HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

CAPSTONE PROJECT REPORT:

Music Genre Classification

Supervised by:

Associate Professor Khoat Than Quang

Group members:

Quach Tuan Anh - 20225469

Pham Tran Tuan Khang - 20225503

Dinh Van Kien – 20225505

Vu Ngoc Ha – 20225490

Nguyen Tran Nghia - 20225452

Hanoi - Vietnam

2024

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Professor Khoat Than Quang for his invaluable guidance and unwavering support throughout the duration of this Music Genre Classification project. His extensive knowledge and expertise were instrumental in navigating the complexities of this research. Professor Quang's encouragement and insightful feedback consistently motivated us to strive for excellence and overcome the challenges we encountered. We are deeply appreciative of his commitment and contributions, which were essential to the successful completion of this project.

ABSTRACT

This report outlines the development of a machine learning model designed to classify songs into specific genres. The process encompasses data preprocessing, feature extraction, feature engineering, model training, evaluation, and fine-tuning, employing a stacking ensemble method to enhance accuracy by combining multiple models. We began with the GTZAN dataset (1000 instances) and expanded it by gathering additional data from Spotify, resulting in a more diverse dataset of 30,000 instances. This larger dataset addressed the limitations of the initial small dataset, enhancing the model's performance across various genres. To ensure broad accessibility, we deployed the model in several environments. We created a Docker image to encapsulate all dependencies and configurations, ensuring consistent performance across different environments. The Dockerized application was then deployed using Fly.io, making it accessible via the internet with an improved user interface compared to the local application. The model serves multiple purposes. For personal use, it can organize music collections by genre, assist in creating genre-specific playlists, and enhance the overall listening experience. Commercially, music streaming services can integrate the model to improve their recommendation algorithms, enable genre-based search and discovery, and personalize user experiences. The primary users include music enthusiasts, developers integrating genre classification into music applications, and researchers studying music genres. This project demonstrates the potential of machine learning to automate and improve music genre classification, offering significant benefits to both individual users and commercial services.

Keywords: *music genre classification, machine learning, feature engineering, ensemble method, data preprocessing, Docker, Fly.io, recommendation systems, deep learning.*

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

The rapid expansion of digital music libraries has necessitated the development of sophisticated methods for organizing and categorizing music. One particularly challenging task is the classification of music into genres, a problem that combines elements of both signal processing and machine learning. Effective genre classification can significantly enhance user experience, enabling more intuitive navigation, personalized recommendations, and improved search functionality. This project, titled "Music Genre Classification," focuses on leveraging machine learning to automate the categorization of music tracks into predefined genres. Our approach integrates advanced data processing techniques, feature extraction, and model training to create a robust and accurate classification system.

Initially, our data collection efforts began with the GTZAN dataset, which includes 1,000 instances of various music genres. Recognizing the limitations of this relatively small dataset, we expanded it by scraping additional data from Spotify. This resulted in a comprehensive dataset of 30,000 instances, ensuring a more diverse and representative training set. We performed extensive feature extraction and engineering, focusing on characteristics such as tempo, rhythm, and spectral properties to capture the nuances of different music genres. Various machine learning models were trained and evaluated, including support vector machines, random forests, and neural networks. To improve accuracy, we implemented an ensemble method known as stacking, which combines multiple models into a single, more powerful classifier.

We created a Docker image to encapsulate the application and its dependencies, facilitating consistent deployment across different environments. The Dockerized application was deployed on Fly.io, making the tool accessible via the internet with an enhanced user interface. The significance of our project extends to both personal and commercial domains. For individual users, our model can automate the organization of music collections, assist in creating genre-specific playlists, and

enrich the overall listening experience. In the commercial sphere, music streaming services can integrate our model to refine their recommendation algorithms, enabling more accurate genre-based searches and personalized user interactions.

The primary users of our music genre classification model include music enthusiasts, developers, and researchers. Music enthusiasts can manage and explore their music collections more effectively, while developers can integrate genre classification capabilities into music applications. Researchers can use advanced tools to support their studies of music genres. By demonstrating the practical application of machine learning in music genre classification, this project highlights the potential for technological advancements to transform the way we interact with and enjoy music.

CHAPTER 2. DATA PREPARATION

2.1 Data Collection

The objective of this data collection effort was to gather music features from various genres available on Spotify for further analysis of musical attributes and their correlation with different genres. Implemented using a Jupyter Notebook, the process involved four primary steps: setting up the environment by configuring Spotify API credentials for authentication and data fetching; defining the playlist and genre by manually inputting the Spotify playlist URL; scraping the music data; and extracting music features, including chroma, spectral contrast, and Mel-frequency cepstral coefficients (MFCCs) and other relevant attributes. This systematic approach facilitates the automated scraping and processing of music data from Spotify playlists, providing a robust foundation for in-depth analysis of musical characteristics across different genres.

2.2 Data Preprocessing

The preprocessing pipeline of the GTZAN dataset involves several key steps to prepare the data for machine learning analysis. First, it imports necessary libraries and loads the dataset, followed by separating the features and target variables. Categorical labels (e.g., blues, jazz) are converted into numerical labels (0-9) using label encoding, facilitating training and prediction. The features are then normalized using a 'MinMaxScaler' to scale the data to the range [0,1], ensuring uniform scales across features and preventing any single feature from dominating the model due to its larger range. The dataset is split into training and test sets, with an 80-20 ratio, using stratified sampling to maintain the same class distribution in both sets, which is crucial in handling class imbalances common in music classification. This preprocessing ensures that the data is formatted suitably for machine learning algorithms, with normalized features and numerical labels, and that the split enables robust evaluation of model performance on unseen data.

2.3 Data Exploration

The dataset comprises various audio features extracted from 30-second audio clips. The objective of this exploration is to understand the dataset's structure, quality, and relationships, and to identify patterns that could be leveraged in the classification model.

The dataset consists of 32,570 entries and 58 columns, each representing various audio features extracted from music tracks. The columns include both statistical measures (mean, variance) of audio features like chroma, spectral centroid, rolloff, and Mel-frequency cepstral coefficients (MFCCs), etc. as well as a label indicating the genre of each track.

