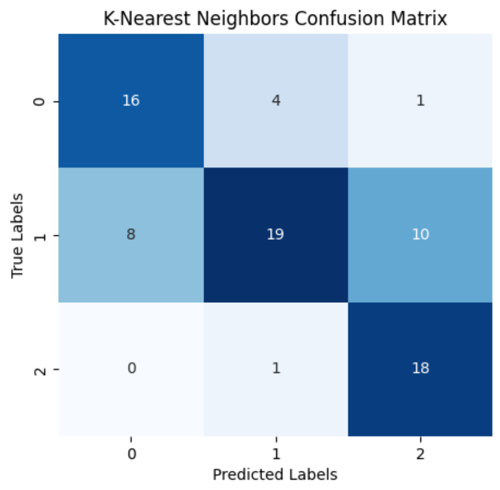
Naive Bayes

For Naive Bayes classification we use GridSearchCV with the parameter grid: 'var_smoothing': np.logspace(0,-9, num=100). The best parameter for this model from GirdSearchCV is 'var_smoothing': 3.51e-06. This resulted in an accuracy of 0.81.



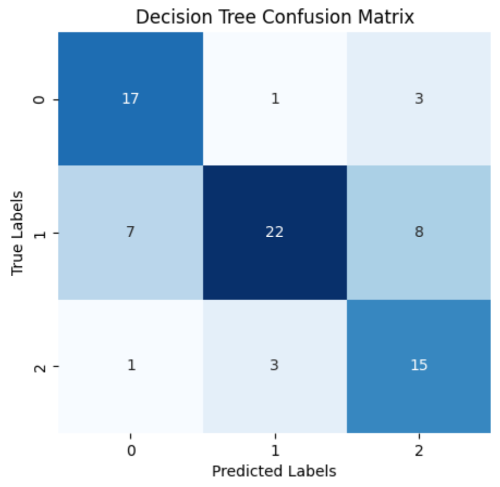
K-Nearest Neighbors

For K-Nearest Neighbors classification we use GridSearchCV with the parameter grid: 'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9]. The GridSearchCV results show the average test scores as 0.668, 0.687, 0.691, 0.710, 0.713, 0.674, 0.694 , 0.704 respectively. Since the score for n_neighbors = 6 is the highest, GridSearchCV outputs that 6 is the best number of neighbors. This results in an accuracy of 0.69 on the test data.



