MAR ATHANASIUS COLLEGE OF ENGINEERING, KOTHAMANGALAM

Initial Project Report

MUSIC GENRE IDENTIFICATION USING MACHINE LEARNING

Done by

AKHIL ROCK BABU

Reg No: MAC23MCA-2010

Under the guidance of

Prof. Manu John

ABSTRACT

The project aims to create a music genre classification system using machine learning algorithms, specifically k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB), to accurately predict the genre of unseen audio files using the GTZAN dataset and Python Librosa package.

The music industry leverages genre classification for enhanced user experience, personalized content, and targeted audience targeting in recommendation engines. This system is crucial in organizing digital music libraries, enhancing user experience, and enhancing the effectiveness of music producers and marketers.

Machine learning algorithms have been found to be effective in classifying music genres. Studies have shown that SVM with an RBF kernel is accurate by 87.5%, while KNN outperforms other models with an accuracy of 92.69%, surpassing CNN's 72.40%. KNN's high accuracy and efficiency make it a leading algorithm for this task.

The GTZAN dataset will be used to evaluate three machine learning algorithms for identifying music genres: KNN, SVM, and NB. The KNN algorithm will be evaluated for accuracy, computational efficiency, and robustness. SVM will be assessed for high-dimensional data handling, while Naive Bayes will be considered for its rapid and probabilistic approach. The goal is to find the most accurate algorithm.

The comprehensive dataset includes 10,000 pre-extracted features from 1,000 audio tracks, categorized into 10 music genres. These features, extracted using the Librosa package, are normalized and used to train machine learning models, ensuring a reliable classification system for each genre.

References:

1. Ndou, N., Ajoodha, R., & Jadhav, A. (2021). Music Genre Classification: A review of Deep-Learning and Traditional Machine-Learning Approaches. 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS). https://doi.org/10.1109/iemtronics52119.2021.9422487
2. Ghildiyal, A., Singh, K., & Sharma, S. (2020). Music Genre Classification using Machine Learning. 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA). https://doi.org/10.1109/iceca49313.2020.9297444
3. Setiadi, D. R. I. M., Rahardwika, D. S., Rachmawanto, E. H., Sari, C. A., Irawan, C., Kusumaningrum, D. P., Nuri, N., & Trusthi, S. L. (2020). Comparison of SVM, KNN, and NB Classifier for Genre Music Classification based on Metadata. 2020 International Seminar on Application for Technology of Information and Communication (iSemantic). https://doi.org/10.1109/isemantic50169.2020.9234199

Submitted By:
Akhil Rock Babu
Reg No: M23CA011
2023 –25 Batch MCA Department, MACE

Faculty Guide:
Prof. Manu John
Associate Professor S3 MCA

INTRODUCTION

In the rapidly evolving landscape of digital music, the efficient categorization of music genres is pivotal for enhancing user engagement and optimizing content delivery. Music genre classification facilitates personalized recommendations, organizes extensive digital music libraries, and supports targeted marketing strategies. As such, developing a robust and accurate music genre classification system is essential for both users and industry professionals.

This project seeks to address this need by employing advanced machine learning techniques to classify music genres. Specifically, it explores the performance of three prominent algorithms: k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB). The evaluation will be based on the GTZAN dataset, a well-regarded benchmark in music genre classification research, and the Python Librosa package, a powerful tool for audio analysis.

The GTZAN dataset comprises 10,000 pre-extracted feature records from 1,000 audio tracks, each 30 seconds in duration and segmented into 3-second intervals. This dataset encompasses 10 distinct genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. Key features of interest include Mel Frequency Cepstral Coefficients (MFCCs), chroma features, zero-crossing rate, and tempo. These features provide a comprehensive representation of the audio data, capturing essential musical characteristics that are critical for genre classification.

Using the Python Librosa package, features from the GTZAN dataset will be meticulously extracted and normalized. This preprocessing step is crucial for ensuring the accuracy and reliability of the subsequent machine learning models. The KNN, SVM, and Naive Bayes algorithms will then be applied to this processed data, each evaluated for its effectiveness in predicting music genres.

Prior studies have demonstrated varying levels of success among these algorithms. SVM with an RBF kernel has achieved an accuracy of approximately 87.5%, showcasing its capability to handle complex data patterns. KNN, however, has consistently delivered superior performance with an accuracy of 92.69%, outperforming other models, including Convolutional Neural Networks (CNNs) which achieved an accuracy of 72.40%. This high accuracy makes KNN a particularly strong candidate for music genre classification. Naive Bayes, with its probabilistic approach, offers rapid classification capabilities, making it an attractive option for certain applications.

In this project, the KNN algorithm will be assessed for its precision, computational efficiency, and scalability. SVM will be evaluated for its performance with high-dimensional feature spaces, and Naive Bayes will be analyzed for its speed and probabilistic classification strengths. By comparing these algorithms, the project aims to identify the most effective method for genre classification, contributing to advancements in automated music analysis and enhancing the overall functionality of music recommendation systems.

The findings of this project will not only provide insights into the relative strengths of these machine learning algorithms but also offer practical recommendations for their application in real-world music classification tasks. Ultimately, the goal is to develop a system that significantly improves the accuracy and efficiency of music genre classification, thereby enhancing user experiences and supporting the broader music industry.

LITERATURE REVIEW

Paper 1: Music Genre Classification: A Review of Deep-Learning and Traditional Machine-Learning Approaches

The paper compares traditional machine learning algorithms like KNN, SVM, and Naive Bayes (NB) with deep learning algorithms like CNN. KNN outperforms SVM and CNN by 92.69%, but KNN is sensitive to noise, SVM is effective but computationally expensive, and CNN requires a lot of data and resources. The authors suggest studying hybrid models and using standardized data and evaluation methods for more reliable studies.

Title of the paper	Ndou, N., Ajoodha, R., & Jadhav, A. (2021). Music Genre Classification: A review of Deep-Learning and Traditional Machine-Learning Approaches. 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS). https://doi.org/10.1109/iemtronics52119.2021.9422487
Area of work	This review is about music information retrieval, especially genre classification.
Dataset	The review looks at different datasets, such as the GTZAN dataset, which is used to compare music genre classification models. The GTZAN file has 60 columns and 10000 entries. https://www.kaggle.com/sets/andradaolteanu/gtzan-dataset-music-genre-classification
Methodology/Strategy	The paper looks at different ways to classify music. This means talking about how to find features, like MFCCs and chroma features, and comparing how well different classification algorithms work.
Algorithm	Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Convolutional Neural Network (CNN)
Result/Accuracy	**KNN: 92.69%** CNN: 72.40% SVM :80.80%
Advantages	This compares different methods and shows how machine learning can help find and classify features more accurately with extracted features.
Future Proposal	Suggests more research into hybrid models that combine deep learning with traditional methods and explore more diverse datasets for better generalization.

Paper 2: Music Genre Classification using Machine Learning

The paper explores the use of machine learning algorithms for music genre classification, focusing on SVM with a Radial Basis Function (RBF) kernel, K-Nearest Neighbors (KNN), and Random Forest (RF). The results show that SVM with an RBF kernel achieved an accuracy of 87.5%, highlighting its high accuracy and ability to handle non-linear data. However, it can be computationally expensive. The authors suggest future research should integrate advanced feature extraction techniques and hybrid models to enhance classification performance.