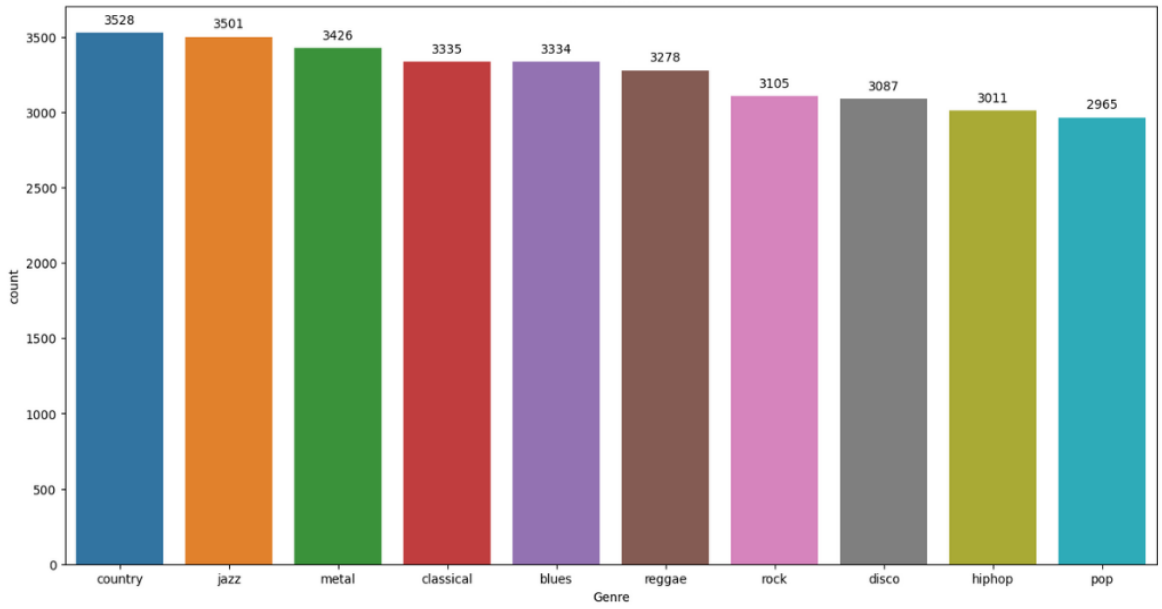

Figure 1. Genre Distribution.

Descriptive statistics reveal the central tendency and dispersion of the audio features:

The Chroma STFT Mean and Variance features: These features measure the intensity of the 12 pitch classes in the musical octave for each audio clip using the Short-Time Fourier Transform (STFT). The mean and variance give statistical summaries of the harmonic content throughout the clip, which is important for

recognizing the harmonic structure and distinguishing between genres with unique harmonic features.

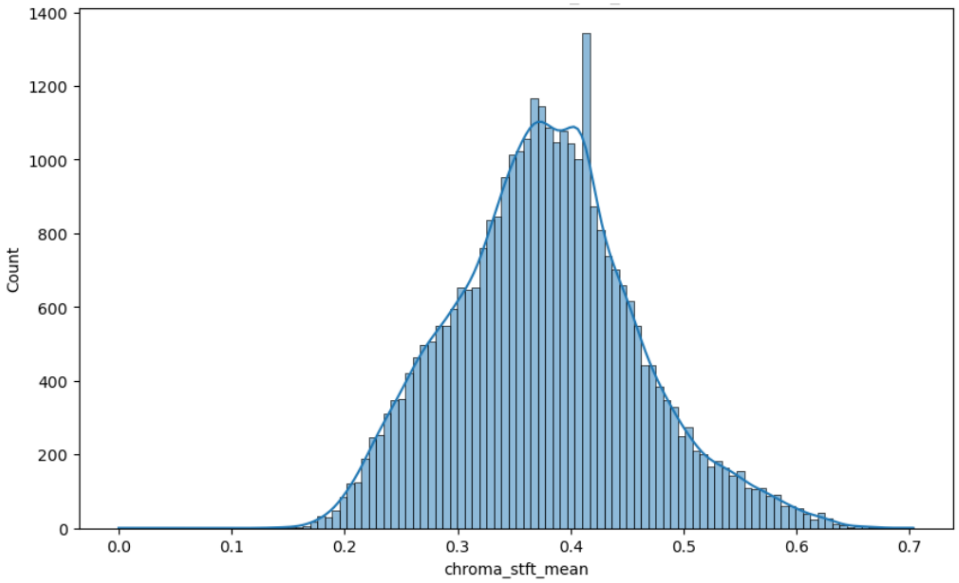

Figure 2. Distribution of chroma_stft_mean.

The histogram of the chroma STFT mean values reveals a clustered distribution with a few distinct peaks. This suggests that there are common pitch class intensities that occur frequently across the dataset. The clustering indicates that certain pitch classes are prevalent in the majority of audio samples, reflecting a pattern in the musical content that emphasizes specific pitches more than others.

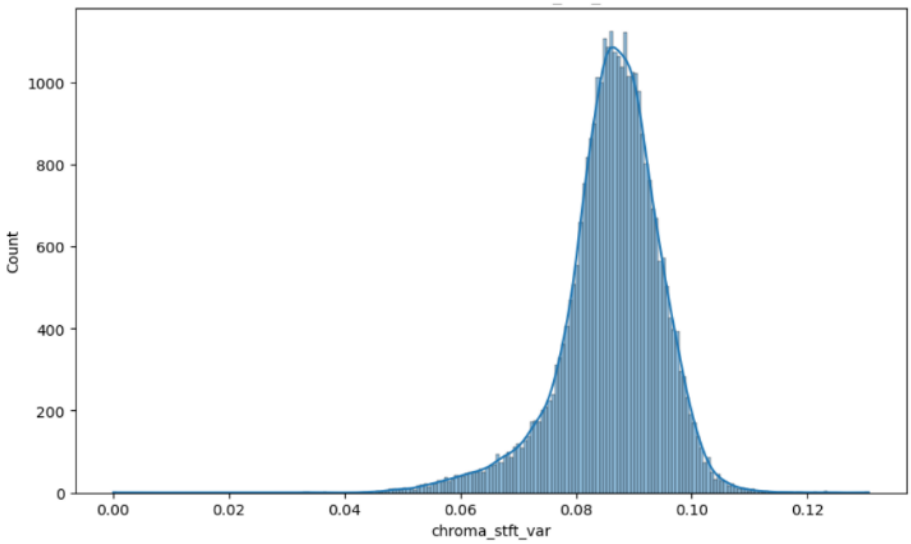

Figure 3. Distribution of chroma_stft_var

The chroma STFT variance histogram shows a wide spread with values tapering off towards higher variances. This indicates significant variability in the intensity of pitch classes within the dataset. While many audio samples maintain consistent pitch intensities, a notable portion exhibits dynamic changes, suggesting a mix of stable and fluctuating pitch patterns across the dataset.

MS Mean and Variance: The audio signal's loudness is measured by the Root Mean Square (RMS) energy. The mean value indicates the overall loudness, while the variance shows how the loudness changes within the clip. These characteristics are crucial for examining the dynamic range and energy distribution of music, which can differ greatly across various music genres.

