**CLASSIFYING SONGS INTO PLAYLIST USING KNN**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the course project of

**CSE338L Applied Data Science Lab**

Submitted by

**ABHIRAM THIRIVEEDHI** (AP20110010457)

**BHARGAV KUMBHAM (**AP20110010467**)**

**MOHAN KRISHNA (**AP20110010433**)**

**A picture containing text

Description automatically generated**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[May,2023]**

# Table of Contents

[Table of Contents 1](#_Toc135689891)

[1. Abstract 2](#_Toc135689892)

[2.Introduction 2](#_Toc135689893)

[3.Survey 3](#_Toc135689894)

[4.Methodology 4](#_Toc135689895)

[4.1 Dataset Collection 4](#_Toc135689896)

[4.2 Feature Extraction 4](#_Toc135689897)

[4.3 Data Preprocessing 5](#_Toc135689898)

[4.4 Normalizing the data 6](#_Toc135689899)

[4.5 Splitting the Dataset into training and testing data 6](#_Toc135689900)

[4.6 Training the model 7](#_Toc135689901)

[4.7 Performance Evaluation 7](#_Toc135689902)

[5 . Results / Screenshots 8](#_Toc135689903)

[5.1 Graphs of attributes varying with songs 8](#_Toc135689904)

[5.2 Accuracy results 9](#_Toc135689905)

[6.Conclusion and Future Work 9](#_Toc135689906)

# 1. Abstract

Users now find it difficult to effectively organize and sort through large music collections as digital music is becoming more and more widely available. To solve this problem, music sorting and recommendation tasks have been subjected to the application of machine learning algorithms, such as the K-Nearest Neighbors (KNN) algorithm. An introduction to music sorting using the KNN algorithm is given in this abstract. KNN classifies or predicts the class of new instances based on their proximity to existing labeled instances. It is a non-parametric, instance-based learning technique. The KNN algorithm takes into account a variety of song characteristics, including tempo, genre, and key, to determine how similar songs are.

The procedure includes gathering data, feature extraction, training the KNN model, selecting the value of K and the distance metric, classifying new songs, sorting songs according to their assigned classes, and making song recommendations. The KNN algorithm for music sorting provides an automated and individualized method for effectively organizing music collections and making song recommendations based on users' preferences and interests.

# 2.Introduction

The Music Sorter project is an application that utilizes the K-Nearest Neighbors (KNN) algorithm to classify and organize music tracks based on their audio features. The goal of the project is to provide an efficient and automated system for sorting a large music collection into distinct genres or categories.

The KNN algorithm is a supervised machine learning technique used for classification tasks. In this project, the algorithm is trained on a labeled dataset of music tracks, where each track is represented by a set of audio features such as tempo, pitch, rhythm, and timbre. These features are extracted using signal processing techniques.

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of K number of neighbors

Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

When a new music track needs to be sorted, the KNN algorithm compares its audio features to those in the database and identifies the k nearest neighbors based on similarity measures. The majority genre/category among the k nearest neighbors is assigned to the new track.

To evaluate the performance of the Music Sorter, a comprehensive dataset of music tracks with diverse genres is used for training and testing. The accuracy of the classification is measured using metrics such as precision, recall, and F1-score.

The Music Sorter project can be extended to incorporate additional features or integrate with existing music platforms to enhance music recommendation systems and provide personalized music experiences to users.

# 3.Survey

1. "Automatic Playlist Generation: Learning from Listened Data" by A. Flexer, B. Grünewald, and G. Widmer (2008)

This paper presents an ML-based approach for playlist generation. It proposes a model that learns from user behavior and preferences to create personalized playlists. The authors utilize collaborative filtering and use matrix factorization techniques to recommend songs based on user listening history.

1. "Playlist Generation Using Mood Analysis" by X. Hu, J. Downie, and R. Bay (2007)

This study focuses on playlist generation based on mood analysis. The authors employ audio feature analysis to extract mood-related characteristics from songs. They apply ML algorithms, including decision trees and support vector machines, to classify songs into different mood categories and generate playlists based on user mood preferences.

These studies demonstrate the application of ML techniques in song playlist generation. They explore various approaches, including collaborative filtering, audio feature analysis, metric learning, and deep learning, to create personalized and mood-based playlists. The field continues to evolve, with newer research incorporating advanced ML algorithms and larger datasets to enhance playlist generation systems.