Title of the paper	Ghildiyal, A., Singh, K., & Sharma, S. (2020). Music Genre Classification using Machine Learning. 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA). https://doi.org/10.1109/iceca49313.2020.9297444
Area of work	This research falls under the domain of music information retrieval, specifically targeting the classification of music into genres using machine learning algorithms.
Dataset	The study utilizes the GTZAN dataset, which is a standard dataset for music genre classification containing 60 columns and 10000 entries. https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
Methodology/Strategy	The methodology involves feature extraction from the audio tracks using techniques like MFCCs, chroma features, and others. The features that are extracted are used to train machine learning models to classify music genres.
Algorithm	Support Vector Machine (SVM), K-Nearest Neighbors (KNN) and Random Forest (RF).
Result/Accuracy	**SVM: 87.5%** RF: 69.6% KNN: 66.4%
Advantages	The study compares different machine learning algorithms, showing which ones work best for music genre classification.
Limitations	The paper mentions potential overfitting issues with more complex models and points out the limitations of the GTZAN dataset, which may not be large enough to capture all the nuances of music genres.
Future Proposal	For future work, the authors suggest using deep learning techniques that have been successful in other studies for figuring out music. They also recommend using larger and more diverse datasets to make the classification models more robust and general.

Paper 3: Comparison of SVM, KNN, and NB Classifier for Genre Music Classification based on Metadata

The paper compares three machine learning algorithms: SVM, KNN, and NB for music genre classification based on metadata. The results show that SVM with a Radial Basis Function (RBF) kernel achieved an accuracy of 80%. KNN and NB also performed well, with KNN achieving competitive results but slightly lower accuracy than SVM. The authors propose further research into optimizing these algorithms and combining them with advanced feature extraction methods to improve classification accuracy.

Title of the paper	Setiadi, D. R. I. M., Rahardwika, D. S., Rachmawanto, E. H., Sari, C. A., Irawan, C., Kusumaningrum, D. P., Nuri, N., & Trusthi, S. L. (2020). Comparison of SVM, KNN, and NB Classifier for Genre Music Classification based on Metadata. 2020 International Seminar on Application for Technology of Information and Communication (iSemantic). https://doi.org/10.1109/isemantic50169.2020.9234199
Area of Work	The paper falls under the domain of music information retrieval, specifically focusing on genre classification based on metadata.
Dataset	The study uses metadata features extracted from Spotify music dataset from www.crowdai.org
Methodology/Strategy	The authors extracted metadata features and applied three classifiers—SVM, KNN, and NB—to classify music genres. They compared the performance of these classifiers to determine which one is most effective for this task.
Algorithm	Support Vector Machine (SVM) K-Nearest Neighbours (KNN) Naive Bayes classifier (NB)
Result/Accuracy	**SVM: 80%** KNN: 75.61% NB: 75.05%
Advantages	The study compares multiple classifiers to find the best one for music genre classification based on metadata.
Limitations	Using only metadata might not capture the full complexity of music genres. This limitation suggests that the classifiers might not work as well as they would with a more complete feature set.
Future Proposal	The authors suggest that future research could use metadata and audio features to improve classification accuracy by using the strengths of both types of data.

LITERATURE SUMMARY

Machine learning algorithms have been found to be effective in classifying music genres. Studies have shown that SVM with an RBF kernel is accurate by 87.5%, while KNN outperforms other models with an accuracy of 92.69%, surpassing CNN's 72.40%.

	TITLE	DATASET	ALGORITHM	ACCURACY
PAPER 1	Ndou, N., Ajoodha, R., & Jadhav, A. (2021). Music Genre Classification: A review of Deep-Learning and Traditional Machine-Learning Approaches. 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS). https://doi.org/10.1109/iemtronics52119.2021.9422487	GTZAN dataset 10,000 Records of 60 features.	K-Nearest Neighbors (KNN) Convolutional Neural Network (CNN) Support Vector Machine (SVM)	**KNN: 92.69%** CNN: 72.40% SVM :80.80%
PAPER 2	Ghildiyal, A., Singh, K., & Sharma, S. (2020). Music Genre Classification using Machine Learning. 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA). https://doi.org/10.1109/iceca49313.2020.9297444	GTZAN dataset 10,000 Records of 60 features.	Support Vector Machine (SVM) Random Forest (RF) K-Nearest Neighbors (KNN)	**SVM: 87.5%** RF: 69.6% KNN: 66.4%
PAPER 3	Setiadi, D. R. I. M., et al . (2020). Comparison of SVM, KNN, and NB Classifier for Genre Music Classification based on Metadata. 2020 International Seminar on Application for Technology of Information and Communication https://doi.org/10.1109/isemantic50169.2020.9234199	The study uses metadata features extracted from Spotify music dataset from www.crowdai.org	Support Vector Machine (SVM) K-Nearest Neighbours (KNN) Naive Bayes classifier (NB)	**SVM: 80%** KNN: 75.61% NB: 75.05%

PROPOSED MODEL

Music Genre Classification Using k-Nearest Neighbors (KNN)

Introduction

Music genre classification is an important part of music information retrieval systems. It affects music recommendation engines, playlist automation, and music library organization. k-Nearest Neighbors (KNN) has shown great performance with short audio features.

Objective

To develop and optimize a music genre classification system using the k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB) algorithms, and compare their accuracy and efficiency to demonstrate the superior performance of the best algorithm among the three.

Background and Motivation:

Recent research highlights the effectiveness of various algorithms in music genre classification, with some achieving higher accuracy than many traditional and deep learning models. For example, in the study by Ndou, Ajoodha, & Jadhav (2021), KNN achieved an impressive accuracy of 92.69%, significantly outperforming Convolutional Neural Networks (CNNs). This study aims to explore and compare the performance of three different algorithms: k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB), in terms of their accuracy and efficiency in music genre classification.

Why Choose KNN, SVM, and NB:

- **High Accuracy:** KNN, SVM, and NB have all demonstrated high accuracy in various classification tasks, making them strong candidates for music genre classification.
- **Simplicity and Efficiency:** These algorithms are easy to implement and computationally efficient, making them suitable for real-time applications.
- **Effectiveness with Short-Duration Features:** KNN, SVM, and NB perform exceptionally well with short-duration audio features, which are crucial for timely and accurate genre classification.

Methodology:

1. Data Collection:

Use the GTZAN dataset and extract relevant audio features, such as Mel Frequency Cepstral Coefficients (MFCCs), chroma, and spectrograms, using the Python Librosa package, and collect the preprocessed feature data into a CSV file.

2. Model Development:

1. Choosing the Algorithms: Based on literature review, k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB) algorithms are chosen due to their high accuracy and efficiency.
2. Training the Models: The KNN, SVM, and NB models are trained using the feature matrix and corresponding labels. Cross-validation is used to optimize the parameters for each algorithm
3. Validation: The performance of each model is validated using 10-fold cross-validation to ensure robustness and avoid overfitting.
4. Hyperparameter Tuning: Each model is fine-tuned by adjusting hyperparameters to achieve the best performance.

4. Prediction Process:

1. Input: An unseen audio file is input into the system.
2. Feature Extraction: Features are extracted from the input audio file using the same methods (Librosa) as during training.
3. Normalization: The extracted features are normalized to match the scale of the training data.
4. Model Prediction: The normalized feature vector is passed to the trained models (KNN, SVM, and NB). Each model makes a prediction based on its specific algorithm.
5. Class Label Assignment: The models predict the genre of the input audio file based on their respective algorithms:
    - KNN: Calculates the distance between the input vector and all training samples, identifying the k-nearest neighbors and predicting the genre based on the majority class among them.
    - SVM: Uses the optimized hyperplane to classify the input vector.
    - NB: Uses the probability distributions learned during training to classify the input vector.

Conclusion:

This project leverages the GTZAN dataset and compares three algorithms—k-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB)—to develop a high-accuracy music genre classification system. By systematically extracting and normalizing relevant audio features, training robust models, and optimizing their parameters, the system is capable of accurately predicting the genre of new, unseen audio files. The combination of high accuracy, computational efficiency, and scalability makes this approach a powerful solution for music genre classification, with potential applications in music recommendation systems, automated playlist generation, and digital music libraries.