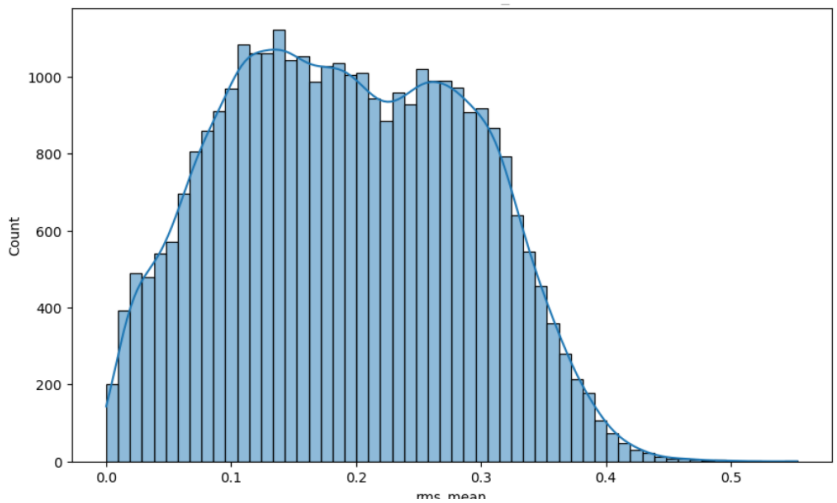

Figure 4. Distribution of rms_mean.

The distribution of RMS mean values is skewed towards lower values, indicating that most audio samples have relatively low average loudness. This skewness reflects a dataset where quieter tracks are more common, with fewer samples exhibiting high average loudness. This could be indicative of the genres or recording conditions represented in the dataset.

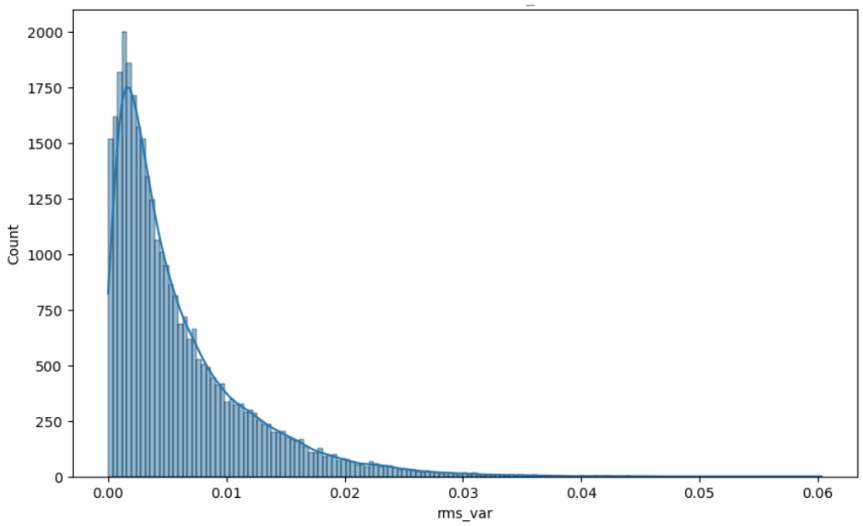

Figure 5. Distribution of rms_var.

The histogram for RMS variance displays a distribution with a long tail towards higher values, suggesting that while many audio samples have stable loudness, there are a few with significant fluctuations. This indicates that the dataset includes both consistently loud tracks and those with varying loudness, which may correspond to dynamic musical compositions or varying recording quality.

Spectral Centroid Mean and Variance: The spectral centroid serves as the "center of mass" of the audio spectrum and is commonly associated with the sound's brightness. Analyzing the mean and variance of the spectral centroid can offer valuable information about the timbre of the audio, aiding in the differentiation of genres based on their spectral profiles.

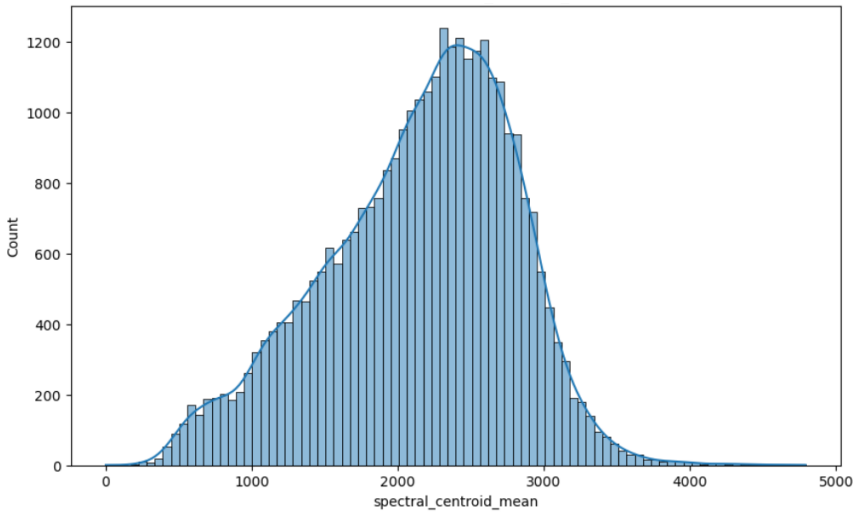

Figure 6. Distribution of spectral_centroid_mean

The spectral centroid mean values form a right-skewed distribution, with most values clustered at the lower end. This indicates that the majority of audio samples are characterized by darker sounds, with more low-frequency components dominating. Fewer samples have higher spectral centroid means, which would be associated with brighter, high-frequency sounds.

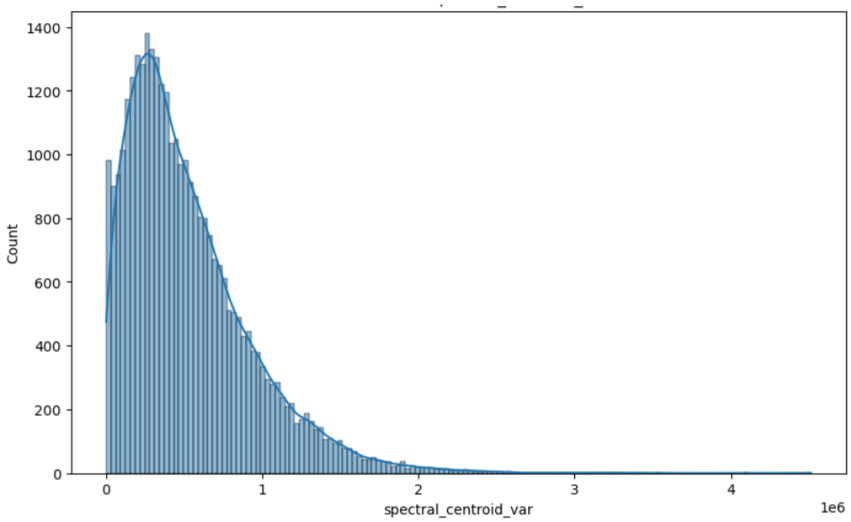

Figure 7. Distribution of spectral_centroid_var.