# 4.Methodology

## 4.1 Dataset Collection

Gather a diverse dataset of music tracks with labeled genres or categories. Ensure that the dataset is representative of various music genres and contains a sufficient number of tracks for each genre.

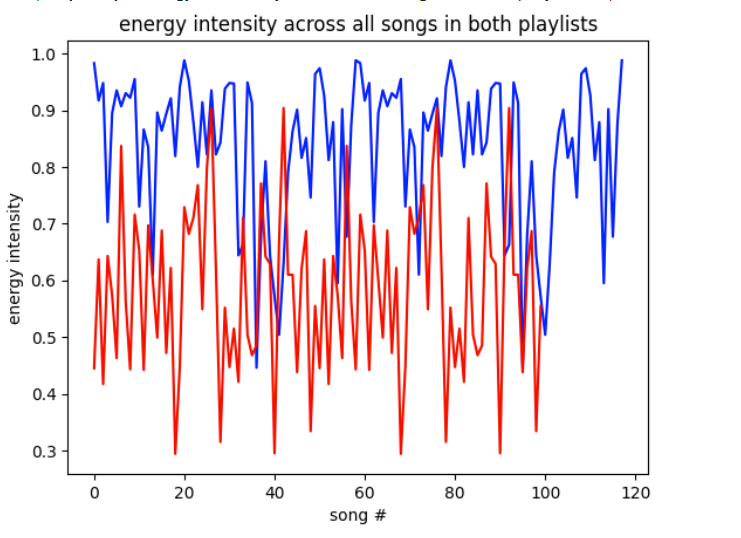
In this project we used the playlists pre made by **Spotify**

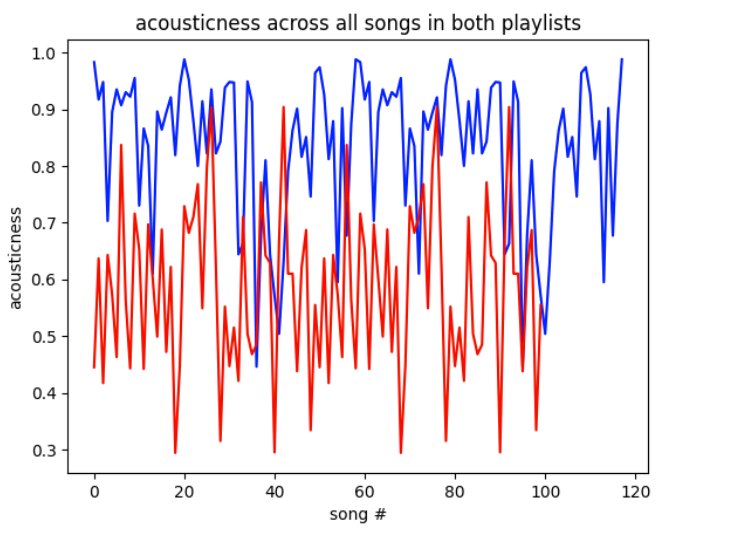
**Genres considered: Mass and Romantic**

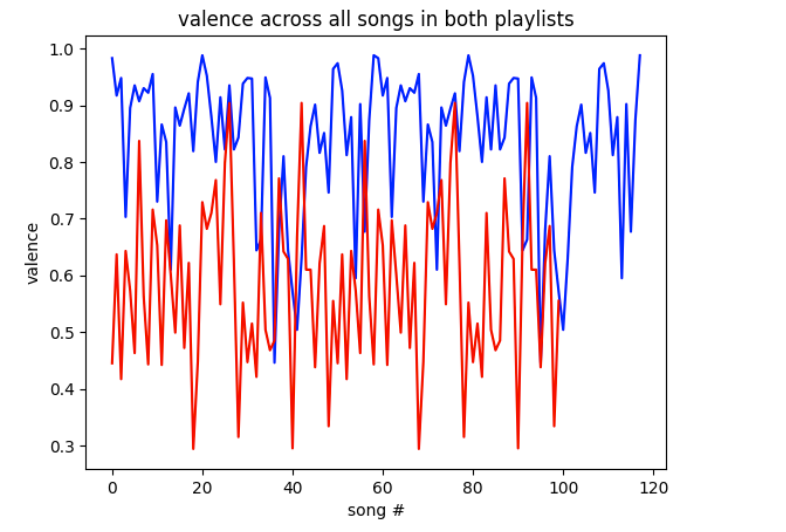
## 4.2 Feature Extraction

Extract relevant audio features from each music track in the dataset. Commonly used features include energy, accousticness, loudness, liveness, valence etc

After visualizing the dataset through graphs, it is known that we can use the attributes energy, valency and accousticness to classify the songs







## 4.3 Data Preprocessing

These three distinct attributes were chosen from the dataset.

