CSCI 184: Final Project

Project Title: ViralVibe: Classifying Song Popularity with ML

Group Members: Samantha Lee, Lydia Myla, and Camelia Siadat

Dataset: CSCI 184 - Dataset of Songs

For our project, we have chosen to use a dataset from Kaggle titled *"song_track.csv"*, that includes data on Spotify songs from a variety of genres, artists, popularity levels, and more. The features included in this dataset are genre, artist_name, track_name, track_id, popularity, acousticness, danceability, duration_ms, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time_signature, and valence. Our target variable for our Machine Learning will be 'popularity'. The goal of our project is to classify songs as 'popular' or 'not popular' using the values of the numeric features in this dataset. Our overall prediction works as follows: a 'popularity' value of 0 encodes songs that are 'not popular,' while a label of 1 encodes songs that are 'popular'. Using training and testing sets, we will implement different types of models and observe how well they perform on the data they have been given.

Project Overview

For this project, our group has decided to program and compare multiple decision tree algorithms, including Random Forest, XGBoost, and the general Decision Tree ID3 algorithm. In addition to these decision tree algorithms, we have also decided to implement a Neural Network algorithm using Keras 3's Models, Optimizers, and Metrics APIs.

Our goal is to preprocess the data, drop unnecessary columns, and perform feature (target variable) extraction. We will implement our Random Forest, XGB Classification, Decision Tree Classification, and Neural Network Classification algorithms using appropriate libraries such as Scikit-Learn, XGBoost, and Keras in Python. We will split the dataset into training and testing sets, and then evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1-score. Using libraries such as Matplotlib and Seaborn, we will select appropriate graphs, such as the confusion matrix, to present the data in a meaningful way. We will also

compare the performance of the different algorithms and identify the most effective approach for classifying ‘popular’ or ‘not popular’ songs based on the given dataset.

Previous Attempts

In 2018, two students from Lund University wrote their thesis on “Classifying energy levels in modern music using machine learning techniques”. Otto Nordander and Daniel Pettersson present a solution based on “artificial neural networks and transfer learning that is able to classify ten different energy levels on a linear scale from one to ten independent of genre.”
In their thesis, Nordander and Pettersson share how music is a big part of people’s lives. When listening to music, certain characteristics of music are more applicable in certain situations, depending on mood, setting, or time of day. One interesting characteristic that the students found was “energy level”. Intrigued by the possibilities of this characteristic, the students strove to find a way to categorize music into different energy levels, so that a user could find a specific type of music they wanted to listen to during a specific time. After implementing their models, Nordander and Pettersson compared the results of their neural network architecture with the results of the baseline model and the XGBoost Classifier. Their results indicated that “machine learning algorithms are able to generalize a solution to the problem to some extent”. By comparing the evaluations of each classifier, they found that MAE presented their data in an accurate way, so they chose to use it as their metric when presenting their results.
Similarly, the goal of our project is to classify songs based on their popularity and observe which model is the most efficient in doing so. The more popular a song is, the more likely a person would be inclined to listen to it. By using decision tree algorithms and neural networks, we strive to classify a dataset of songs and report its performance by using the following metrics: accuracy, precision, recall, and F1-score. For visualization, we plan to use a confusion matrix to share the results of each model. By using these tools, we hope to find the best model to achieve our goal of classifying songs based on their popularity.

Methodology: Design, Process, Implementation, and Results

- **Importing Library**

  In this initial step of our data analysis process, we lay the groundwork by importing essential libraries and loading our dataset. We imported critical libraries and modules

tailored to our needs, including NumPy, pandas, Matplotlib, Scikit-learn, and Seaborn. These libraries provide comprehensive support for numerical computation, data manipulation, and a diverse range of plotting options. Subsequently, we loaded our dataset, "*song_track.csv*," utilizing pandas' read_csv() function, which facilitated the creation of a DataFrame named data.