The histogram of spectral centroid variance shows a wide range of values, indicating diverse changes in the brightness of the sound. While some audio samples maintain a consistent spectral centroid, others experience significant fluctuations. This suggests a mix of stable and dynamically varying brightness within the dataset, reflecting different musical styles and structures.

Spectral Bandwidth Mean and* Mean and Variance*: Spectral bandwidth measures the width of the frequency band in the audio signal, reflecting its complexity and richness. The mean and variance of the spectral bandwidth summarize the overall texture and richness of the music, which can vary widely between genres and are key for genre classification.

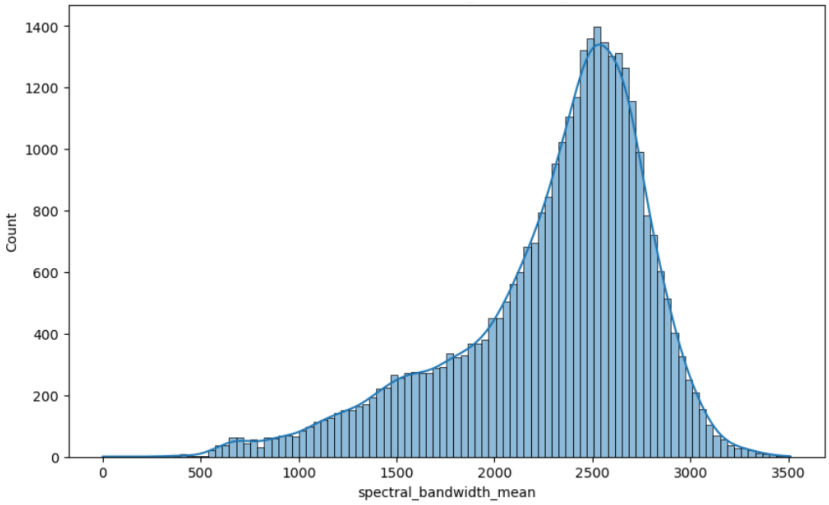

Figure 8. Distribution of spectral_bandwidth_mean.

The spectral bandwidth mean histogram peaks at lower values but has a spread towards higher values. This indicates that most audio samples have a limited range of frequencies, but there are also samples with a broader frequency range. This variability suggests that the dataset contains both narrow and wide spectral content, reflecting diverse musical elements.

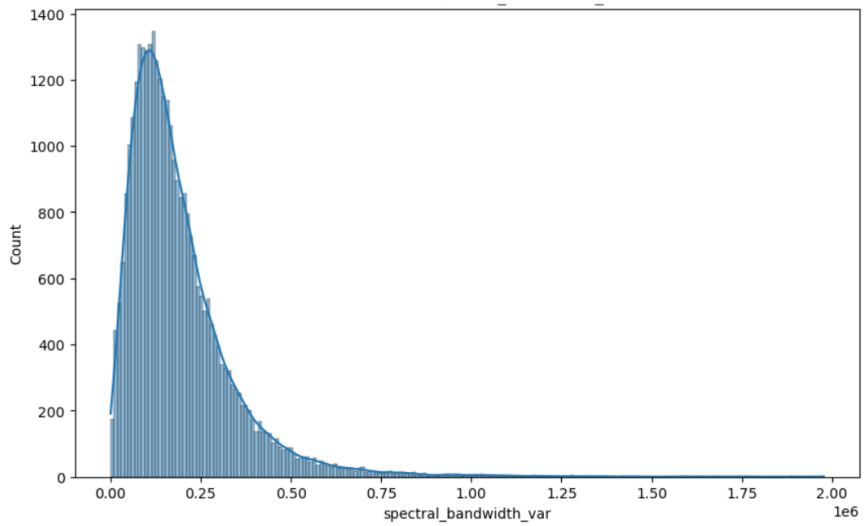

Figure 9. Distribution of spectral_bandwidth_var.

The spectral bandwidth variance histogram is right-skewed, with a few samples showing high variance. This indicates that while many audio samples have consistent frequency ranges, some experience significant changes in their spectral bandwidth

over time. This variability points to the dynamic nature of some audio tracks within the dataset.

***Rolloff Mean and Variance**:* is the frequency below which a specified percentage of the total spectral energy lies, typically 85%. The mean and variance of the spectral rolloff help differentiate between harmonic and percussive sounds, aiding in the classification of genres with different rhythmic and harmonic characteristics.

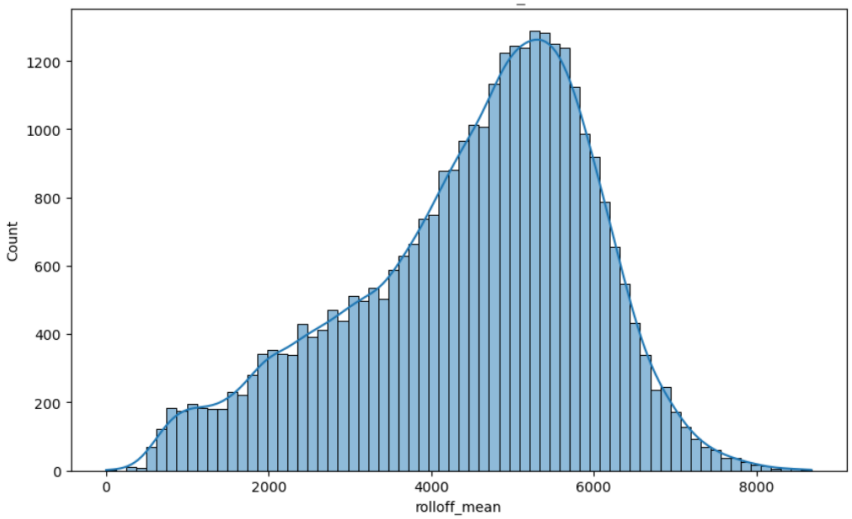

Figure 10. Distribution of rolloff_mean.

The rolloff mean values are concentrated towards the lower end of the distribution, as shown by the histogram. This suggests that most audio samples have lower spectral rolloff points, indicating less high-frequency content. This could be characteristic of certain musical genres that emphasize lower frequencies.