DATASET DESCRIPTION

Dataset Overview:

The GTZAN dataset is a widely used benchmark for music genre classification tasks. It contains 1,000 audio tracks each 30 seconds long, divided into 10 genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. Each genre has 100 tracks, making the dataset balanced and suitable for classification tasks.

Source:

The GTZAN dataset is publicly available on Kaggle and was originally compiled by George Tzanetakis in 2002. It is a go-to dataset for researchers and practitioners working on music genre classification.

filename	length	chroma_stft_mean	chroma_stft_var	rms_mean
blues.00000.0.wav	66149	0.3354063630104065	0.091048293	0.13040502369403
blues.00000.1.wav	66149	0.3430653512477875	0.086146526	0.11269924789667
blues.00000.2.wav	66149	0.34681475162506104	0.092242889	0.13200338184833
blues.00000.3.wav	66149	0.3636387884616852	0.086856157	0.13256472349166
blues.00000.4.wav	66149	0.33557942509651184	0.088128544	0.14328880608081
blues.00000.5.wav	66149	0.3766697347164154	0.089702107	0.13261780142784

Features:

For music genre classification, various audio features can be extracted from the raw audio files. Using the Python Librosa package, we can extract the following key features:

- Chroma Feature (chroma_stft): 12 coefficients representing the energy distribution across the 12 different pitch classes.

- Root Mean Square Value (rms): Represents the power of the audio signal.

- Spectral Centroid (spectral_centroid): Indicates where the center of mass of the spectrum is located.

- Spectral Bandwidth (spectral_bandwidth): Measures the width of the band of frequencies.

- Spectral Contrast (spectral_contrast): The difference in amplitude between peaks and valleys in the sound spectrum.

- Spectral Rolloff (spectral_rolloff): The frequency below which a specified percentage of the total spectral energy lies.

- Zero Crossing Rate (zero_crossing_rate): The rate at which the signal changes sign.

- Harmony and Perceived Pitch (harmony, perceptr): Represent harmony and pitch features.

- Tempo (tempo): The estimated tempo of the music.

-MFCCs (mfcc1-mfcc20): 20 coefficients representing the Mel Frequency Cepstral Coefficients.

Class Labels:

The dataset is labeled with 10 distinct music genres:

1. Blues
2. Classical
3. Country
4. Disco
5. Hiphop
6. Jazz
7. Metal
8. Pop
9. Reggae
10. Rock

EXPLORATORY ANALYSIS

```
#determining the volume of dataset
data.shape
```

```
(9990, 60)
```

Figure 1

Dataset consists of 9990 rows and 60 columns.

#	Column	Non-Null Count	Dtype
0	filename	9990 non-null	object
1	length	9990 non-null	int64
2	chroma_stft_mean	9990 non-null	float64
3	chroma_stft_var	9990 non-null	float64
4	rms_mean	9990 non-null	float64
5	rms_var	9990 non-null	float64
6	spectral_centroid_mean	9990 non-null	float64
7	spectral_centroid_var	9990 non-null	float64
8	spectral_bandwidth_mean	9990 non-null	float64
9	spectral_bandwidth_var	9990 non-null	float64
10	rolloff_mean	9990 non-null	float64
11	rolloff_var	9990 non-null	float64
12	zero_crossing_rate_mean	9990 non-null	float64
13	zero_crossing_rate_var	9990 non-null	float64
14	harmony_mean	9990 non-null	float64
15	harmony_var	9990 non-null	float64
16	perceptr_mean	9990 non-null	float64
17	perceptr_var	9990 non-null	float64
18	tempo	9990 non-null	float64
19	mfcc1_mean	9990 non-null	float64
20	mfcc1_var	9990 non-null	float64
21	mfcc2_mean	9990 non-null	float64
22	mfcc2_var	9990 non-null	float64
23	mfcc3_mean	9990 non-null	float64
24	mfcc3_var	9990 non-null	float64
25	mfcc4_mean	9990 non-null	float64
26	mfcc4_var	9990 non-null	float64
27	mfcc5_mean	9990 non-null	float64
28	mfcc5_var	9990 non-null	float64
29	mfcc6_mean	9990 non-null	float64
30	mfcc6_var	9990 non-null	float64
31	mfcc7_mean	9990 non-null	float64
32	mfcc7_var	9990 non-null	float64
33	mfcc8_mean	9990 non-null	float64
34	mfcc8_var	9990 non-null	float64
35	mfcc9_mean	9990 non-null	float64
36	mfcc9_var	9990 non-null	float64
37	mfcc10_mean	9990 non-null	float64
38	mfcc10_var	9990 non-null	float64
39	mfcc11_mean	9990 non-null	float64
40	mfcc11_var	9990 non-null	float64
41	mfcc12_mean	9990 non-null	float64
42	mfcc12_var	9990 non-null	float64
43	mfcc13_mean	9990 non-null	float64
44	mfcc13_var	9990 non-null	float64
45	mfcc14_mean	9990 non-null	float64
46	mfcc14_var	9990 non-null	float64
47	mfcc15_mean	9990 non-null	float64
48	mfcc15_var	9990 non-null	float64
49	mfcc16_mean	9990 non-null	float64
50	mfcc16_var	9990 non-null	float64
51	mfcc17_mean	9990 non-null	float64
52	mfcc17_var	9990 non-null	float64
53	mfcc18_mean	9990 non-null	float64
54	mfcc18_var	9990 non-null	float64
55	mfcc19_mean	9990 non-null	float64
56	mfcc19_var	9990 non-null	float64
57	mfcc20_mean	9990 non-null	float64
58	mfcc20_var	9990 non-null	float64
59	label	9990 non-null	object

Figure 2

Dataset consists of 58 Numerical attributes and 2 non-numerical attributes.

label
blues
jazz
metal
pop
reggae
disco
classical
hiphop
rock
country

Figure 3

Dataset consists of 10 unique Class Labels

DATA PREPROCESSING

Missing Values

```
#checking for null values
print(data.isnull().sum())
```

```
chroma_stft_var           0
rms_mean                  0
rms_var                   0
spectral_centroid_mean    0
spectral_centroid_var     0
spectral_bandwidth_mean   0
spectral_bandwidth_var    0
rolloff_mean              0
rolloff_var               0
zero_crossing_rate_mean   0
zero_crossing_rate_var    0
harmony_mean              0
harmony_var               0
perceptr_mean             0
perceptr_var              0
tempo                     0
mfcc1_mean                0
mfcc1_var                 0
mfcc2_mean                0
mfcc2_var                 0
mfcc3_mean                0
mfcc3_var                 0
mfcc4_mean                0
mfcc4_var                 0
mfcc5_mean                0
mfcc5_var                 0
mfcc6_mean                0
mfcc6_var                 0
mfcc7_mean                0
mfcc7_var                 0
mfcc8_mean                0
mfcc8_var                 0
mfcc9_mean                0
mfcc9_var                 0
mfcc10_mean               0
mfcc10_var                0
mfcc11_mean               0
mfcc11_var                0
mfcc12_mean               0
mfcc12_var                0
mfcc13_mean               0
mfcc13_var                0
mfcc14_mean               0
mfcc14_var                0
mfcc15_mean               0
mfcc15_var                0
mfcc16_mean               0
mfcc16_var                0
mfcc17_mean               0
mfcc17_var                0
mfcc18_mean               0
mfcc18_var                0
mfcc19_mean               0
mfcc19_var                0
mfcc20_mean               0
mfcc20_var                0
label                     0
```

Figure 4

There are no missing values in the dataset.

