CIS 550 Advance Machine Learning
Fall'23
Project Summary

Identifying the Genre of a Song using Machine Learning Algorithms

Group-10
Sai Alekhya Ravi (59)
Sravani Kairam Konda (26)
Uday Reddy Polepally (54)

Dr. Ashok Patel

Department of Computer Science
University of Massachusetts Dartmouth, MA

1. Introduction

1.1 Background

The categorization of music genres based on audio data is an important activity that plays an important role in the music industry. In the realm of music information retrieval, the classification of musical styles into genres is used rather frequently. The suggested structure addresses three primary steps: the preparation of data, the extraction of features, and the classification of data. Convolutional neural network, also known as CNN, is the technique that is used to classify different types of music. The proposed approach feeds the feature values of spectrograms that are created from slices of songs into a CNN so that the songs can be categorized according to the music genres to which they belong. Following the completion of the categorization process comes the introduction of a recommendation system. The goal of the recommendation system is to tailor song suggestions to the tastes and passions of individual users. Extensive testing performed on the GTZAN dataset demonstrates that the suggested system is superior to previous approaches in terms of its level of efficacy.

1.2 Existing System

The fields of recommendation systems, instrument recognition, and track separation have all recently taken an interest in how to classify music based on its genre. Recently, the classification of music genres has seen a significant development. The classification of music genres is an important stage in the development of a recommendation system. Many real-world applications, such as efficient search, retrieval, and recommendation, rely on music genre classification due to the large amount of music archives. Music genre classification and music recommendation are the major goals of this article, which uses data from music time slices.

1.3 Proposed System

Data preparation, feature extraction, classification, and recommendation system all take place in these four stages. Librosa python Library is used for feature extraction and data pre-processing. Data preprocessing and feature extraction are both handled by librosa. It is utilized in classification problems with the help of a neural network. CNN is used to determine music genres from audio data in a technique known as music genre classification. An unknown audio clip was also utilized to classify into a musical genre using a majority voting mechanism, and spectrogram feature values from time slices of songs were employed. Cosine similarity can then be used to construct a recommendation system for songs, based on the song's attributes.

1.4 Software Requirements

Product viewpoint and features, OS and operating environment, graphics requirements, design limitations and user documentation are all included in the functional requirements or overall description documents. The project's strengths and weaknesses, as well as the best ways to address them, may be seen in the appropriation of requirements and implementation restrictions.

Python IDEL 3.7 (**or**) Anaconda 3.7 (**or**) Jupiter (**or**) Google Colab

2. Dataset

2.1 Problem statement

The rapid increase in the volume of music releases on platforms like Spotify and Soundcloud poses a challenge for effective database management and search functionality. With tens of thousands of songs being added monthly, accurate genre classification is crucial for organizing and analyzing this vast musical content.

Our goal is to develop and implement robust music genre classification algorithms using various approaches, like k-nearest neighbors, a basic feed-forward network, and an advanced convolutional neural network. By experimenting with both raw amplitude data and mel-spectrograms, we aim to identify the most effective method for predicting genres out of a set of 10 common music genres. Our ultimate achievement is to surpass human accuracy, providing valuable insights for music streaming and purchasing services

2.2 Data Collection

We found our GTZAN dataset from Kaggle. The most popular publicly available dataset for assessment in machine listening studies pertaining to music genre identification (MGR) is the GTZAN dataset. To reflect a range of recording settings, the files were gathered in 2000–2001 from a number of sources, such as radio, personal CDs, and microphone recordings.

2.3 Data Contents

genres original - A collection of 10 genres with 100 audio files each, all having a length of 30 seconds (the famous GTZAN dataset, the MNIST of sounds)

images original - A visual representation for each audio file. One way to classify data is through neural networks. Because NNs (like CNN, what we will be using today) usually take in some sort of image representation, the audio files were converted to Mel Spectrograms to make this possible.

2 CSV files - Containing features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs were split before into 3 seconds audio files (this way increasing 10 times the amount of data we fuel into our classification models).

10 Classes - Classical, Jazz, Country, Rock, Blues, Reggae, Metal, Disco, Hiphop, Pop

Approximately 1000 samples for each class - 10.000 Total Samples

58 Features (Numerical) - Chroma stft, rms, spectral_Centroid, spectral_ Bandwidth, Rollof, Zero_crossing, Harmony, perceptr, tempo, MFCC(1-20), length

About this file

features 30 seconds: a csv with a mean and average value of different extracted features from the audio files

filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rn
1000 unique values	660k 676k	0.17 0.66	0.04 0.11	0.01 0.4	0
blues.00000.wav	661794	0.35008811950683594	0.08875656872987747	0.1302279233932495	0.00 3
blues.00001.wav	661794	0.3409135937690735	0.09498025476932526	0.09594780951738358	0.00
blues.00002.wav	661794	0.36363717913627625	0.08527519553899765	0.17557041347026825	0.00 6
blues.00003.wav	661794	0.4047847092151642	0.09399903565645218	0.14109300076961517	0.00
blues.00004.wav	661794	0.30852603912353516	0.0878409817814827	0.09152871370315552	0.00
blues.00005.wav	661794	0.3024562895298004	0.08753237873315811	0.10349363833665848	0.00 6
blues.00006.wav	661794	0.29132798314094543	0.09398132562637329	0.14187414944171906	0.00

Fig 1: sample of csv file features_30_seconds

About this directory

genres original: 10 genres with 100 audio files each, 30 seconds long in wav format