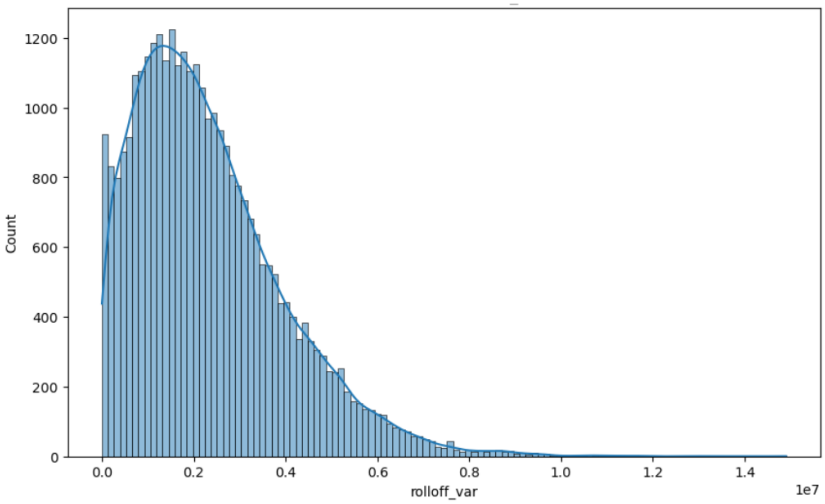

Figure 11. Distribution of rolloff_var.

The distribution of rolloff variance values is broad, indicating that the extent to which spectral energy distribution varies over time differs across the dataset. Some audio samples maintain consistent spectral energy distribution, while others show significant variability, reflecting diverse audio dynamics and production techniques.

Zero-Crossing Rate Mean and Variance: The zero-crossing rate (ZCR) measures how often the audio signal changes sign. The mean value represents the overall rate of sign changes, and the variance indicates its variability. These features are useful for identifying noisy or percussive sounds, which are more common in some genres than others.

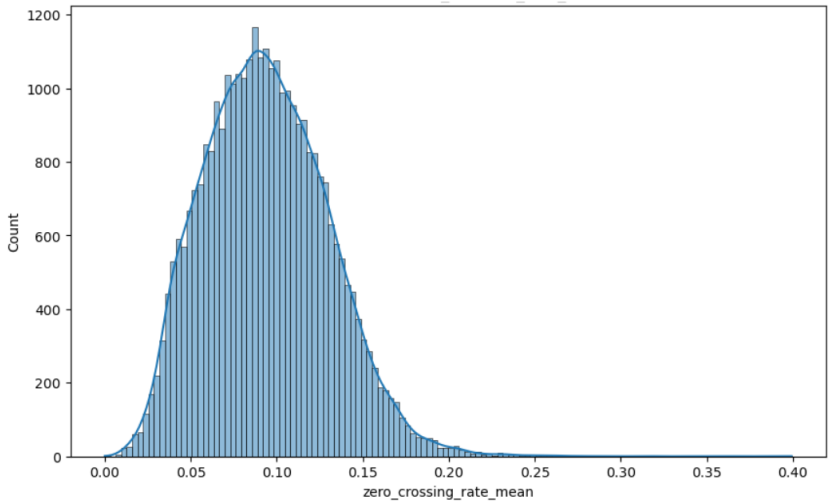

Figure 12. Distribution of zero_crossing_rate_mean.

The histogram for zero-crossing rate mean values is skewed towards lower values, indicating that most audio samples have lower zero-crossing rates, typical of harmonic sounds rather than percussive sounds. This suggests that the dataset predominantly consists of tracks with harmonic content, which could be indicative of the genres represented.

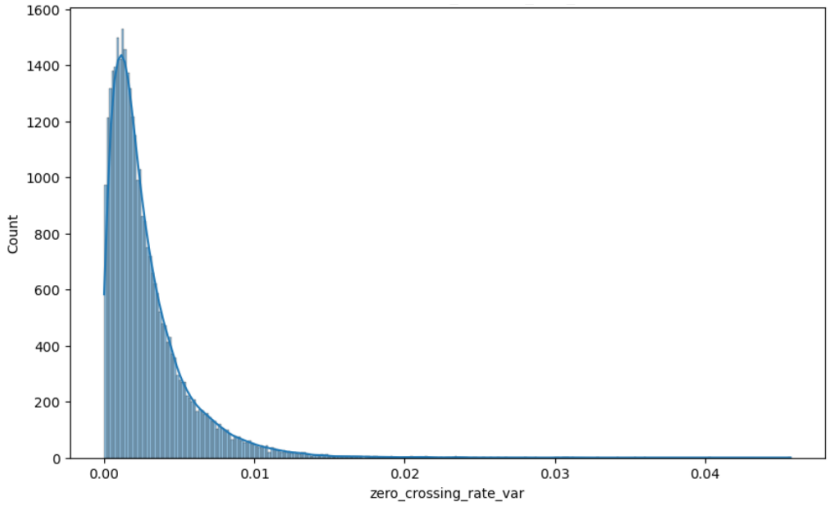

Figure 13. Distribution of zero_crossing_rate_var.

The zero-crossing rate variance histogram shows a broad range of values, suggesting variability in the zero-crossing rates across the dataset. While some audio samples maintain a consistent zero-crossing rate, others exhibit significant changes, indicating a mix of stable and dynamically varying audio signals.

Harmony Mean and Variance: Harmony features measure the harmonic content of the audio signal, providing a summary of the degree of harmonic structure present in the clip. The mean and variance of these features are important for distinguishing between genres with different harmonic properties.

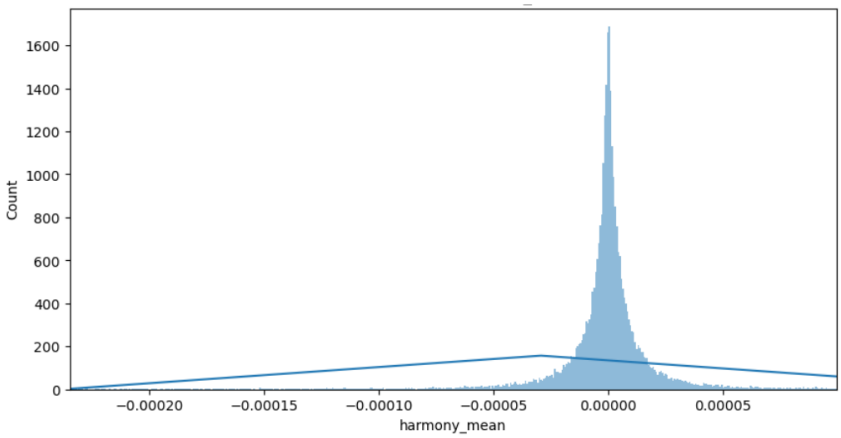

Figure 14. Distribution of harmony_mean.

The histogram for harmony mean values resembles a normal distribution around a central value, indicating that harmony levels are relatively consistent across the dataset. This suggests that most audio samples have similar harmonic content, which may reflect common harmonic structures in the music genres included in the dataset.