Feature Selection

Feature selection is crucial for music classification, as it helps reduce complexity, time, and computational power. It removes irrelevant features, reduces noise, and reduces overfitting. By choosing only relevant features, we can save time, money, and resources, and achieve better accuracy for our model. Overloading with irrelevant variables can lead to overfitting and poor predictions.

Feature selection using Random Forest and XGBoost

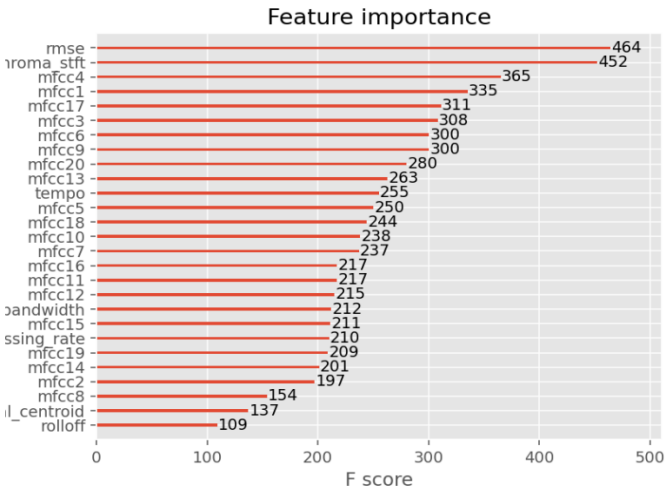


Figure 5

By analyzing feature importance through Random Forest and XGBoost, we have identified the top 15 features that contribute most significantly to the classification task.

The following 15 features were identified as the most important for classifying music genres, based on their F-scores in the feature importance graph:

1. **rms_mean:** The root mean square value of the audio signal, reflecting the energy level.
2. **chroma_stft_mean:** The mean value of the chroma short-time Fourier transform, indicating harmonic content.
3. **mfcc4_mean:** The mean of the 4th Mel-frequency cepstral coefficient, capturing timbral aspects of the audio.
4. **mfcc1_mean:** The mean of the 1st Mel-frequency cepstral coefficient, often correlated with overall spectral shape.
5. **mfcc17_mean:** The mean of the 17th Mel-frequency cepstral coefficient, providing higher order cepstral information.
6. **mfcc3_mean:** The mean of the 3rd Mel-frequency cepstral coefficient, another timbral feature.
7. **mfcc6_mean:** The mean of the 6th Mel-frequency cepstral coefficient, contributing to timbre analysis.
8. **mfcc9_mean:** The mean of the 9th Mel-frequency cepstral coefficient, also important in timbre.
9. **mfcc20_mean:** The mean of the 20th Mel-frequency cepstral coefficient, capturing the finest details in timbre.
10. **mfcc13_mean:** The mean of the 13th Mel-frequency cepstral coefficient, bridging mid-level timbral information.
11. **tempo:** The tempo of the music track, a critical feature in distinguishing genres like jazz, rock, and classical.
12. **mfcc5_mean:** The mean of the 5th Mel-frequency cepstral coefficient, further enriching the timbral representation.
13. **mfcc18_mean:** The mean of the 18th Mel-frequency cepstral coefficient, capturing specific timbral nuances.
14. **mfcc10_mean:** The mean of the 10th Mel-frequency cepstral coefficient, contributing to detailed timbral characterization.
15. **mfcc7_mean:** The mean of the 7th Mel-frequency cepstral coefficient, adding to the diverse timbral information.

By narrowing down the feature set to these 15 key features, we aim to enhance the model's performance while reducing computational complexity. This selection process not only simplifies the model but also focuses on the most informative attributes, leading to potentially higher classification accuracy.

Handling Outliers

- **rms_mean**

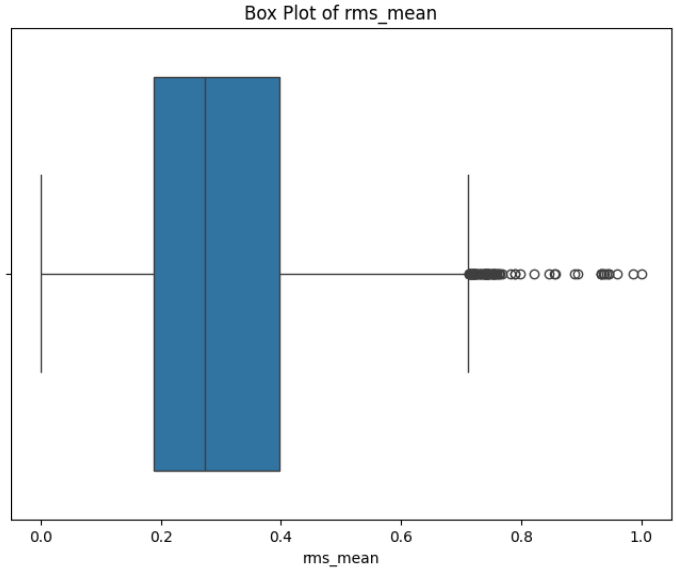


Figure 6
Type: Boxplot
x-label: The range of values of rms_mean

rms_mean above 0.78 are considered as outliers and is they are removed.

- **chroma_stft_mean**

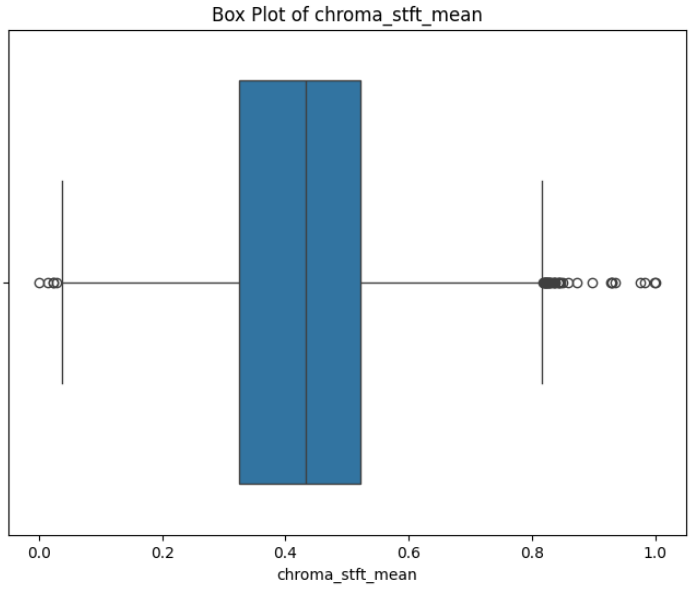


Figure 7
Type: Boxplot
x-label: The range of values of chroma_stft_mean

chroma_stft_mean above 0.85 and below 0.03 are considered as outliers and they are removed.

- **mfcc4_mean**

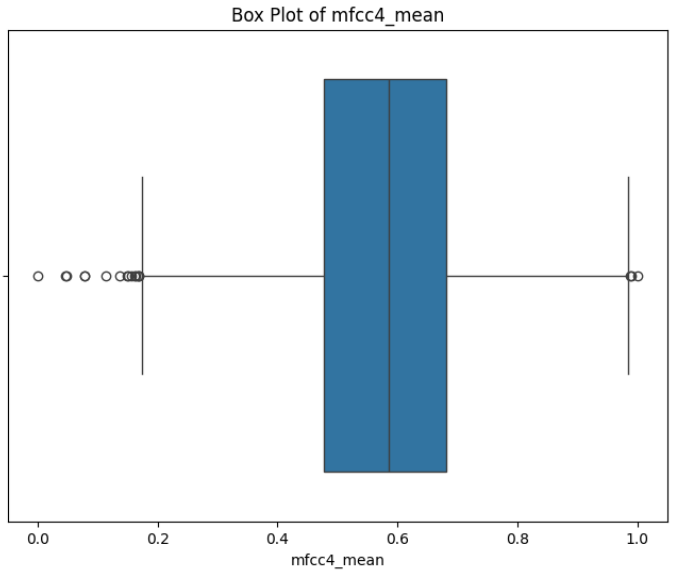


Figure 8
Type: Boxplot
x-label: The range of values of mfcc4_mean

mfcc4_mean below 0.18 are considered as outliers and they are removed.