- **Preprocessing**

  Using pandas' DataFrame indexing, we chose our target column, "popularity" and dropped irrelevant feature columns such as 'genre', 'artist_name', 'track_name', 'track_id', 'time_signature', 'mode', and 'key'. This selective extraction relied on eliminating categorical features, since decision tree-based classifiers, and neural networks, perform best when using numeric feature values.

  Using pandas' DataFrame Max, DataFrame Min, and DataFrame Mean functions, we printed out the maximum, minimum, and mean values for the 'popularity' column of our dataset. The minimum and mean values were used to determine the range of the first bin, -1 to 41, since the minimum popularity value was 0 and the mean value was 41.1. Songs with popularities within this range would be classified as 'not popular,' and receive a popularity label of 0. Similarly, the mean and maximum values were used to determine the range of the second bin, 41 to 100, since the maximum popularity value was 100. Songs with popularities within this range would be classified as 'popular' and receive a popularity label of 1.

- **Target Variable**

  For the purposes of this project, we decided to select 'popularity' as our target variable, since this feature was numeric, and could be used for binary classification.

- **Train and Test**

  In this segment, we divided the dataset into separate training and testing sets, a common practice in machine learning. This function takes the predictor variables and the target variable as inputs, along with the optional parameter 'random_state', ensuring consistent and reproducible results across different runs. Upon execution, it generates four arrays: 'X_train', 'X_test', 'y_train', and 'y_test', representing the training and testing subsets for the predictor features and target variable. The training set size was 70%, while the test size was 30%. This splitting strategy allowed for training the model on one subset and

validating its efficacy on unseen data from the other subset, aiding in assessing generalization capabilities and avoiding overfitting. We recognize that adjusting the 'random_state' parameter can influence the partitioning of data, facilitating experimentation for optimal model performance.

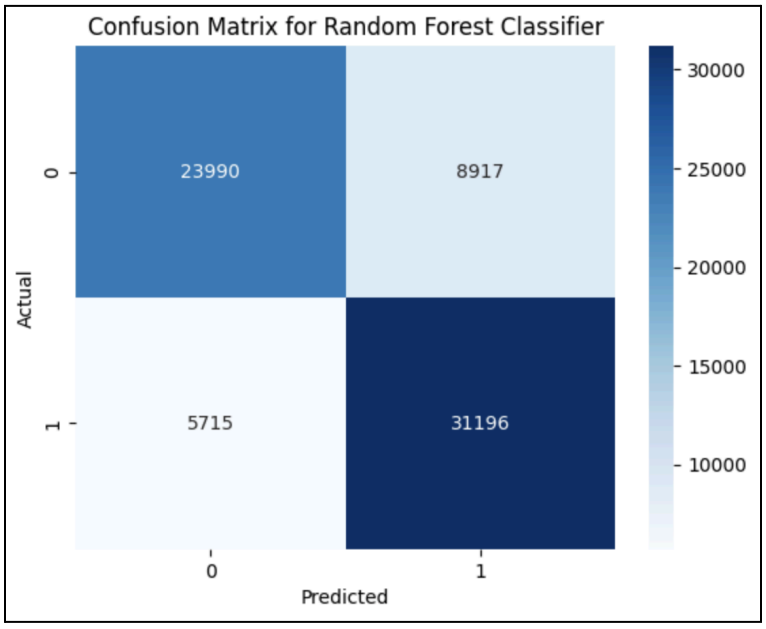
- **Random Forest**

  A random forest algorithm is used for classification and regression tasks. It constructs multiple decision trees during training and outputs the classification results. This algorithm introduces randomness by selecting random subsets and data samples to create each tree. Using Scikit-learn's Random Forest Classifier, we were able to implement this algorithm for our project. First, we trained our Random Forest Classifier on X_train and y_train. We then predicted the results, y_pred, using the test set X_test. To visualize our results, we used a confusion matrix to show the True Positive, True Negative, False Positive, and False Negative results.