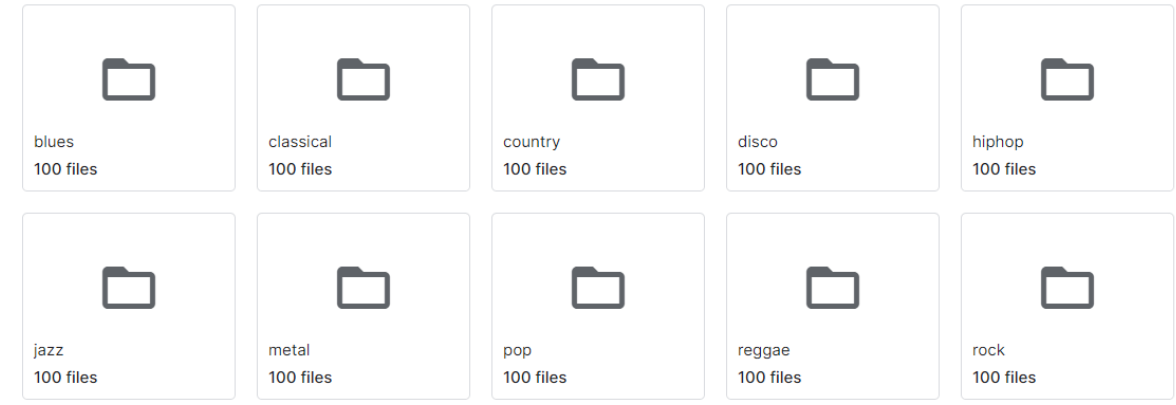


Fig 2: Folder with different genres each genre with 100 files

2.4 Exploring the data

We began by downloading the GTZAN dataset from Kaggle and then saving the downloaded files to a folder on our hard drive. This data was then uploaded to Colab. A pop-up window appears, requesting permission for this Colab notebook to access the Google Drive files. Once the mounting is complete, we can begin analyzing the data by specifying a dataset path to access the dataset's folder.

Importing the Data

Mounting the drive

```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

Setting the dataset path

```
[ ] dataset_path = '/content/drive/MyDrive/Project/Data.zip (Unzipped Files)/'
```

Fig 3: Importing the data

Exploring the Data

```
pd.set_option('display.max_columns', 60)
df = pd.read_csv(f'{dataset_path}/features_30_sec.csv')
df.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	rolloff
0	blues.00000.wav	661794	0.350088	0.088757	0.130228	0.002827	1784.165850	129774.064525	2002.449060	85882.761315	3805.8
1	blues.00001.wav	661794	0.340914	0.094980	0.095948	0.002373	1530.176679	375850.073649	2039.036516	213843.755497	3550.5
2	blues.00002.wav	661794	0.363637	0.085275	0.175570	0.002746	1552.811865	156467.643368	1747.702312	76254.192257	3042.2
3	blues.00003.wav	661794	0.404785	0.093999	0.141093	0.006346	1070.106615	184355.942417	1596.412872	166441.494769	2184.7
4	blues.00004.wav	661794	0.308526	0.087841	0.091529	0.002303	1835.004266	343399.939274	1748.172116	88445.209036	3579.7

```
[ ] df.shape
```

```
    (1000, 60)
```

Fig 4: Exploring the data

The df.info() method is frequently used in pandas, a prominent Python data manipulation tool. This method returns a brief summary of a DataFrame, including details like the number of non-null elements, data types, and memory utilization.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 60 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   filename                 1000 non-null   object
 1   length                   1000 non-null   int64
 2   chroma_stft_mean         1000 non-null   float64
 3   chroma_stft_var          1000 non-null   float64
 4   rms_mean                 1000 non-null   float64
 5   rms_var                  1000 non-null   float64
 6   spectral_centroid_mean   1000 non-null   float64
 7   spectral_centroid_var    1000 non-null   float64
 8   spectral_bandwidth_mean  1000 non-null   float64
 9   spectral_bandwidth_var   1000 non-null   float64
 10  rolloff_mean             1000 non-null   float64
 11  rolloff_var              1000 non-null   float64
 12  zero_crossing_rate_mean  1000 non-null   float64
 13  zero_crossing_rate_var   1000 non-null   float64
 14  harmony_mean             1000 non-null   float64
 15  harmony_var              1000 non-null   float64
 16  perceptr_mean            1000 non-null   float64
 17  perceptr_var             1000 non-null   float64
 18  tempo                    1000 non-null   float64
 19  mfcc1_mean               1000 non-null   float64
 20  mfcc1_var                1000 non-null   float64
 21  mfcc2_mean               1000 non-null   float64
 22  mfcc2_var                1000 non-null   float64
 23  mfcc3_mean               1000 non-null   float64
```

```
df.columns
```

```
Index(['filename', 'length', 'chroma_stft_mean', 'chroma_stft_var', 'rms_mean',
       'rms_var', 'spectral_centroid_mean', 'spectral_centroid_var',
       'spectral_bandwidth_mean', 'spectral_bandwidth_var', 'rolloff_mean',
       'rolloff_var', 'zero_crossing_rate_mean', 'zero_crossing_rate_var',
       'harmony_mean', 'harmony_var', 'perceptr_mean', 'perceptr_var', 'tempo',
       'mfcc1_mean', 'mfcc1_var', 'mfcc2_mean', 'mfcc2_var', 'mfcc3_mean',
       'mfcc3_var', 'mfcc4_mean', 'mfcc4_var', 'mfcc5_mean', 'mfcc5_var',
       'mfcc6_mean', 'mfcc6_var', 'mfcc7_mean', 'mfcc7_var', 'mfcc8_mean',
       'mfcc8_var', 'mfcc9_mean', 'mfcc9_var', 'mfcc10_mean', 'mfcc10_var',
       'mfcc11_mean', 'mfcc11_var', 'mfcc12_mean', 'mfcc12_var', 'mfcc13_mean',
       'mfcc13_var', 'mfcc14_mean', 'mfcc14_var', 'mfcc15_mean', 'mfcc15_var',
       'mfcc16_mean', 'mfcc16_var', 'mfcc17_mean', 'mfcc17_var', 'mfcc18_mean',
       'mfcc18_var', 'mfcc19_mean', 'mfcc19_var', 'mfcc20_mean', 'mfcc20_var',
       'label'],
      dtype='object')
```