- **mfcc17_mean**

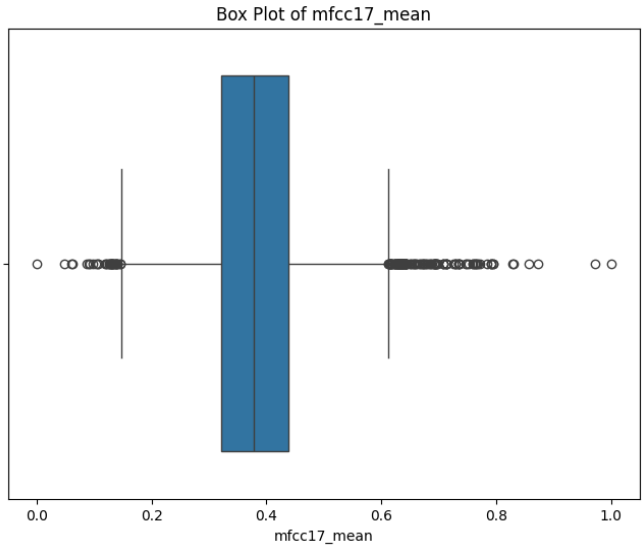
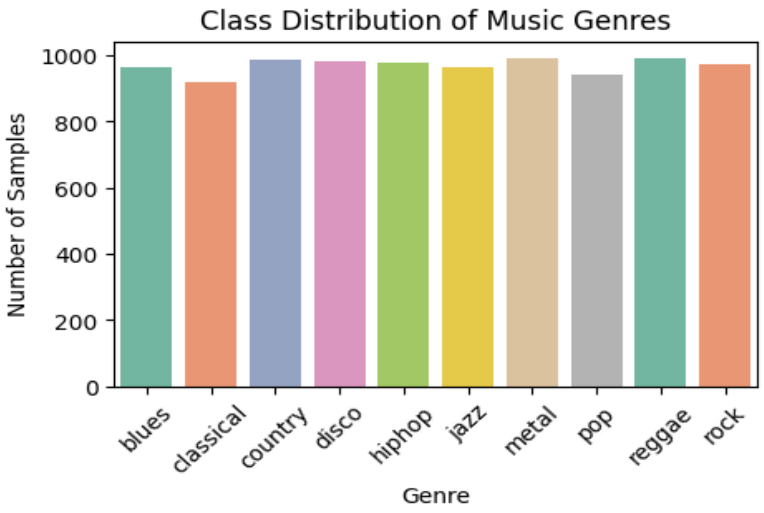


Figure 9
Type: Boxplot
x-label: The range of values of mfcc17_mean

mfcc17_mean above 0.8 and below 0.1 are considered as outliers and they are removed.

Before moving forward with model training, I made sure to check for and handle any outliers in the 15 key features we've selected. Outliers can mess with a model's performance, so I took care of them to ensure our data is solid and ready to deliver accurate predictions.

Handling Class Imbalance




```
data['label'].value_counts()
```

label	count
metal	993
reggae	989
country	988
disco	984
hiphop	979
rock	974
blues	964
jazz	963
pop	943
classical	918

Figure 10
Type: Count plot
x-label: Genre
y-label: No of Samples

In our dataset of 9,990 records, we assessed the distribution of samples across various music genres. The analysis revealed minimal differences between the largest and smallest classes. The small variation is not significant enough to be considered a class imbalance. Therefore, we can confidently proceed with model training without the need for any adjustments related to class distribution.

TRENDS AND PATTERNS

- **rms_mean**

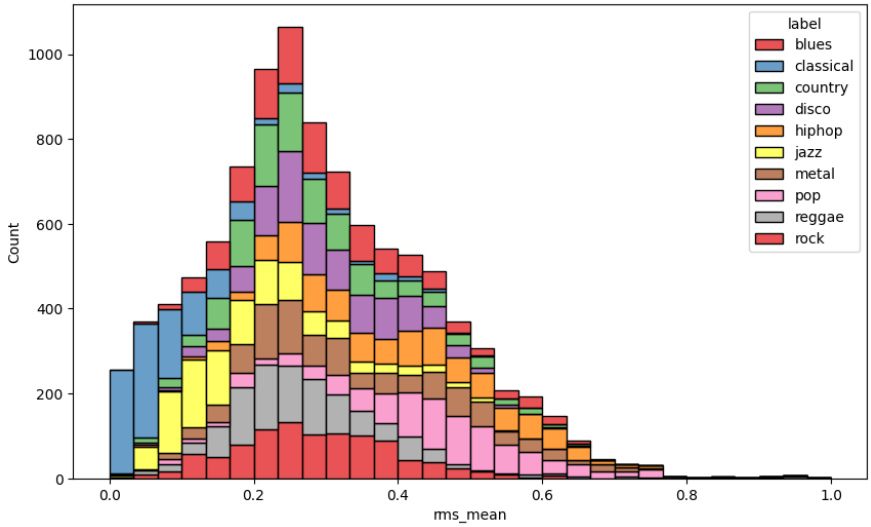


Figure 11
Type: histogram
x-label: rms_mean
y-label: count of music genres
Legend: different colours show different music genres

- The overall distribution of rms_mean seems to be approximately normal (bell-shaped), peaking around 0.2. This suggests that most audio files have an rms_mean value in this range.

- Classical Music: The classical genre shows a strong presence on the lower end of the rms_mean spectrum, indicating that classical music tends to have lower rms_mean values. This makes sense as classical music often has quieter passages and less intense sound energy compared to other genres.

- Rock and Metal: The rock and metal genres show more distribution towards the higher rms_mean values, reflecting their typically louder and more intense sound profiles.

- Pop and Disco: The pop and disco genres also have a noticeable presence around the central peak but are distributed across a wider range, indicating a mix of sound intensities.

- The blues genre is fairly uniformly distributed across the range but seems slightly more concentrated in the central region.

Perform further analysis using other features or a combination of features to better separate the overlapping regions.

- **rms_mean vs chroma_stft_mean**

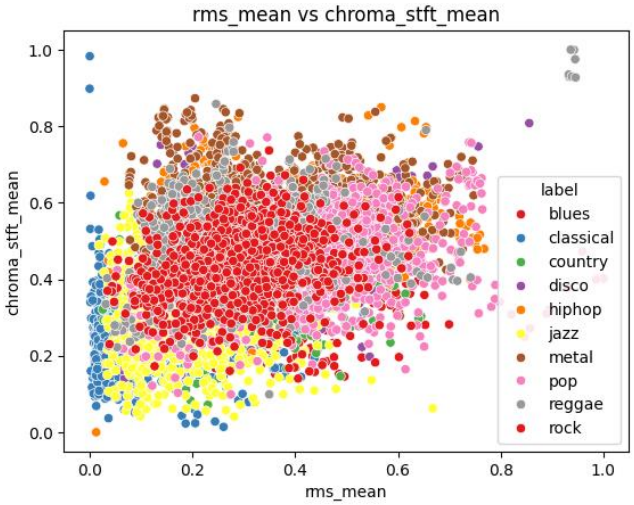


Figure 13
Type: Scatter plot
x-label: rms_mean
y-label: chroma_stft_mean
Legend: different colours show different music genres

  - Classical and reggae show more distinct clustering, which could be useful for classification.

  - Rock, pop, jazz, and disco show significant overlap, indicating the need for additional features to differentiate these genres effectively.