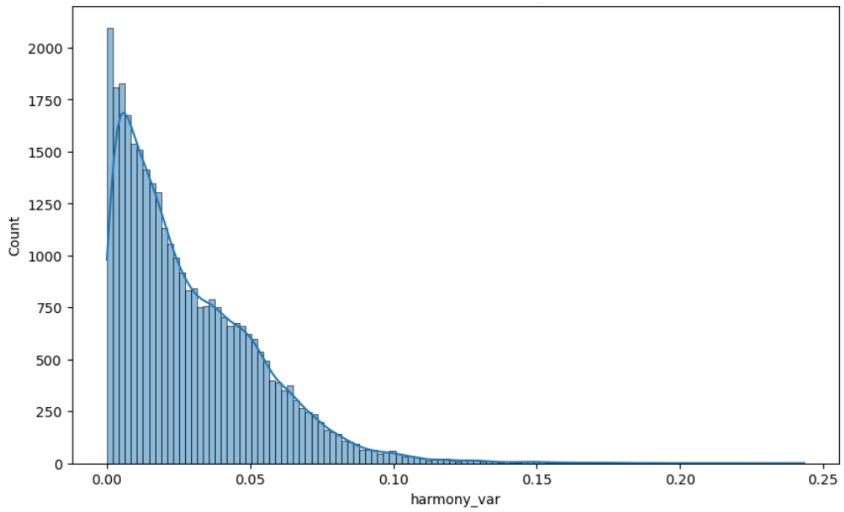

Figure 15. Distribution of harmony_var.

The distribution of harmony variance values is wide, indicating significant variability in harmonic content across the dataset. While some audio samples have stable harmonic content, others experience substantial changes, suggesting diverse harmonic structures and musical elements within the dataset.

Perceptual Mean and Variance: These features capture perceptual characteristics of the audio, such as loudness, sharpness, or roughness, which reflect how the audio is perceived by the human ear. The mean and variance summarize these perceptual metrics, aligning closely with human auditory perception and aiding in genre classification.

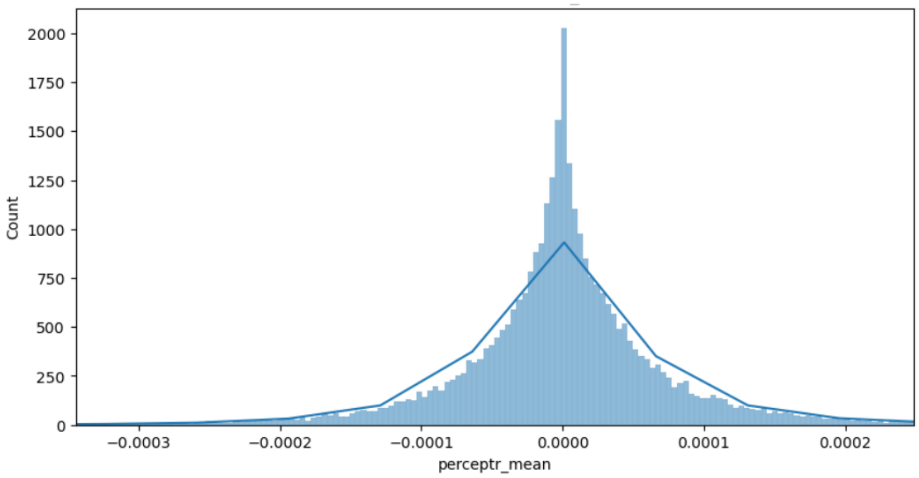

Figure 16. Distribution of perceptr_mean.

The perceptual mean values form a distribution with a central peak, indicating that perceptual features are fairly uniform across the dataset. This uniformity suggests that the audio samples share common timbral qualities, reflecting similar perceptual characteristics in the music.

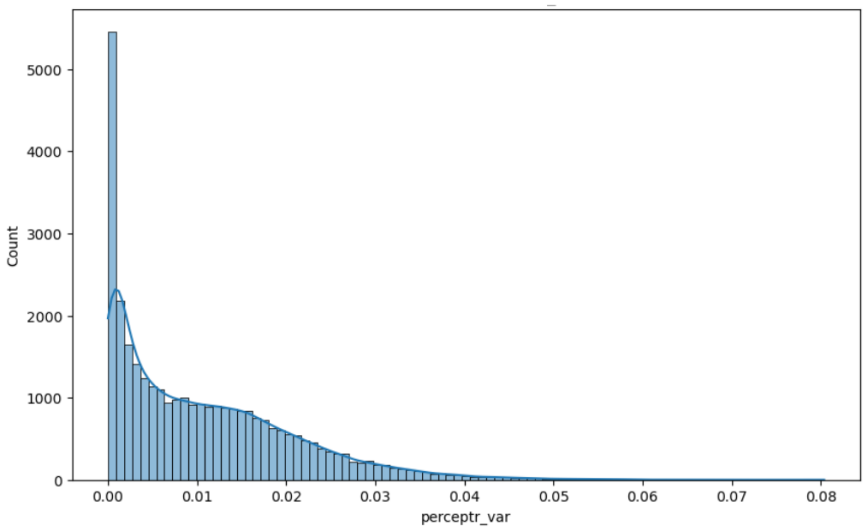

Figure 17. Distribution of perceptr_var.

The histogram for perceptual variance shows a wide range of values, indicating that perceptual qualities vary significantly across the dataset. Some audio samples have stable perceptual features, while others exhibit significant changes, reflecting diverse timbral variations within the dataset.

Tempo: Tempo measures the speed or pace of the music in beats per minute (BPM). It is a critical feature for distinguishing between genres with different rhythmic patterns and speeds, providing insights into the overall pace of the music.

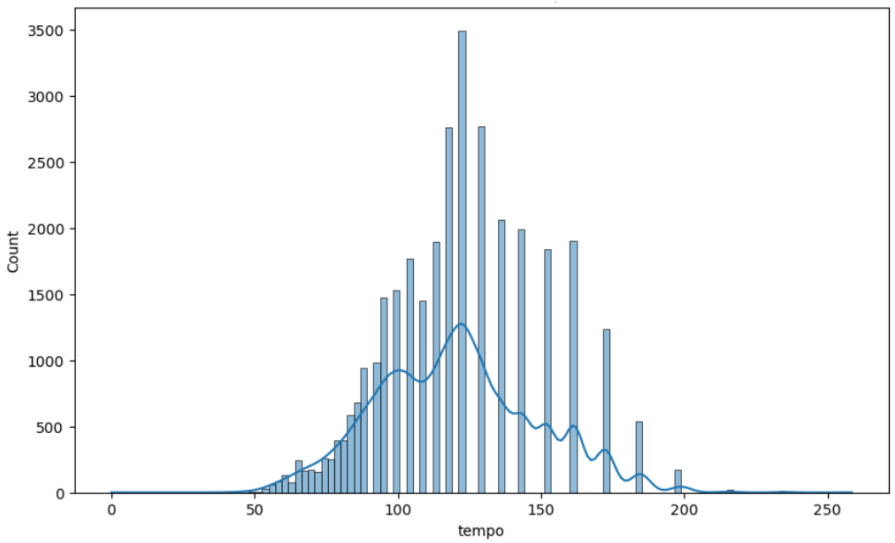

Figure 18. Distribution of tempo.