```
[ ] df.label.unique()
```

```
array(['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz',
       'metal', 'pop', 'reggae', 'rock'], dtype=object)
```

The df.columns feature in pandas is used to get the column labels of a DataFrame. It returns an Index object with the column labels as its data.

df.label.unique() returns the unique values in the DataFrame df's 'label' column. These distinct values are often useful when attempting to comprehend the many categories or labels included in a category column.

```
label_tempo_df= df[['label', 'tempo']]

f, ax = plt.subplots(figsize = (16,9))
sns.boxplot(x = 'label', y = 'tempo', data = label_tempo_df, palette = 'rocket' )

plt.title('BPM Boxplot for Genres', fontsize = 15)
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)
plt.xlabel('Genre', fontsize = 20)
plt.ylabel('BPM', fontsize = 20)
plt.show()
```

df[['label', 'tempo']] = label_tempo_df:

This line generates a new DataFrame label_tempo_df by removing the columns 'label' and 'tempo' from the original DataFrame df. It effectively builds a DataFrame with two columns of interest for the boxplot.

f, ax = plt.subplots(figsize=(16, 9)):

This line generates a Matplotlib figure (f) and axis (ax) with the requested width and height of 16 and 9 units, respectively. The plt.subplots() function is used to create a subplot, and the resultant fig, ax objects are Figure and Axes objects that can be customized further.

sns.boxplot(x="label," y="tempo," data="label_tempo_df," palette="rocket"):

This line makes a boxplot using Seaborn (sns).

The x-axis is allocated the 'label' column (x='label'), which represents the various genres.

The y-axis is allocated the 'tempo' column (y='tempo'), which represents the BPM values.

The data option indicates the DataFrame (label_tempo_df) to be used.

The palette='rocket' argument specifies the plot's color palette.

plt.title('Genres BPM Boxplot,' fontsize=15):

This line changes the plot's title to 'BPM Boxplot for Genres' with a font size of 15.

plt.xticks(fontsize=15) and plt.yticks(fontsize=15):

These lines define the font size for the x- and y-axis tick labels, respectively.

plt.ylabel('BPM', fontsize=20) and plt.xlabel('Genre', fontsize=20):

These lines define the font size of 20 for the x- and y-axis labels.

plt.show():

This line shows the plot.

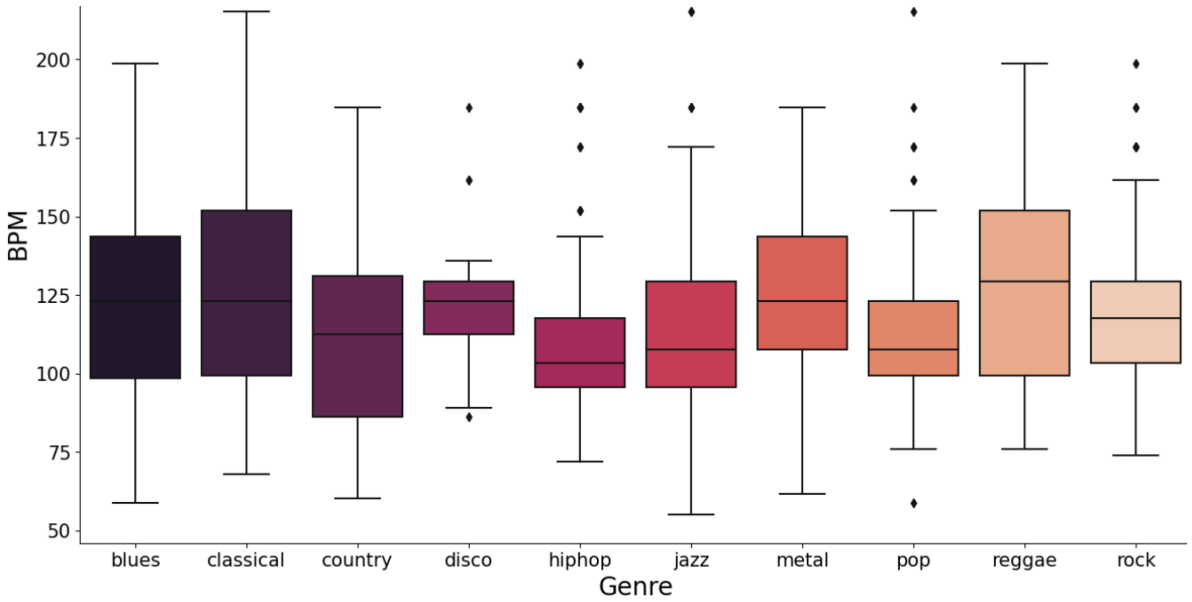


Fig 5: The Box plot of Genre vs Tempo

In the DataFrame's 'label' column, the code generates a boxplot to display the distribution of BPM values for distinct genres. The boxplot reveals the central tendency and distribution of tempo values for each genre, allowing for a fast comparison of tempo features across different music genres. The 'rocket' color palette is employed to visually enhance the plot.

We have chosen box plot to visualize the data. Box plots provide a concise description of a dataset's distribution, including the median, quartiles, and any outliers. This summary can provide a rapid and clear assessment of the core trend and spread of the data. Box plots make identifying probable outliers in a dataset simple. Outliers are displayed as isolated points beyond the "whiskers" of the box plot, highlighting values that may range dramatically from the majority. Box plots provide a simple visual comparison for comparing distributions across several groups or categories (different genres). Differences in medians, spreads, and the existence of outliers are immediately discernible.

3. Pre-processing

3.1 Why?

In the context of data analysis and machine learning, preprocessing refers to the actions and techniques that are done to raw data before it is used for modeling or analysis. The purpose of preprocessing is to convert raw data into a format suitable for the tasks at hand, as well as to increase model performance and interpretability.

Preprocessing is an important step in the data analysis and machine learning pipeline since the quality of the input data influences model performance and the veracity of insights produced from the data. The specific preprocessing steps used are determined by the qualities of the data as well as the objectives of the analysis or modeling assignment.

3.2Pre-processing steps

Pre-Processing