  - Results of Random Forest Classifier

```
Random Forest Accuracy: 79.04%
Precision: 0.79
Recall: 0.79
F1 Score: 0.79
Confusion Matrix for Random Forest:
[[23990  8917]
 [ 5715 31196]]
```

  - Confusion Matrix for Random Forest Classifier



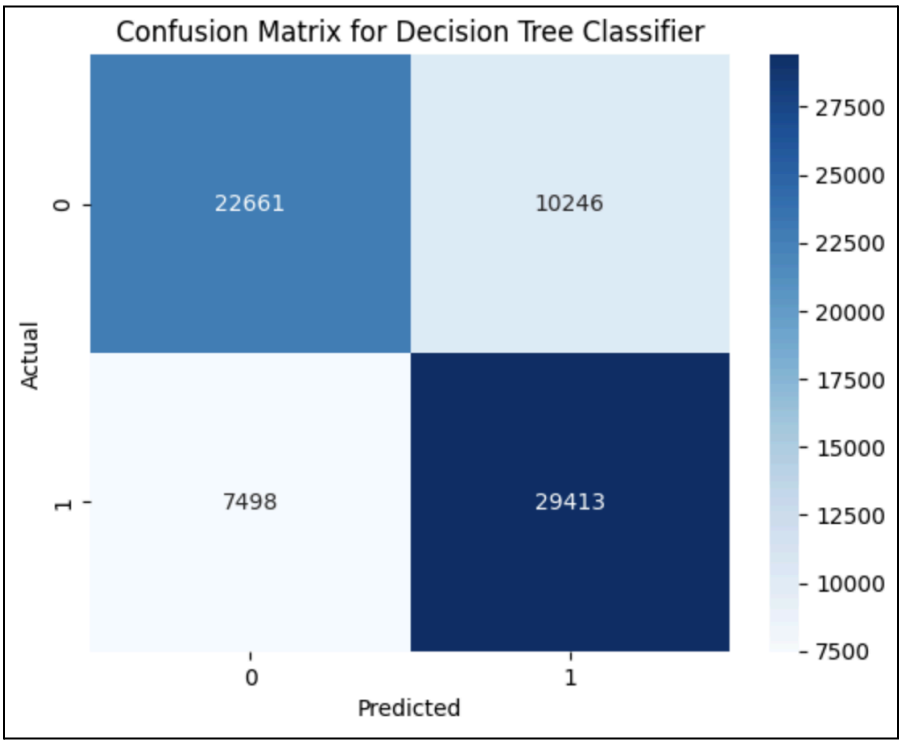
- **Decision Tree**

  A decision tree algorithm is a supervised learning method used for classification. It begins with all examples at the root node. Next, it calculates the information gain for all possible features and picks the feature with the highest information gain. It splits the dataset according to the selected feature and creates left and right branches of the tree. Finally, it repeats the splitting process until the stopping criteria are met. Using Scikit-learn's Decision Tree Classifier, we were able to implement this algorithm given our dataset. First, we trained our Decision Tree Classifier on X_train and y_train. We then predicted the results, y_pred, using the test set X_test. To visualize our results, we used a confusion matrix to show the True Positive, True Negative, False Positive, and False Negative results.

  - Results of Decision Tree Classifier

```
Decision Tree Accuracy: 74.59%
Precision: 0.75
Recall: 0.75
F1 Score: 0.74
Confusion Matrix for Decision Tree:
[[22661 10246]
 [ 7498 29413]]
```