- **mfcc4_mean vs mfcc1_mean**

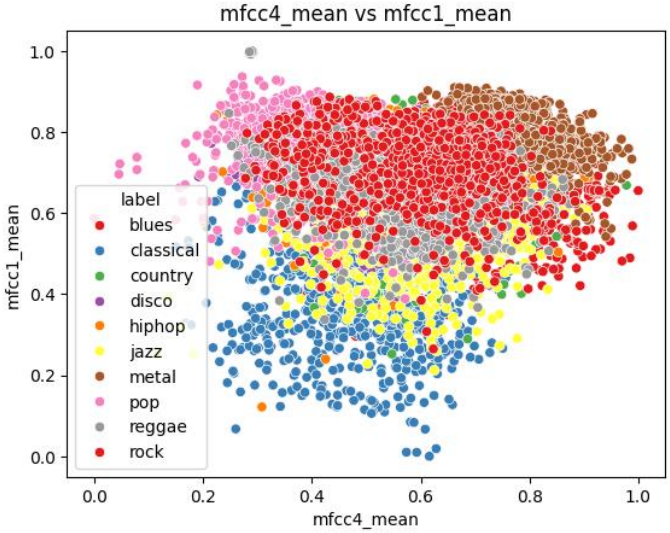


Figure 14
Type: Scatter plot
x-label: mfcc4_mean
y-label: mfcc1_mean
Legend: different colours show different music genres

- Certain genres, such as classical, rock, and blues, show more distinct clustering patterns. This indicates that mfcc4_mean and mfcc1_mean can partially separate these genres, suggesting that these features are relevant for distinguishing these specific genres.

- There is noticeable overlap between genres like reggae and blues. This suggests that these two features alone might not be sufficient to fully distinguish between these genres. Additional features or more complex models might be needed to achieve better classification performance.

Conclusion

- The analysis of multiple audio features, including MFCCs, reveals that while individual features like mfcc4_mean and mfcc1_mean provide some degree of separability among music genres, they are not sufficient on their own to achieve perfect classification. However, when a combination of multiple features is considered, distinct patterns emerge, and the genres become increasingly separable.

- This suggests that each feature captures different aspects of the audio signal, and their combined use enhances the model's ability to differentiate between genres. The overlapping observed in certain genres when using single features diminishes significantly when multiple features are analyzed together, indicating that the complexity and diversity of musical characteristics are better captured through a multi-feature approach.

- In conclusion, the integration of various key audio features is essential for achieving accurate and robust music genre classification. By leveraging the unique information provided by each feature, the model can effectively distinguish between genres, minimizing overlap and leading to more precise classification outcomes. This holistic approach underscores the importance of comprehensive feature selection in developing high-performance machine learning models for music genre classification.

WORKING OF ALGORITHMS

Algorithms used in 'Music Genre Classification Using Machine Learning' are K-Nearest Neigbour(KNN), Support Vector Machine(SVM) and Naïve Baye's Classifier(NB).

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, yet powerful, non-parametric classification and regression algorithm. It classifies a data point based on how its neighbors are classified. The idea is that similar data points are likely to be close to each other, so by looking at the nearest neighbors of a point, you can predict its category (in classification) or value (in regression).

Working:

1. Choose the Number of Neighbors (K):

   Select the number of neighbors, K. This is the number of nearest neighbors the algorithm will look at when making a prediction. The value of K is a hyperparameter and must be chosen carefully.

2. Calculate the Distance:

   For the given test data point, calculate the distance between this point and all the other points in the training dataset. The most common distance metric used is Euclidean distance, but other metrics like Manhattan or Minkowski can also be used.

   Euclidean Distance Formula between two points (x1,y1) and (x2,y2) in a 2D space:

   $$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

   This can be generalized to n-dimensional space.

3. Sort the Distances:

   Sort all the distances calculated in the previous step in ascending order. This step helps in identifying the K nearest neighbors.

4. Select the Nearest Neighbors:

   From the sorted list of distances, select the top K nearest data points. These are the K neighbors closest to the test data point.

5. Make Predictions:

   For Classification:

   Assign the class that is most frequent among the K nearest neighbors (majority voting).

For Regression:

Compute the average of the values of the K nearest neighbors and assign this as the prediction.

6. Return the Predicted Value:

The class or value obtained from the previous step is the final prediction for the test data point.

Pseudocode:

Input:

- Training data (X_train, y_train)
- Test data point (X_test)
- Number of neighbors (K)

Output:

- Predicted class label or value for X_test

Steps:

1. For each point (X_test) in the test dataset:
    a. Calculate the distance between X_test and all points in the training dataset:
        - For i = 1 to len(X_train):
            - Compute distance d = distance(X_test, X_train[i])
            - Store the distance and the corresponding label of X_train[i]
    b. Sort the calculated distances in ascending order.
    c. Select the top K entries from the sorted list.
    d. Perform majority voting (for classification) or average (for regression):
    - For classification:
        - Count the number of occurrences of each class in the top K entries.
        - Assign the class with the highest count as the predicted class.
    - For regression:
        - Compute the average of the values corresponding to the top K entries.
        - Assign this average as the predicted value.
    e. Return the predicted class label or value.

End

Support Vector Machine (SVM)

Support Vector Machine is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

Working:

- Gather dataset which consists of feature vectors (inputs) and corresponding labels (outputs).
- Represent each data point as a vector in a multi-dimensional space, where each dimension corresponds to a feature of the data.
- Find the hyperplane that maximizes this margin, ensuring the best possible separation between the two classes.
- If the data is not linearly separable, the original feature space is mapped into a higher dimensional space by using kernel trick where a linear hyperplane can separate the classes.
- Find the weights and bias that define the hyperplane by maximizing the margin and minimizing the classification errors.
- Once trained, the SVM model can classify new data points. For each new point, the model checks on which side of the hyperplane the point falls.

Pseudocode:

1. Initialize Parameters:
    - Set the regularization parameter C.
    - Initialize weights w and bias b to 0 or small random values.
2. Training Process:
    - Repeat for a fixed number of iterations:
        a. For each data point (xi, yi) in the training set:

i. Calculate the decision function: f(xi) = w * xi + b

ii. If the data point is misclassified (i.e., yi * f(xi) < 1):

- Update weights: w = w + C * yi * xi

- Update bias: b = b + C * yi

3. Prediction: -

For a new data point x_new, compute:

f(x_new) = w * x_new + b

- Return the predicted class:

if f(x_new) >= 0, return +1

else return -1

Naive Bayes (NB)

Naive Bayes (NB) is a family of simple, yet effective probabilistic classifiers based on applying Bayes' Theorem with strong (naive) independence assumptions between the features. Despite its simplicity, Naive Bayes often performs surprisingly well in many real-world applications, particularly in text classification tasks like spam filtering, sentiment analysis, and document categorization.

Working:

1. Calculate Prior Probabilities:

   The prior probability P(Ci) for each class Ci is calculated from the training data. This is simply the proportion of instances belonging to each class in the training dataset.

$$P(C_i) = \frac{\text{Number of instances in } C_i}{\text{Total number of instances}}$$

2. Calculate Likelihoods:

   For each feature xj and class Ci, calculate the likelihood P(xj | Ci). The likelihood is the probability of feature xj given class Ci.

   Depending on the type of data (categorical or continuous), different distributions are assumed:

   i. For Categorical Data: Use frequency counts to estimate P(xj | Ci).

   ii. For Continuous Data: Assume a Gaussian (Normal) distribution and calculate the probability density function.

For a Gaussian distribution, the likelihood is calculated as:

$$P(x_j|C_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_j-\mu)^2}{2\sigma^2}\right)$$

where μ and σ^2 are the mean and variance of the feature x_j in class C_i.