```
from sklearn import preprocessing

data = df.iloc[0:, 1:]
y = data['label']
X = data.drop('label', axis = 1)

cols = X.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)
X = pd.DataFrame(np_scaled, columns = cols)

from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, y], axis = 1)
print('variance ratio : ', pca.explained_variance_ratio_)
print('sum : ', sum(pca.explained_variance_ratio_))
```

```
variance ratio :  [0.2439355  0.21781804]
sum :  0.4617535410562956
```

The code depicts a common preparation workflow that includes scaling features to a common range and using PCA to reduce dimensionality. The print statements show the variance preserved by the specified major components. In this situation, keeping two components (n_components=2) is a standard technique for presenting high-dimensional data.

Data Selection:
data = df.iloc[0:, 1:]: This selects a subset of the original DataFrame df, excluding the first column (assuming it contains the index) and considering all rows.

Target and Features:
y = data['label']: Extracts the target variable 'label'.
X = data.drop('label', axis=1): Creates a DataFrame X containing the features by dropping the 'label' column.

Min-Max Scaling:
min_max_scaler = preprocessing.MinMaxScaler(): Initializes a Min-Max scaler.
np_scaled = min_max_scaler.fit_transform(X): Applies Min-Max scaling to the features.
X = pd.DataFrame(np_scaled, columns=cols): Replaces the original features with the scaled features in a new DataFrame.

PCA (Principal Component Analysis):
pca = PCA(n_components=2): Initializes PCA with the number of components set to 2.
principalComponents = pca.fit_transform(X): Applies PCA to reduce the dimensionality of the features.