The tempo histogram reveals a concentration around certain values, with a few outliers. This suggests that most audio samples have tempos within a common range, reflecting popular tempos in music. However, the presence of outliers indicates that there are also samples with distinctly different tempos, adding to the diversity of the dataset.

Mel-Frequency Cepstral Coefficients (MFCCs): MFCCs are a representation of the short-term power spectrum of sound, widely used in audio and speech processing. The mean of an MFCC provides a measure of the central tendency of that particular coefficient, reflecting the average spectral shape of the audio signal. The variance of an MFCC indicates the spread or variability of that particular coefficient, capturing the dynamic range and fluctuations in the spectral shape of the audio signal.

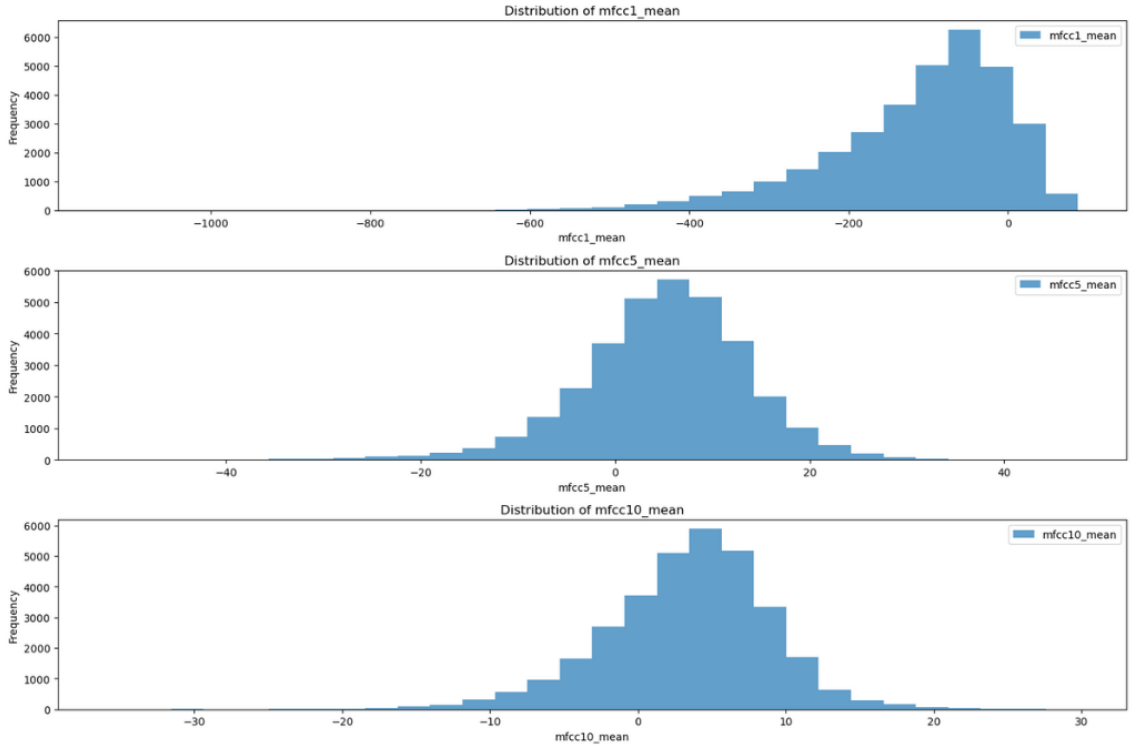

Figure 19. Distribution of mfcc1_mean, mfcc5_mean, mfcc10_mean.

For the MFCC means, the histograms indicate that most features exhibit a roughly normal distribution, suggesting that the average spectral characteristics of the audio signals are fairly consistent within the dataset. However, there are notable variations in the range and central tendency among different MFCCs, with some showing a wider spread and others more tightly clustered around their mean values. This variation reflects the diverse spectral content across different audio samples. Additionally, certain MFCC mean features display skewness, indicating the presence of outliers that deviate from the typical spectral pattern.

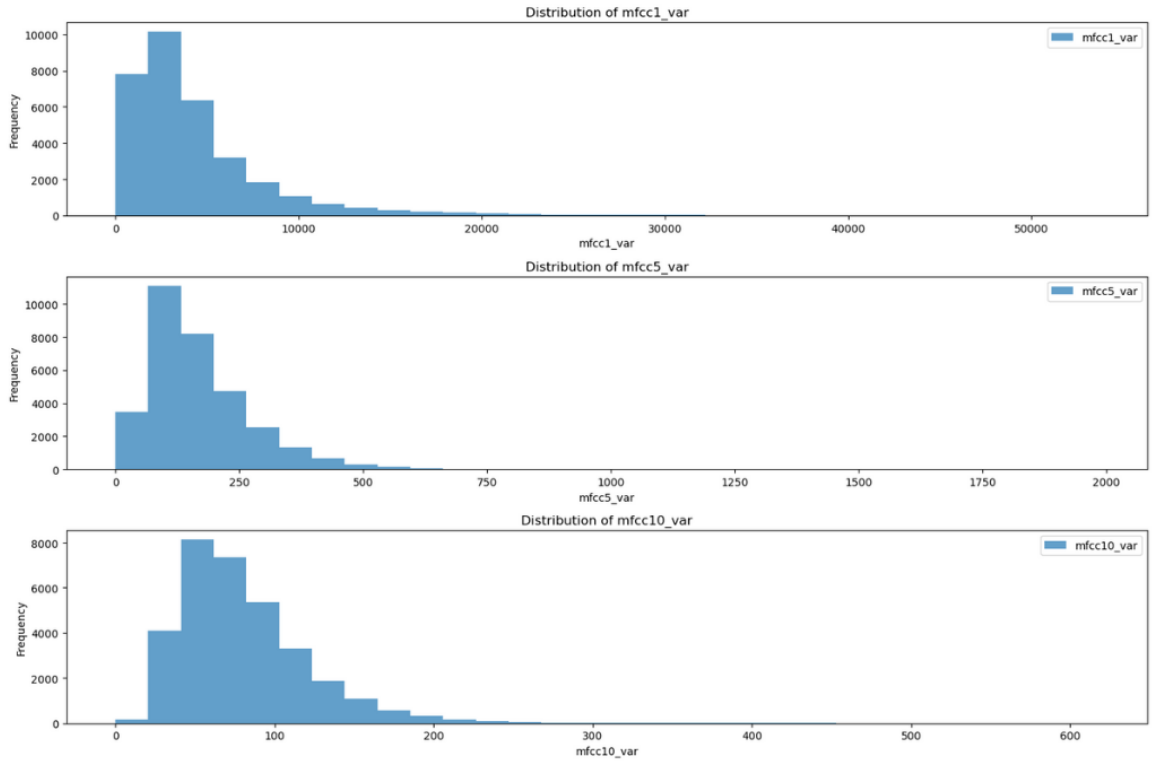

Figure 20. Distribution of mfcc1_var, mfcc5_var, mfcc10_var.