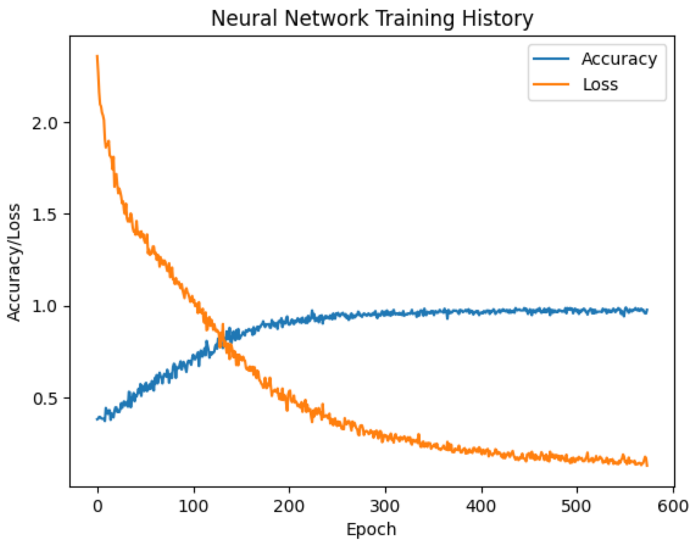
Decision Tree

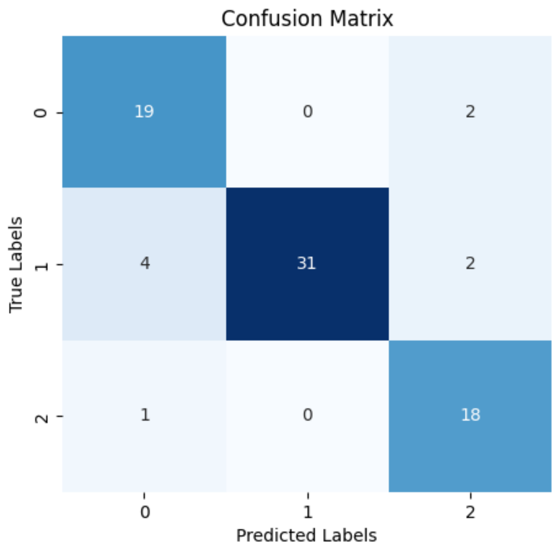
For Decision Tree classification, we use the GridSearchCV with the parameter grid: 'max_features': [None, 'auto', 'sqrt', 'log2'], 'max_depth' : [None, 1, 2, 3, 4, 5, 6, 7, 8], 'criterion' : ['gini', 'entropy']. The best parameters for this model from GridSearchCV are 'criterion': 'gini', 'max_depth': None, 'max_features': None, resulting in an accuracy of 0.7 on the test data.



Artificial Neural Network

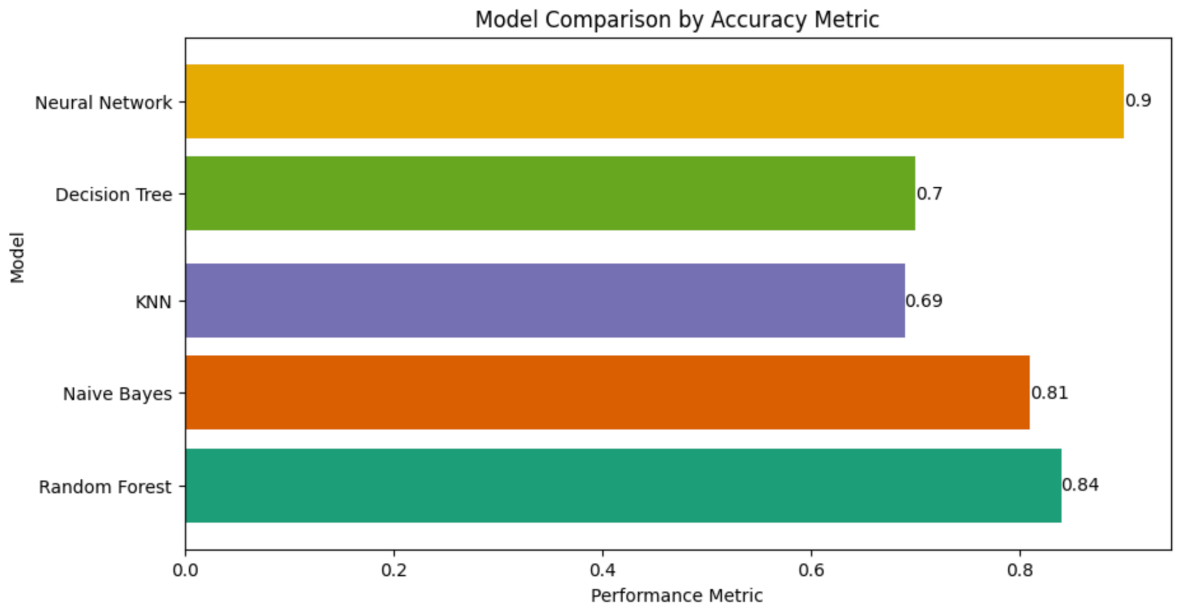
For classification using artificial neural networks with Keras, a sequential model with three dense hidden layers of sizes 192, 32, and 16 are used with ReLU activation functions and L2 kernel regularization. A data normalization layer is also applied directly after the input layer. Each hidden layer is separated by a dropout layer with rates of 0.5 to prevent overfitting. These layers are fed into an output layer of size 3 using a softmax activation function. We evaluate the model using categorical cross-entropy as the loss function and Adam as the optimizer with a learning rate of 0.0001. We train the model for 574 epochs and reach a minimal training loss of 0.1290 with a validation accuracy of 89.61% before early-stopping after 50 epochs without an improvement in validation loss from 0.4677.





Model Comparison

Our artificial neural network model performs the best on our test dataset with an accuracy score of 89.6%, followed by the Random Forest model with an accuracy of 84%. The Naive Bayes model scores slightly lower with an accuracy of 81%. The Decision Tree model has an accuracy of 70%, and K-Nearest Neighbors has an accuracy of 69%.