principalDf = pd.DataFrame(data=principalComponents, columns=['principal component 1', 'principal component 2']): Creates a DataFrame with the reduced features.

Concatenation:
finalDf = pd.concat([principalDf, y], axis=1): Combines the reduced features with the target variable 'label'.

Print Explained Variance:
print('variance ratio:', pca.explained_variance_ratio_): Prints the explained variance ratio for each principal component.
print('sum:', sum(pca.explained_variance_ratio_)): Prints the sum of explained variance ratios.

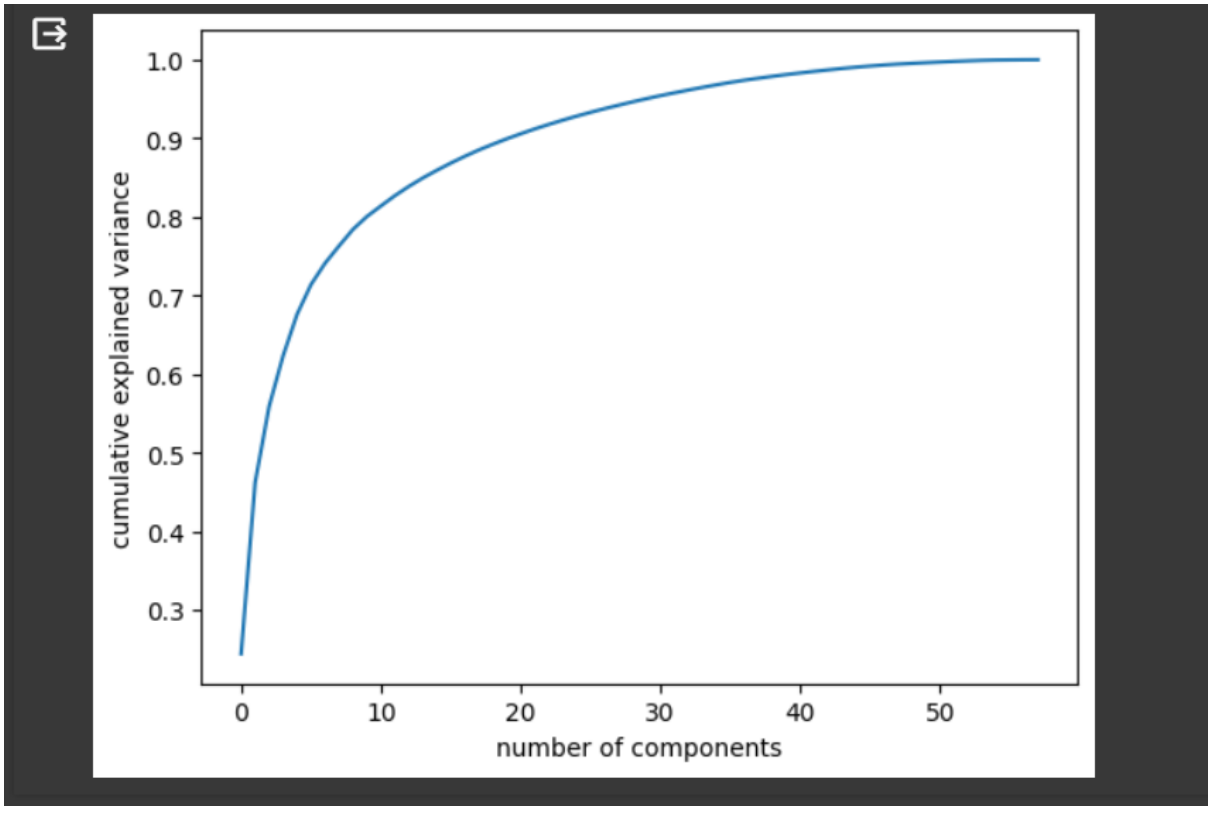


Fig 6: Plots the cumulative explained variance as a function of the number of components.

4. Model building

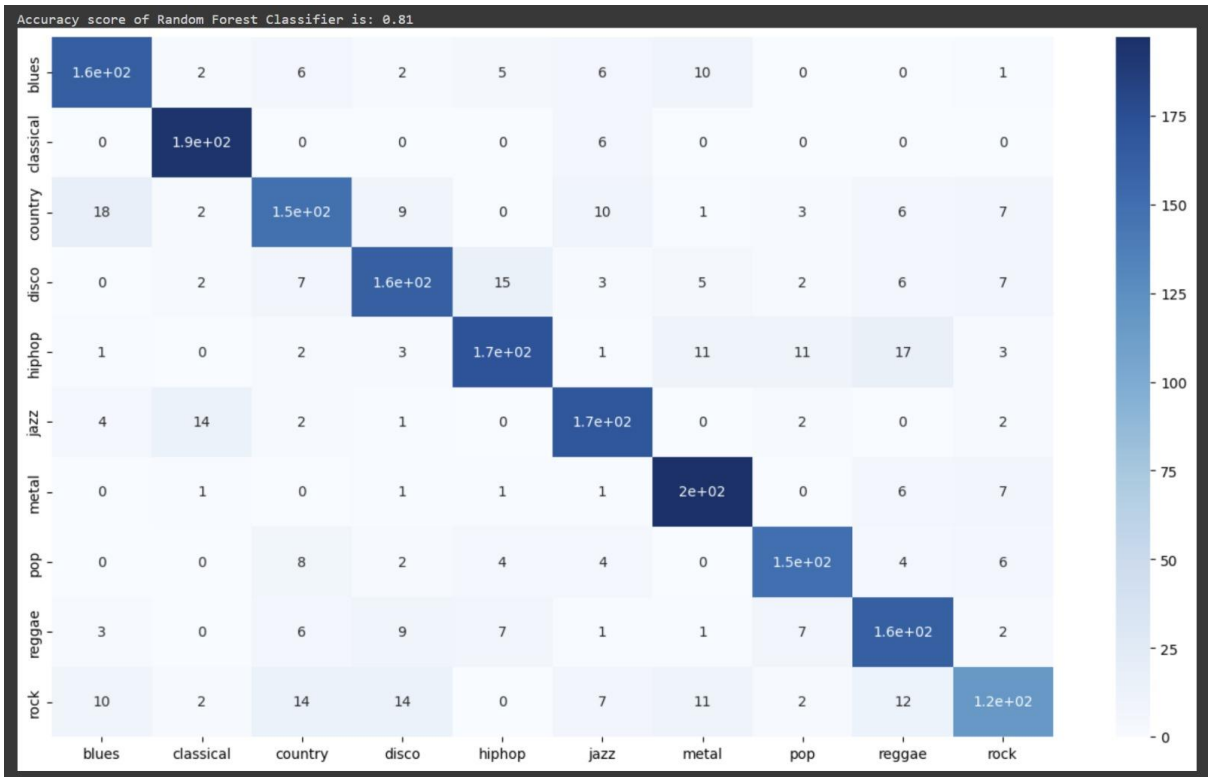
4.1 Machine Learning Classification Models

Machine learning classification models are algorithms that learn from input data to assign predefined labels or categories based on their attributes. These models are frequently employed in tasks that require predicting the category class or label of fresh, unknown data.

Few Machine learning classification models we used for our problem statement:

Random forest

An ensemble learning method that integrates the predictions of many decision trees. It is well-known for its robustness and capacity to handle large datasets.



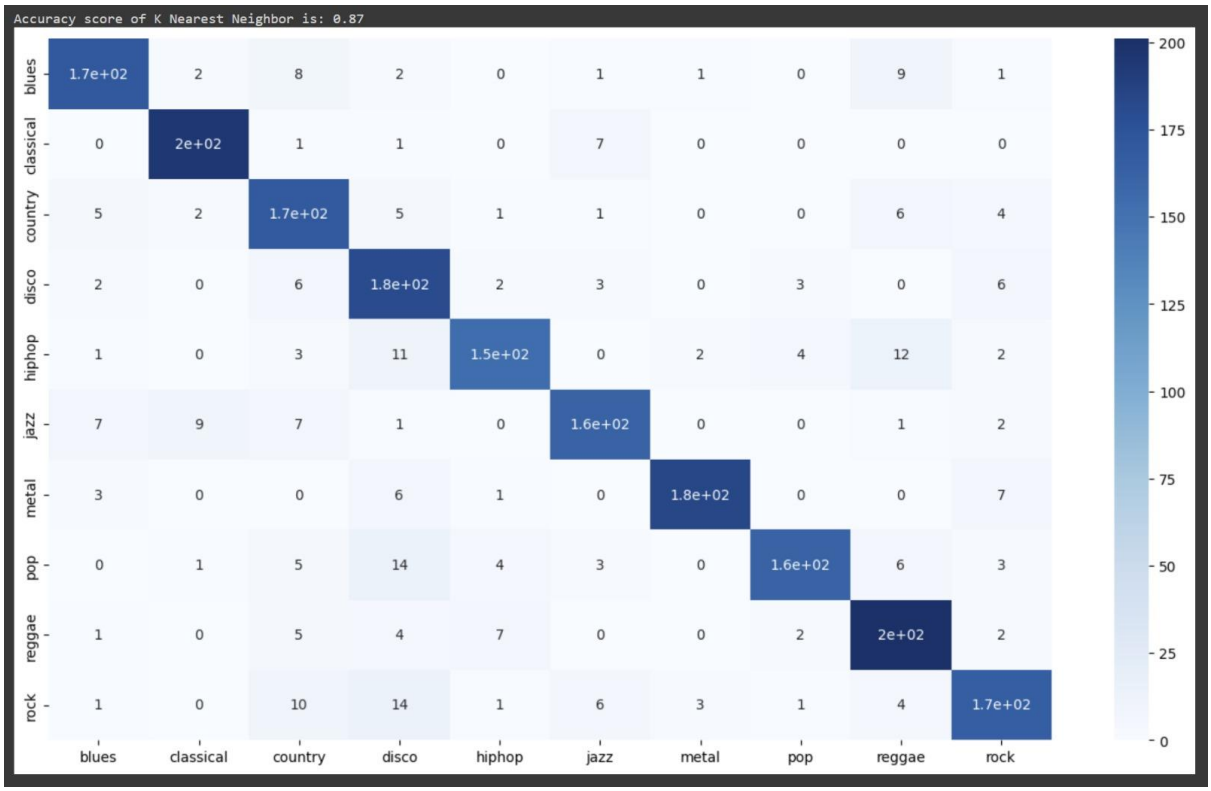
Accuracy of Random Forest classifier: 81%

K-nearest neighbour (KNN)

A lazy learning technique that classifies a data point based on its k-nearest neighbors' majority class. It works well for both binary and multiclass issues.