Both csv files were imported into separate tabs of the sheet for each playlist. Then, all other attributes were removed (such as song names, artists, danceability, etc.) and three attributes from each dataset were combined (energy, acousticness, and valence).

Each data point now has a "Target" column that indicates which playlist it belongs to. 0 is a romantic playlist; 1 is a general playlist. The attributes of energy, acousticness, and valence are the inputs to the model, and the target column is the output (playlist).

## 4.4 Normalizing the data

Setting all numerical values in each column to the same scale, or normalizing, prevents the difference between values from appearing distorted. Fundamentally, we're rescaling the data to have values between 0 and 1 and creating z-scores.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(mass\_romantic.drop('target', axis=1))

An object was created of the standard scaler, which will normalize the dataset, by importing StandardScaler from sklearn as mentioned above. The "target" column was then deleted because that is what would be output after we had fitted the object to all the data.

## 4.5 Splitting the Dataset into training and testing data

Divide the dataset into training and testing sets. The training set will be used to train the KNN algorithm, while the testing set will be used to evaluate its performance.

The KNN algorithm needs to take in 2 sets: the training and testing set. These sets come from the dataset itself where a share of the dataset gets trained (into a model) and the rest gets tested for the model’s accuracy.

from sklearn.model\_selection import train\_test\_split

X = mass\_romantic\_feat

y = mass\_romantic['target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=30, shuffle=True)

To actually divide the dataset, variables must be created that represent the input (X) and output (y) and import train\_test\_split from sklearn.

The playlist ("target" column) will be the output (y) and the input (X) will be the music attributes/features (in this case: energy, acousticness, and valence), as we want to know what playlist is the best fit for specific songs.

The random state, which is the initial value chosen by the random generator, has no bearing on how the algorithm behaves and can take on any value. To make sure the data I split is randomly distributed and the most accurate representation of the entire dataset, shuffle is used.

The test size is the ratio of the dataset that was putting towards the testing set which was 0.3, meaning 0.7 (70%) of the dataset is going to be trained.

## 4.6 Training the model

Now that officially 0.7 of the dataset was split towards testing, it’s time to finally use KNeighborsClassifier from sklearn to train the model.

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7,p=2,metric='euclidean')

knn.fit(X\_train, y\_train)

## 4.7 Performance Evaluation

Assess the performance of the Music Sorter by comparing the predicted genres/categories with the actual genres/categories of the testing set. Calculate evaluation metrics such as precision, recall, and F1-score to measure the accuracy of the classification.

Now that the model has been trained, it is time to test it and use the Classification Report to determine its accuracy.

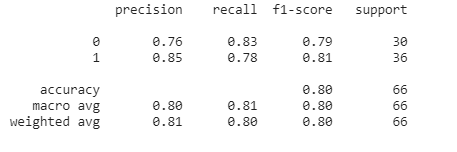
knn = KNeighborsClassifier(n\_neighbors=27)

knn.fit(X\_train, y\_train)

prediction= knn.predict(X\_test)