3. Apply Bayes' Theorem:
    a. For a given test data point X, calculate the posterior probability for each class using Bayes' Theorem:

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)}$$

Since $P(X)$ is constant for all classes, it can be ignored for classification:

$$P(C_i|X) \propto P(X|C_i) \cdot P(C_i)$$

Here, $P(X|C_i)$ is the product of the likelihoods of each feature:

$$P(X|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i)$$

4. Make Predictions:
    a. Assign the class Ci to the test data point X that has the highest posterior probability:

$$\text{Predicted Class} = \arg\max_{C_i} P(C_i|X)$$

Pseudocode:

Input:

- Training data (X_train, y_train)
- Test data point (X_test)

Output:

- Predicted class label for X_test

Steps:

1. For each class C_i in the dataset:
    a. Compute the prior probability P(C_i) from the training data:

- P(C_i) = (Number of instances in C_i) / (Total number of instances)

b. For each feature x_j in X_test:

- Compute the likelihood P(x_j | C_i):

  - For categorical data, use frequency counts.

  - For continuous data, assume Gaussian distribution and calculate using the formula.

c. Compute the posterior probability P(C_i | X):

- P(C_i | X) ∝ P(X | C_i) * P(C_i)

- P(X | C_i) = Product of all likelihoods P(x_j | C_i) for each feature x_j.

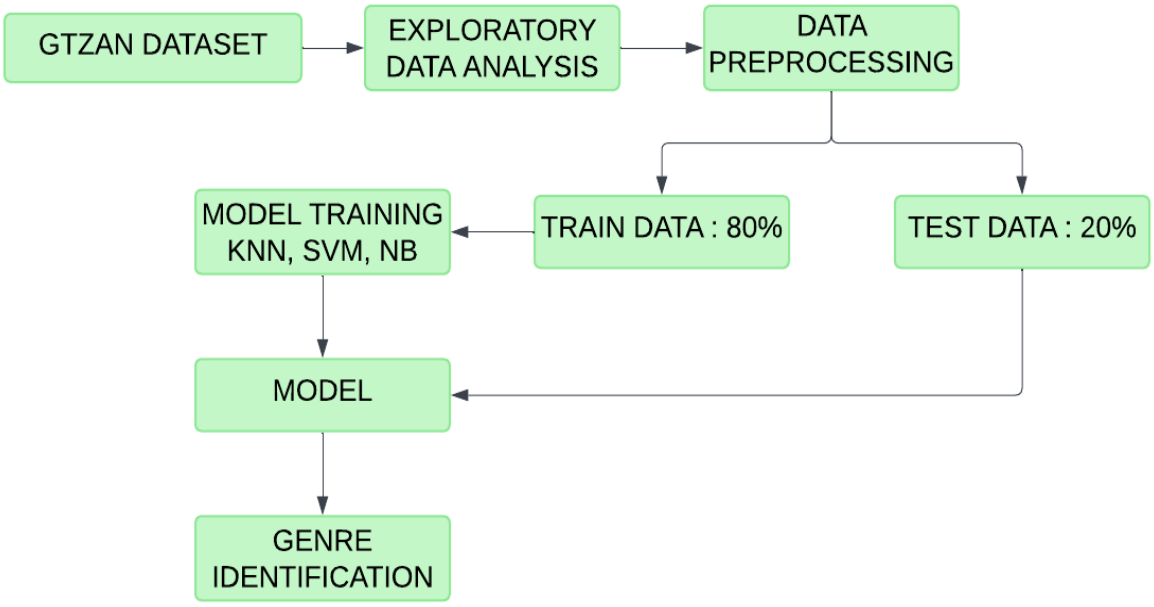
2. Assign the class label to X_test that has the highest posterior probability.

End

PACKAGES AND FUNCTIONS

- Pandas
    - read_csv()
    - head()
    - shape()
    - info()
    - drop()
    - value_counts()
    - isnull()
    - describe()
    - tail()
- Matplotlib
    - figure()
    - title()
    - show()
    - xlabel()
    - ylabel()
- Seaborn
    - boxplot()
    - countplot()
    - heatmap()
    - scatterplot()
- Numpy
- Librosa
    - librosa.feature.rms()
    - librosa.feature.chroma_stft()
    - librosa.feature.mfcc()
    - librosa.beat.tempo()

PIPELINE DIAGRAM



TIMELINE

➢ Submission of project synopsis with Journal Papers - 22.07.2024

➢ project proposal approval - 26.07.2024

➢ Presenting project proposal before the Approval Committee - 29.07.2024 & 30.07.2024

➢ Initial report submission - 12.08.2024

➢ Analysis and design report submission - 16.08.2024

➢ Verification of the report and PPT by the guide - 19.08.2024

➢ First project presentation - 21.08.2024 & 23.08.2024

➢ Sprint Release I - 30.08.2024 ➢ Sprint Release II - 26.09.2024

➢ Interim project presentation - 30.09.2024 & 01.10.2024

➢ Sprint Release III - 18.10.2024

➢ Project execution, submission of project report and PPT before the guide - 24.10.2024

➢ Submission of the project report to the guide - 28.10.2024

➢ Final project presentation - 28.10.2024 & 29.10.2024

➢ Submission of project report after corrections - 01.11.2024

SYSTEM DESIGN

Model Building

The primary aim of the model planning phase in the music genre classification system is to establish a comprehensive framework that guides subsequent model-building activities. This phase involves defining the scope, selecting appropriate algorithms, and outlining the overall strategy for accurately classifying music genres based on audio features, tailored to the unique characteristics of each audio file.

1. Objective:
   Build a Music Genre Classification System using machine learning techniques to classify music tracks based on their audio features. The goal is to accurately predict the genre of a given track by analyzing its pre-extracted features such as MFCCs, spectral centroid, chroma, etc.

2. Approach:
   Utilize a comparative study of three machine learning algorithms: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB). These models will be evaluated for accuracy, efficiency, and robustness in classifying music genres based on the extracted features from the GTZAN dataset.

3. Data Preparation:
   Import and preprocess the dataset, which consists of pre-extracted features from audio files. Handle missing values and normalize the features between 0 and 1. Ensure that the dataset is ready for training and testing by splitting it into training and test sets.

4. Exploratory Data Analysis (EDA):
   Perform EDA to understand the distribution of music genres, analyze feature distributions, and check for missing values. Visualize key audio features like MFCCs, spectral centroid, and chroma features to gain insights into the patterns between different genres.

5. Model Building:
   Implement the K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes (NB) algorithms. Train the models on the pre-extracted audio features and evaluate their performance using accuracy, confusion matrix, and classification report.

6. Model Comparison:
   Compare the performance of the three algorithms based on accuracy, speed, and generalization ability. Choose the best-performing model to classify music genres for new, unseen audio tracks.

Model Training

The dataset was divided into two parts. X representing the input features, and y representing the target variable. The training set consists of 80% of the data and is used to train the model.

KNN Model

```
#Training the KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

#Predicting on the test set
y_predict = knn.predict(X_test)
```

A K-Nearest Neighbors (KNN) classifier, which is a non-parametric, instance-based learning method, is used to train the model. The model considers the 5 nearest neighbors (k=5) for classification. The distance metric used is Euclidean distance, which calculates the straight-line distance between feature points. All features are equally weighted, and the majority class among the nearest neighbors is assigned to the input sample during prediction.

```
knn.score(X_train, y_train)
```

```
0.9294294294294294
```

The model achieved an accuracy of 92.94 % on the training data.