As training data, KNN uses a dataset with labeled examples. Each example in the dataset has a set of features (attributes) and a class label. When a new, unlabeled data point is supplied, KNN computes the distance between it and all other points in the training dataset. Euclidean distance, Manhattan distance, and other similarity measures are examples of common distance metrics. Based on the calculated distances, KNN selects the k data points (neighbors) from the training dataset that are closest to the new data point. KNN assigns the class label that is most

common among the k neighbors to the new data point for categorization. It is easier to avoid ties if K is an odd number.



Accuracy of KNN classifier: 87%

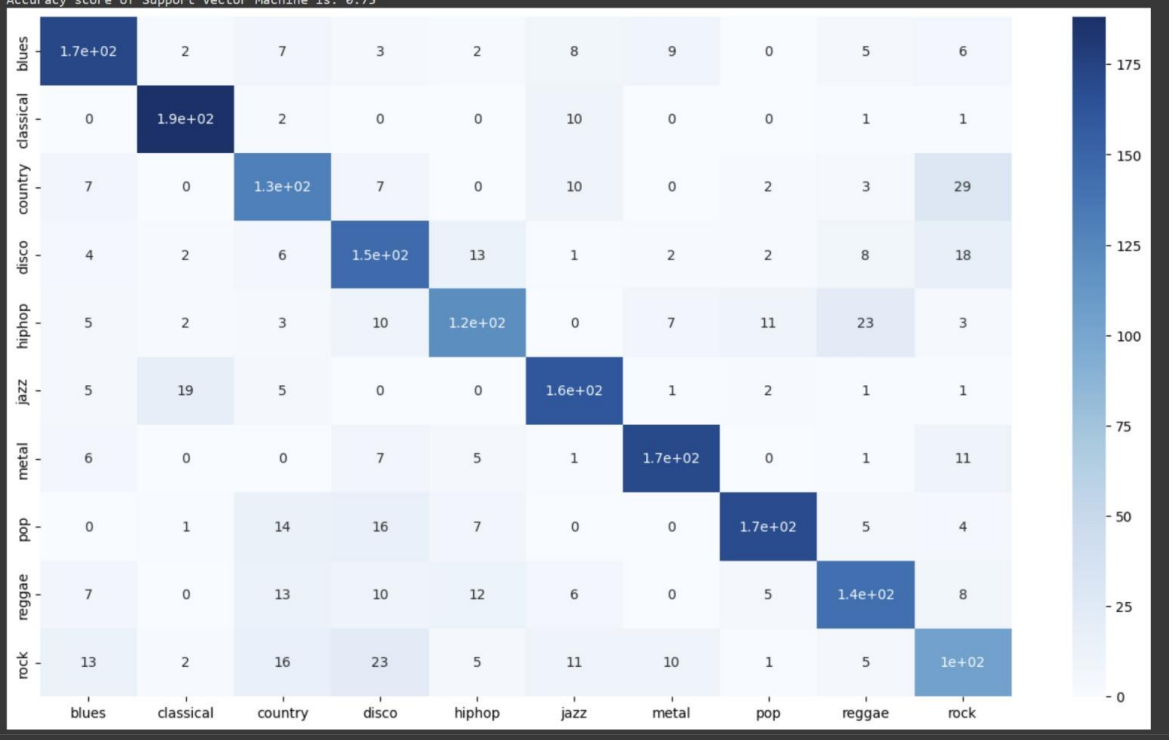
Support Vector Machines (SVM)

A model that finds a hyperplane in the feature space to distinguish various classes. SVMs can be used for binary as well as multiclass classification.

SVC-support vector classification (implementation of svm in scikit-learn. Decision_function_shape is set to ovo- one vs one. In one vs one a binary subproblem is created for each pair of classes. Model build- building svm model. This function trains model using k-fold cross-validation, evaluates its performance and displays the confusion matrix for the last split using a heatmap. Lists are initialized to store accuracy precision recall f1 scores for each fold during cross validation. This function iterates over the folds generated by the kfold object. For each fold it splits the data into training and testing sets, fits the model to the training data, predicta labels and calculates accuracy for that fold. After that, average accuracy is printed. Finally, this function computes the confusion matrix for the last fold and visualizes it using a heatmap. The heatmap is displayed using matplotlib and seaborn libraries