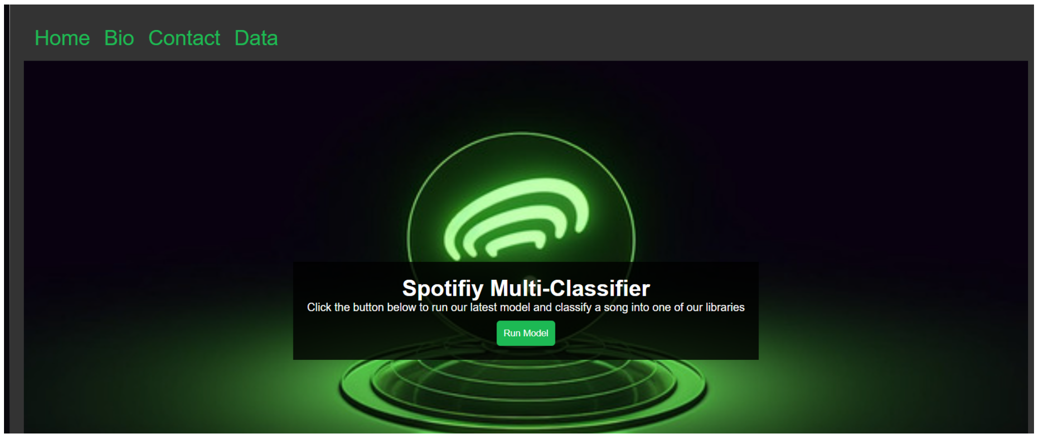


Evaluation

As mentioned, our sequential Keras neural network model performs the best. After manually tuning the hyperparameters of the neural network, it correctly classifies songs to who listened to them (0: Alex, 1: Jordan, or 2: Ryan) 89.6% of the time. The parameters that achieve this accuracy for us are: 574 epochs due to early stopping, batch size of 4, learning rate of 0.0001, and 3 layers of sizes 192, 32, and 16 with ReLU activation functions and L2 kernel regularization. Our output layer of size 3 uses a softmax activation function. Dropout layers with rates of 0.5 are used between each hidden layer to prevent overfitting the dataset. We use categorical cross entropy for our loss function and Adam as our optimizer. We are pleased with our results of 89.6% accuracy, and this far passes our goal accuracy of greater than 33%

Front-End

A secondary goal of our project is to provide a simple and intuitive interface for creating model predictions and exploring the collected data. A front-end web application built on Flask and Angular. Flask is used to provide a Python-based web application capable of interacting with our model and hosting the data. The web-interface allowing for user interaction is implemented using Angular. Angular allows for seamless integration with our backend flask server by utilizing RESTful API calls to get and post information to our flask server. Angular also allows for modular creation which is exceptionally useful for displaying several models using various libraries. This allows users to select the model they wish to run and have results and visualizations displayed in seconds.


Home Bio Contact Data
Spotifiy Multi-Classifier
Click the button below to run our latest model and classify a song into one of our libraries
Run Model

Conclusion

Our goal of creating a machine learning classification system based upon our group members' personal music tastes was accomplished to a degree beyond which we thought would be attainable given our relative experience level. Challenges presented to our group included heavily preprocessing and collecting data, experimenting with different machine learning algorithms, and learning to communicate our findings in a clear and concise manner for our peers to follow along with. Our data collection included compiling quite a large amount of data from each of our Spotify libraries. Preprocessing our data proved to be the most challenging aspect given the multiple metrics provided by Spotify's API. Deciding which metrics contained valuable information and dropping those which were deemed unimportant greatly benefitted our selected model's ability to learn and extrapolate meaningful representations of each of our individual music tastes. Furthermore, preprocessing our data specifically for use with a Keras artificial neural network proved quite challenging relative to the process needed for models provided by Scikit Learn's API. However, the accuracy provided by our trained neural network greatly surpassed our other models and proved invaluable in our understanding of different preprocessing methods necessary for various types of machine learning algorithms. With an overall accuracy of 89.6%, our selected model could accurately predict which group member would enjoy a previously unseen song nine times out of ten. Future expansion of our work would include expanding our training data to include a wider array of genres and adding more group members - which would correspond to more potential classifications for the model to learn. These additions would likely require increasing the complexity of our model and require longer training times to reach a similar level of accuracy. Overall, the experience gained from this project greatly informed our group members of the levels of work and understanding necessary to create, maintain, and improve upon machine learning algorithms that will only become more important to the field of computer science in the near future.