SVM Model

```
# Initialize the SVM model
svm_model = SVC(C=100, gamma='scale', kernel='rbf')

# Fit the model on training data
svm_model.fit(X_train, y_train)
```

A Support Vector Machine (SVM) model is used to train the classification model. The best hyperparameters were selected using the GridSearchCV method, which systematically tested various combinations. The model uses the Radial Basis Function (RBF) kernel, which is effective for non-linear classification tasks. The C=100 parameter controls the regularization strength, balancing correct classification of training examples with maximizing the margin between classes. The gamma='scale' parameter defines the influence of individual training examples, adjusted automatically based on the feature dimensions to ensure an appropriate balance of model complexity.

```
svm_model.score(X_train, y_train)
```

```
0.9625875875875876
```

The model achieved an accuracy of 96.25 % on the training data.

Naive Baye's Model

```
# Bagging with Naive Bayes
nb_model = GaussianNB()
bagging_model = BaggingClassifier(n_estimators=50, random_state=42)
bagging_model.fit(X_train, y_train)
```

A Naive Bayes (NB) model is used to train the classification model. Specifically, the Gaussian Naive Bayes (GaussianNB) algorithm is employed, which assumes that the features follow a normal distribution. To improve the model's accuracy and reduce variance, the Bagging method is applied, where 50 base models are trained on different random subsets of the training data. By aggregating the predictions from these models, the overall classification performance is enhanced, leading to better generalization and robustness. The Bagging Classifier is initialized with 50 estimators and a random state of 42 to ensure reproducibility of results.

```
bagging_model.score(X_train, y_train)
```

```
0.9991241241241241
```

The model achieved an accuracy of 99.91 % on the training data.

Model Testing

KNN model

```
#Predicting on the test set
y_predict = knn.predict(X_test)

#Evaluating the model
accuracy = accuracy_score(y_test, y_predict)
classification_rep = classification_report(y_test, y_predict)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 89.49%
Classification Report:
              precision    recall  f1-score   support

       blues       0.91      0.91      0.91       208
   classical       0.86      0.97      0.91       203
     country       0.82      0.90      0.86       186
       disco       0.85      0.91      0.88       199
      hiphop       0.93      0.84      0.89       218
        jazz       0.91      0.85      0.88       192
       metal       0.94      0.93      0.93       204
         pop       0.94      0.90      0.92       180
      reggae       0.93      0.91      0.92       211
        rock       0.88      0.82      0.85       197

    accuracy                           0.89      1998
   macro avg       0.90      0.89      0.89      1998
weighted avg       0.90      0.89      0.89      1998
```

KNN obtained an accuracy of 89.49 % on test data.

SVM Model

```
#Predicting on the test set
y_pred = svm_model.predict(X_test)

# Step 4: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

# Step 5: Display the evaluation results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(confusion_mat)
```

```
Accuracy: 88.79%
Classification Report:
              precision    recall  f1-score   support

       blues       0.90      0.92      0.91       208
   classical       0.93      0.96      0.94       203
     country       0.81      0.89      0.85       186
       disco       0.86      0.85      0.85       199
      hiphop       0.91      0.85      0.88       218
        jazz       0.89      0.91      0.90       192
       metal       0.92      0.95      0.94       204
         pop       0.93      0.93      0.93       180
      reggae       0.89      0.86      0.88       211
        rock       0.83      0.76      0.80       197

    accuracy                           0.89      1998
   macro avg       0.89      0.89      0.89      1998
weighted avg       0.89      0.89      0.89      1998
```

SVM obtained an accuracy of 88.79 % on test data.

Naïve Baye's Model

```
#Predicting on the test set
y_pred = bagging_model.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

#Display the evaluation results
print(f"Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 84.53%
Classification Report:
              precision    recall  f1-score   support

       blues       0.82      0.84      0.83       208
   classical       0.94      0.95      0.94       203
     country       0.73      0.80      0.76       186
       disco       0.83      0.78      0.80       199
      hiphop       0.88      0.83      0.86       218
        jazz       0.82      0.88      0.85       192
       metal       0.86      0.91      0.88       204
         pop       0.91      0.91      0.91       180
      reggae       0.85      0.85      0.85       211
        rock       0.81      0.71      0.76       197

    accuracy                           0.85      1998
   macro avg       0.85      0.85      0.84      1998
weighted avg       0.85      0.85      0.84      1998
```

Naïve Baye's obtained an accuracy of 84.53 % on test data.

Testing with Unseen Data

```
#Load the audio file
audio_path = '/content/drive/MyDrive/Music_Genre_Classification/country.00063.wav'
y, sr = librosa.load(audio_path)

#trim leading and trailing silence
y, _ = librosa.effects.trim(y)


segment_length = 3
samples_per_segment = int(segment_length * sr)

features_list = []

# Loop through the audio file and extract features from each 3-second segment
for start in range(0, len(y), samples_per_segment):
    end = start + samples_per_segment
    if end > len(y):
        break

    y_segment = y[start:end]

    # Extract features from the current segment
    features_dict = {
        'chroma_stft_mean': np.mean(librosa.feature.chroma_stft(y=y_segment, sr=sr)),
```

```
    'chroma_stft_var': np.var(librosa.feature.chroma_stft(y=y_segment, sr=sr)),
    'rms_mean': np.mean(librosa.feature.rms(y=y_segment)),
    'rms_var': np.var(librosa.feature.rms(y=y_segment)),
    'spectral_centroid_mean': np.mean(librosa.feature.spectral_centroid(y=y_segment, sr=sr)),
    'spectral_centroid_var': np.var(librosa.feature.spectral_centroid(y=y_segment, sr=sr)),
    'spectral_bandwidth_mean': np.mean(librosa.feature.spectral_bandwidth(y=y_segment, sr=sr)),
    'spectral_bandwidth_var': np.var(librosa.feature.spectral_bandwidth(y=y_segment, sr=sr)),
    'rolloff_mean': np.mean(librosa.feature.spectral_rolloff(y=y_segment, sr=sr)),
    'rolloff_var': np.var(librosa.feature.spectral_rolloff(y=y_segment, sr=sr)),
    'zero_crossing_rate_mean': np.mean(librosa.feature.zero_crossing_rate(y=y_segment)),
    'zero_crossing_rate_var': np.var(librosa.feature.zero_crossing_rate(y=y_segment)),
    'harmony_mean': np.mean(librosa.effects.harmonic(y_segment)),
    'harmony_var': np.var(librosa.effects.harmonic(y_segment)),
    'perceptr_mean': np.mean(librosa.feature.spectral_flatness(y=y_segment)),
    'perceptr_var': np.var(librosa.feature.spectral_flatness(y=y_segment)),
    'tempo': librosa.beat.tempo(y=y_segment, sr=sr)[0]
}

# Extract MFCC features (10 coefficients)
mfccs = librosa.feature.mfcc(y=y_segment, sr=sr, n_mfcc=10)
for i in range(10):
    features_dict[f'mfcc{i+1}_mean'] = np.mean(mfccs[i])
    features_dict[f'mfcc{i+1}_var'] = np.var(mfccs[i])
```

```
normalized_new_data = normalize_new_data(mean_df, X_min, X_max)
mean_df = pd.DataFrame(normalized_new_data, columns=mean_df.columns)

normalized_new_data = normalize_new_data(max_df, X_min, X_max)
max_df = pd.DataFrame(normalized_new_data, columns=max_df.columns)

normalized_new_data = normalize_new_data(min_df, X_min, X_max)
min_df = pd.DataFrame(normalized_new_data, columns=min_df.columns)


genre = model.predict(min_df)
print(f"The predicted genre in min is: {genre[0]}")

genre = model.predict(mean_df)
print(f"The predicted genre in mean is: {genre[0]}")

genre = model.predict(max_df)
print(f"The predicted genre in max is: {genre[0]}")
```

```
The predicted genre in min is: country
The predicted genre in mean is: country
The predicted genre in max is: reggae
```