  - Confusion Matrix for Decision Tree



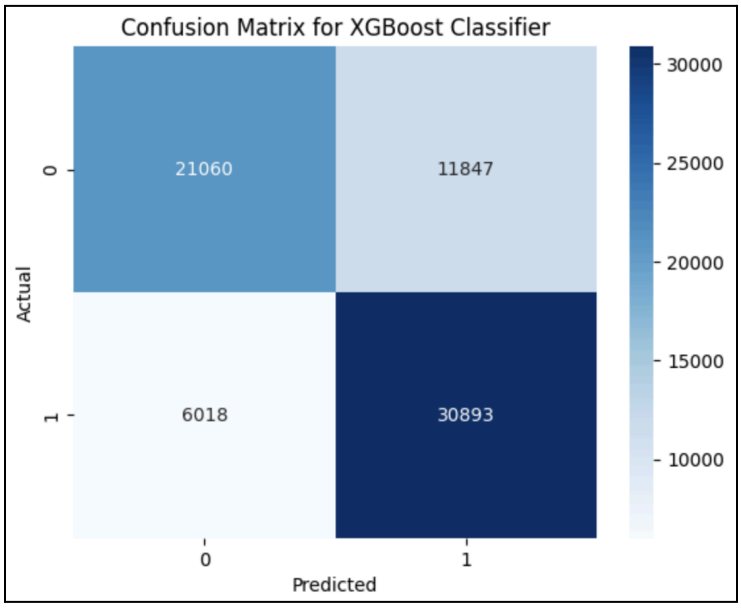
- **XGBoost**

  A boosted decision tree algorithm (such as XGBoost) is used for classification and regression tasks. Similar to random forest, this type of algorithm constructs multiple decision trees during training and outputs the classification results. This algorithm introduces randomness by selecting random subsets to train each tree on. Instead of randomly selecting examples, like in Random Forest classification, a boosted tree algorithm involves picking from all possible examples with the following addition: examples that were misclassified by previously trained trees are chosen with higher probability. Using XGBoost's XGB Classifier, we were able to implement this algorithm for our project. First, we trained our XGB Classifier on X_train and y_train. We then predicted the results, y_pred, using the test set X_test. To visualize our results, we used a confusion matrix to show the True Positive, True Negative, False Positive, and False Negative results.

  - Results of XGBoost

```
XGBoost Accuracy: 74.59%
Precision: 0.75
Recall: 0.75
F1 Score: 0.74
Confusion Matrix for XGBoost:
[[21060 11847]
 [ 6018 30893]]
```