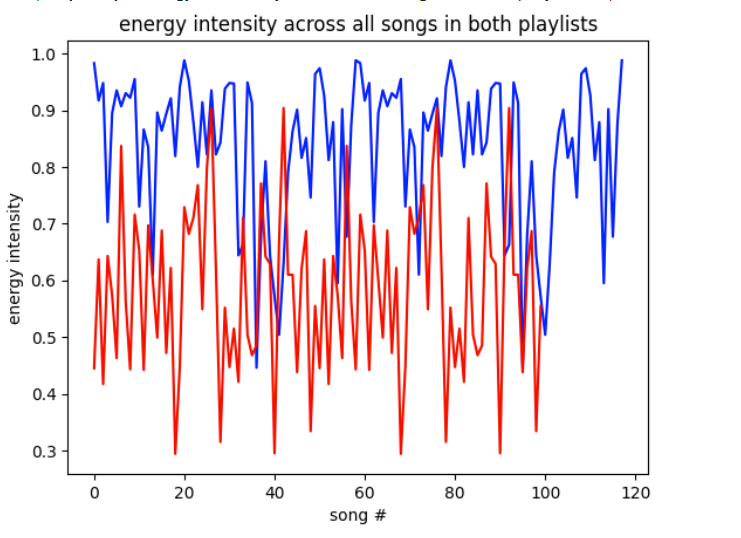
prediction

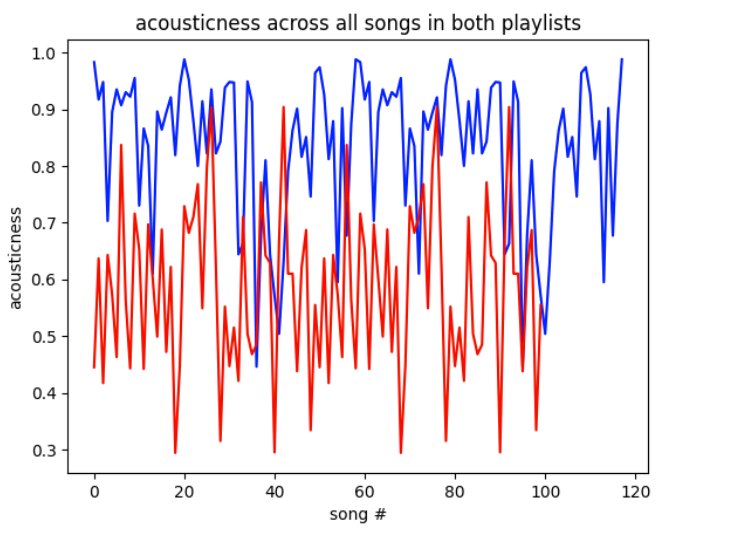
All of the inputs (music attributes) from the testing set are contained in the prediction object, and the trained model produces a prediction of the playlists each song should be included in. Precision, Recall, and F1 values are used by comparing the output values from the training set with those from the testing set, to determine how accurate the model is.

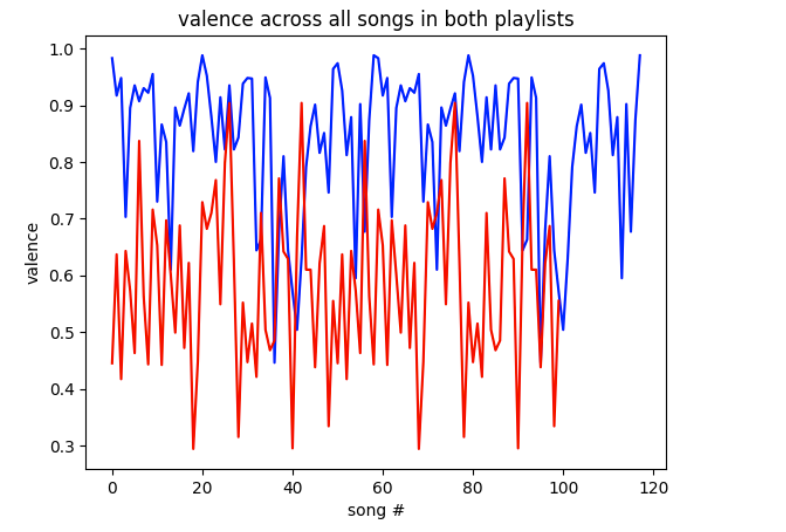


# 5 . Results / Screenshots

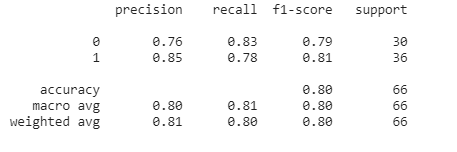
## 5.1 Graphs of attributes varying with songs







## 5.2 Accuracy results



# 6.Conclusion and Future Work

This project's goal was to build a dependable system that can take a song and categorize it into playlists according to genre. Any Spotify playlist can use this. Even though it would be nice if the precision and accuracy were a little higher, this model can always be improved by being trained with better songs that fit the genre. The difficulty this model may have encountered is telling some songs apart in the two playlists that may share the same genre.

Explore opportunities for extending the Music Sorter project. This can include integrating with existing music platforms, incorporating additional features like lyrics analysis or mood detection, or enhancing the recommendation system to provide personalized music suggestions based on user preferences.