```
svm = SVC(decision_function_shape="ovo")
model_build(svm, split, 'Support Vector Machine')
```

```
Accuracy score of Support Vector Machine is: 0.75
```



Accuracy of SVM classifier: 75%

Gradient boosting models (XG Boost)

XGBoost (eXtreme Gradient Boosting) is a powerful and frequently used machine learning algorithm noted for its efficiency, flexibility, and success in a wide range of predictive modeling tasks. It is a member of the gradient boosting algorithm family, which combines weak learners (usually decision trees) consecutively to generate a powerful predictive model.

```
from sklearn.preprocessing import LabelEncoder

def model_build_xgb(model, kf, title = "Default"):
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y_encoded[train_index], y_encoded[test_index]
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy_scores.append(accuracy_score(y_test, y_pred))
        y_pred_labels = label_encoder.inverse_transform(y_pred)

    print("Accuracy score of", title, "is:", round(np.mean(accuracy_scores), 2))
    # Let's see the confusion matrix of the last split for a little insight
    con_mat = confusion_matrix(y_test, y_pred)
    plt.figure(figsize = (16, 9))
    sns.heatmap(con_mat, cmap="Blues", annot=True,
                xticklabels = ["blues", "classical", "country", "disco",
                               "hiphop", "jazz", "metal", "pop", "reggae", "rock"],
                yticklabels=["blues", "classical", "country", "disco", "hiphop",
                             "jazz", "metal", "pop", "reggae", "rock"])
    plt.show()