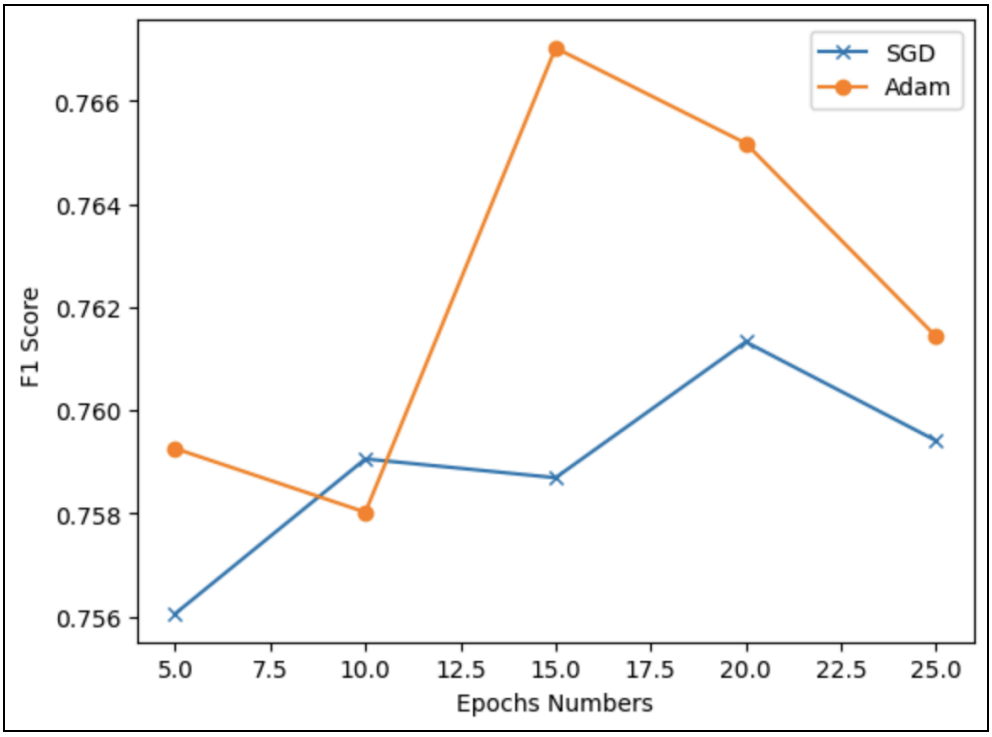
  - Confusion Matrix for XGBoost



- **Neural Networks**

  A Neural Network algorithm is used for both regression and classification. Since a neural network is composed of an input layer, a number of hidden layers, and an output layer, a neural network algorithm first begins by feeding input values into the perceptrons in the first hidden layer. An activation function is called on the linear combination of the weights and inputs, plus biases, to each perceptron, and each of the values from this activation sequence is passed to the perceptrons in the next layer. This process continues until the final hidden layer produces the final value to be passed into an activation function. This ultimate activation function then produces an output value, y-hat. Using Keras 3's Sequential model, we were able to implement this algorithm for our project. Our first step was to create a neural network that would use stochastic gradient descent (SGD) optimization. This form of optimization is a type of gradient descent algorithm using momentum. In contrast, the other neural network that we created relied on Adam optimization, another gradient descent optimizer. For both neural networks, we first trained each network on X_train and y_train. We then predicted the results, y_pred, using the test set Xtst. Originally, we had planned to visualize our results by recording and comparing the accuracy of each neural network at 5, 10, 15, 20, 25, 30, 35, 40, 45, and 50 epochs. However, doing so proved to be extremely computationally expensive, and as a result, we decided instead to record and compare the accuracy of each neural network at 5, 10, 15, 20, and 25 epochs. This comparison was performed by plotting the F1 scores of the SGD-optimized neural network versus the Adam-optimized neural network.

  - F1 Scores of the SGD Optimizer at 5, 10, 15, 20, and 25 epochs
    - [0.7560567569242209, 0.7590566436084996, 0.7586921511591616, 0.7613250527823391, 0.7594108836744883]
  - F1 Scores of the Adam Optimizer at 5, 10, 15, 20, and 25 epochs
    - [0.7592583337498126, 0.7580234335201222, 0.7670210553360167, 0.7651737746032136, 0.7614295313705483]
  - F1 Score Comparison between SGD Optimized Neural Network and Adam Optimized Neural Network



Results and Conclusion

We assessed the accuracy, precision, recall, and F1 score metrics of 4 machine learning models: Random Forest, Decision Tree, XGBoost, and Neural Network. In our evaluations, we decided to use F1 Score since A), it is a good metric for assessing model performance when there is class imbalance, and B), it provides a balance between precision and recall. Although the F1 scores among all models were not vastly different, our Random Forest model had the highest F1 Score of 79%. It demonstrated an accuracy of 79.04%, precision of 79%, and recall of 79%. The other three models showed promising results, but they either did not have as high of an F1 score or were not efficient computationally. Our group originally considered the Neural Network classifiers as the best choice, due to their ability to represent more complex patterns. The Neural Network classifiers produced F1 scores close to that of the Random Forest classifier, however, the network classifiers proved to be very computationally expensive and took a much greater amount of time to execute compared to the other models. They required more resources to achieve optimal results. Although the neural networks posed potential, the computational complexity and expenses made them less practical and ideal for our purposes. Based on our results, we have decided to choose Random Forest as our optimal model for classifying ‘popular’ and ‘not popular’ songs.

Code File: CSCI 184 Final Project Code

Additional Resources and Links

- Kaggle
    - We retrieved the dataset from Kaggle in a .csv file.
        - Song Track Dataset
- Google Sheets
    - We imported the data (.csv) into a Google sheet to organize, store it, and then downloaded a full .csv file to use in our code.
        - Dataset in Google Sheets
- Jupyter Notebooks
    - We implemented our project code on a shared notebook using Google Colab.
        - Notebook: CSCI 184 Final Project Notebook