# Leave 2 blank spaces after a function definition
xgb = XGBClassifier()
model_build_xgb(xgb, split, 'XG Boost')
```

Import libraries:

Imports the LabelEncoder class from scikit-learn for encoding categorical labels.

Function Definition:

Defines a function named model_build_xgb that takes three parameters:

model: The XGBoost classifier.

kf: A scikit-learn KFold object for cross-validation.

title: A string specifying a title for the model (default is "Default").

Label encoding:

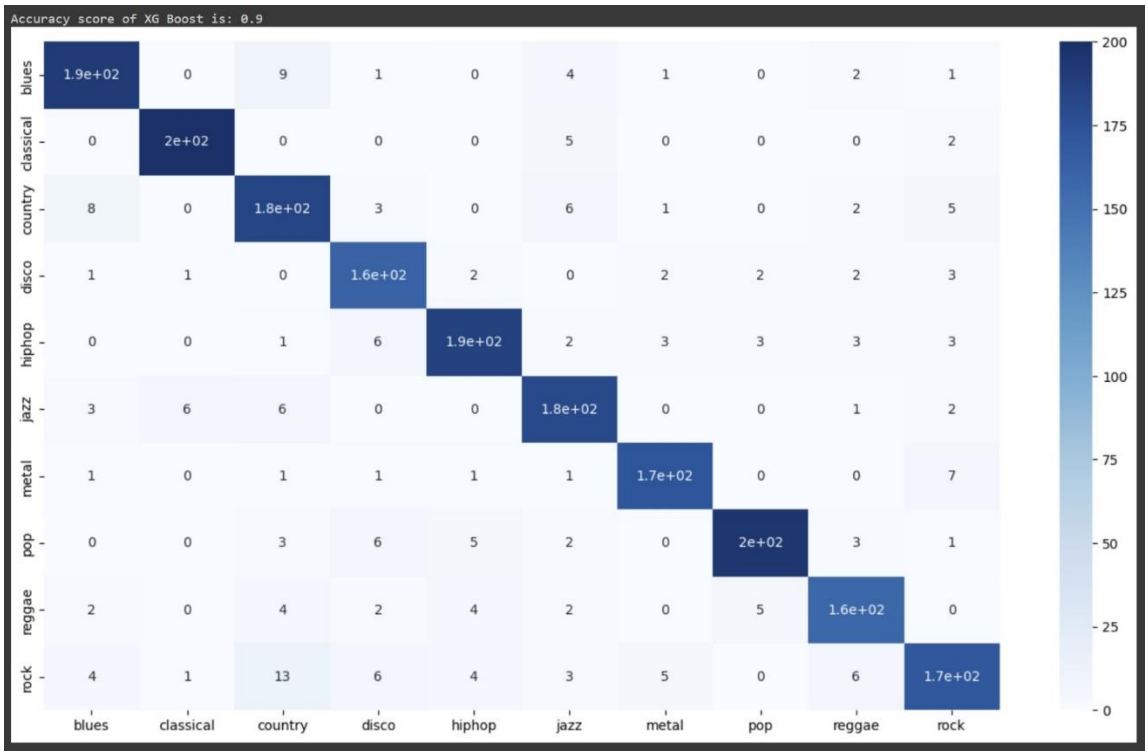
Creates a LabelEncoder object and encodes the target variable y into numeric labels. This is necessary for training the XGBoost model, which requires numeric labels.

Cross-validation loop:

Performs k-fold cross-validation using the provided KFold object (kf).
Splits the data into training and testing sets for each fold.
Fits the XGBoost model to the training data (X_train, y_train).
Predicts the labels for the test data (X_test).
Calculates the accuracy for each fold and appends it to accuracy_scores.

Confusion matrix visualization:

Calculates the confusion matrix for the last fold using confusion_matrix.
Visualizes the confusion matrix using seaborn's heatmap



Accuracy of XGBoost classifier: 90%

Convolutional neural network (CNN)

Convolutional Neural Networks (CNNs) are a type of deep neural network that has shown to be extremely effective at image and video recognition, natural language processing, and a variety of other tasks utilizing grid-like data. CNNs are especially well-suited for jobs involving spatial linkages and local patterns in the input data.

CNN Key Components:

Convolutional Layers:

CNNs are built on the foundation of convolutional layers. They use filters (also known as kernels) to apply convolution operations to incoming data. These filters scan the input for patterns such as edges, textures, or higher-level features.

Activation Functions:

After convolutional processes, activation functions such as ReLU (Rectified Linear Unit) are applied element-wise. They give the model non-linearity, letting it to learn more complicated patterns and relationships in the data.

Pooling Layers:

By down-sampling the spatial dimensions of the input, pooling layers help to lower the computational load and the number of parameters in the network. Max pooling and average pooling are two common pooling methods.

Fully connected layers:

Every neuron in one layer communicates with every neuron in the following layer via fully connected layers. In the final stages of a CNN, these layers are frequently employed to map the extracted features to the output classes.

Flattening:

The output of the convolutional and pooling layers is flattened into a vector before being passed to fully connected layers.

Dropout:

Dropout is a regularization technique that is used to prevent overfitting. It loses a percentage of connections at random during training to encourage the network to learn more robust characteristics.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 11, 64)       640

max_pooling2d (MaxPooling2   (None, 64, 6, 64)         0
D)

batch_normalization (Batch   (None, 64, 6, 64)         256
Normalization)

conv2d_1 (Conv2D)            (None, 62, 4, 32)         18464

max_pooling2d_1 (MaxPoolin   (None, 31, 2, 32)         0
g2D)

batch_normalization_1 (Bat   (None, 31, 2, 32)         128
chNormalization)

conv2d_2 (Conv2D)            (None, 30, 1, 32)         4128

max_pooling2d_2 (MaxPoolin   (None, 15, 1, 32)         0
g2D)

batch_normalization_2 (Bat   (None, 15, 1, 32)         128
chNormalization)

conv2d_3 (Conv2D)            (None, 15, 1, 16)         528

max_pooling2d_3 (MaxPoolin   (None, 8, 1, 16)          0
g2D)

batch_normalization_3 (Bat   (None, 8, 1, 16)          64
chNormalization)

flatten_2 (Flatten)          (None, 128)               0

dense_9 (Dense)              (None, 64)                8256

dropout_3 (Dropout)          (None, 64)                0

dense_10 (Dense)             (None, 10)                650

=================================================================
Total params: 33242 (129.85 KB)
Trainable params: 32954 (128.73 KB)
Non-trainable params: 288 (1.12 KB)
```

5. Conclusion

Across all models, using frequency based mel-spectrograms produced higher accuracy results. Whereas amplitude only provides information on intensity, or how “loud" a sound is, the frequency distribution over time provides information on the content of the sound. Additionally, mel-spectrograms are visual, and CNNs work better with pictures. The CNN performed the best, as we expected. It took the longest time to train as well, but the increase in accuracy justifies the extra computation cost. However, we were surprised to see the similarity in accuracy between the KNN, SVM, and feed-forward neural network.

In the future, we hope to experiment with other types of deep learning methods, given they performed the best. Given that this is time series data, some sort of RNN model may work well (GRU, LSTM, for example). We are also curious about generative aspects of this project, including some sort of genre conversion (in the same vein as generative adversarial networks which repaint photos in the style of Van Gogh, but for specifically for music). Additionally, we suspect that we may have opportunities for transfer learning, for example in classifying music by artist or by decade.

6. Future Enhancements

Advanced Neural Networks:

Explore architectures beyond CNN, like RNNs or attention mechanisms, for improved pattern recognition.

Dynamic Recommendation System:

Implement real-time adaptation to user preferences for a more engaging user experience.

Multi-Modal Data Fusion:

Combine data from various sources (lyrics, user history) for a comprehensive understanding of preferences.

Transfer Learning:

Fine-tune models using pre-trained ones on larger datasets to leverage broader musical context.

Real-Time Audio Analysis:

Enable the system to analyze and recommend music as it's released to stay updated on trends.

User Interaction Features:

Incorporate user behavior analysis (skips, playlists) to refine the recommendation algorithm.

Explainability and Interpretability:

Provide insights into why certain songs are recommended for user trust and understanding.

Cross-Domain Recommendations:

Extend recommendations beyond genres to include concerts, merchandise, or related events.

7. References

- https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification/data
- https://github.com/chittalpatel/Music-Genre-Classification-GTZAN
- https://colab.research.google.com/drive/1pH27AUhPAzn_KAMjRLizlbZGjQn-SZCu#scrollTo=QL-7J9qi_P4J
- https://www.chosic.com/list-of-music-genres/