The histograms for the MFCC variances illustrate a broader range of values compared to the means, highlighting the dynamic nature of the audio signals. The distributions of the variance features tend to be more dispersed, indicating greater variability in the spectral content. Some MFCC variances show a right skew, suggesting that while many audio samples have moderate variability, there are instances with significantly higher fluctuations in specific frequency bands. This dynamic range captured by the variances is crucial for distinguishing between genres with varying spectral complexities.

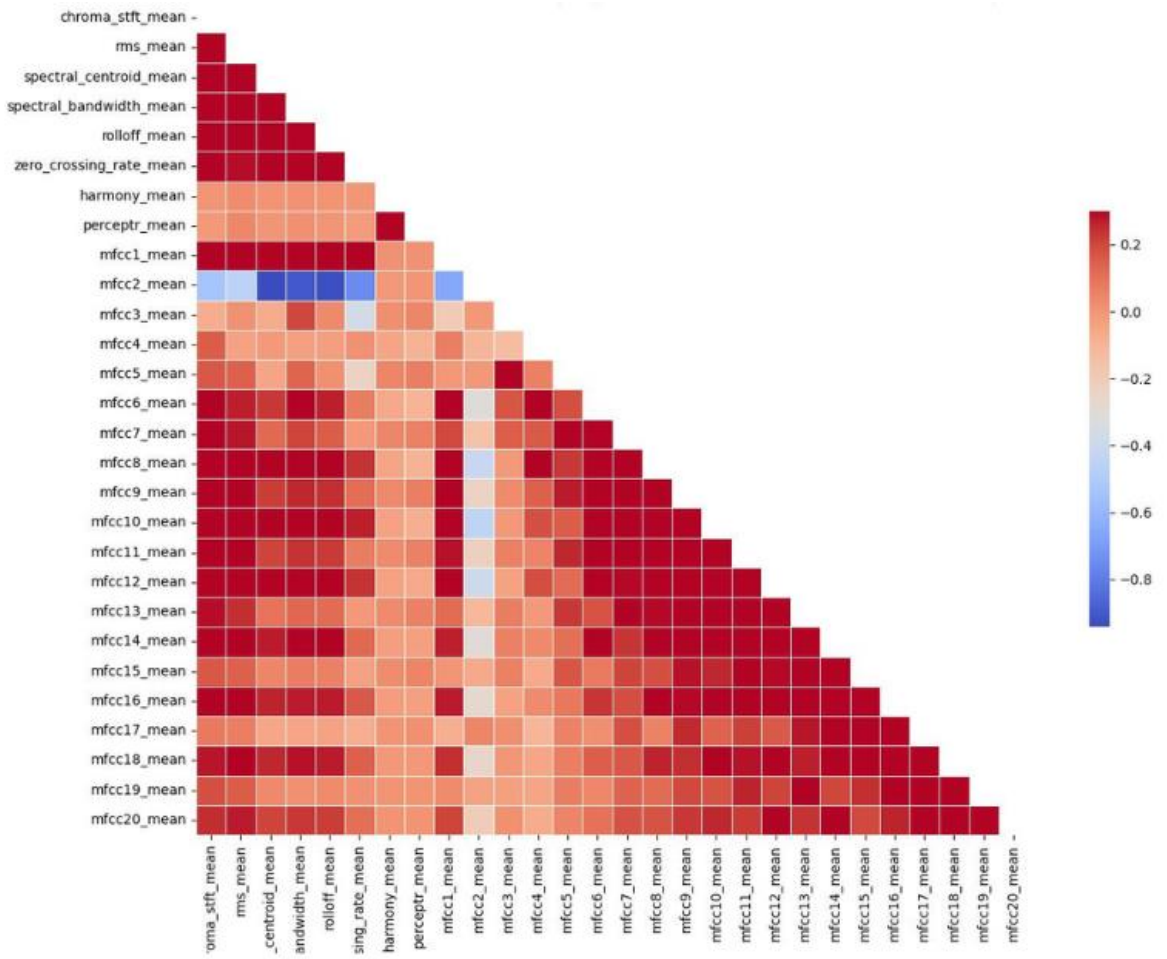

Figure 21. Correlation Heatmap (for the MEAN variables).

The correlation analysis of the mean values in the dataset reveals significant relationships among various audio features. The `chroma_stft_mean` demonstrates moderate positive correlations with `rms_mean` and `spectral_centroid_mean`, suggesting that higher chroma values are associated with higher energy and brighter sounds. It also correlates with `spectral_bandwidth_mean`, indicating a broader range of frequencies. Spectral features like `spectral_centroid_mean` have strong positive correlations with `spectral_bandwidth_mean`, `rolloff_mean`, and `zero_crossing_rate_mean`, collectively describing the frequency content and brightness of the audio signal. For the MFCC features, `mfcc1_mean` is positively correlated with `mfcc2_mean` and `mfcc3_mean`, highlighting their joint variation in capturing broad spectral shapes. However, `mfcc2_mean` negatively correlates with higher-order MFCCs such as `mfcc4_mean`, `mfcc5_mean`, and `mfcc6_mean`,

indicating complex interdependencies. Higher-order MFCCs, including `mfcc13_mean`, `mfcc14_mean`, and `mfcc15_mean`, show moderate to strong positive correlations with each other, reflecting their detailed capture of spectral characteristics. Overall, the spectral feature means are highly inter-correlated, providing a comprehensive description of spectral content, while the MFCC means capture various spectral envelope aspects, with tempo being relatively independent. This analysis underscores the intricate relationships among audio features, crucial for applications like music genre classification and audio content analysis.

Dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) were applied to visualize the high-dimensional data. PCA transforms the original features into a set of linearly uncorrelated components, ordered by the amount of variance they explain. By reducing the dataset to two principal components, a scatter plot was created, revealing clusters of genres. This visualization helps identify whether certain genres are naturally separable in the reduced feature space, which is crucial for effective classification. t-SNE, another dimensionality reduction technique, is particularly effective for visualizing high-dimensional data by preserving local structures. Applying t-SNE to the dataset provided a more detailed view of genre clustering, often revealing intricate patterns not apparent in PCA.

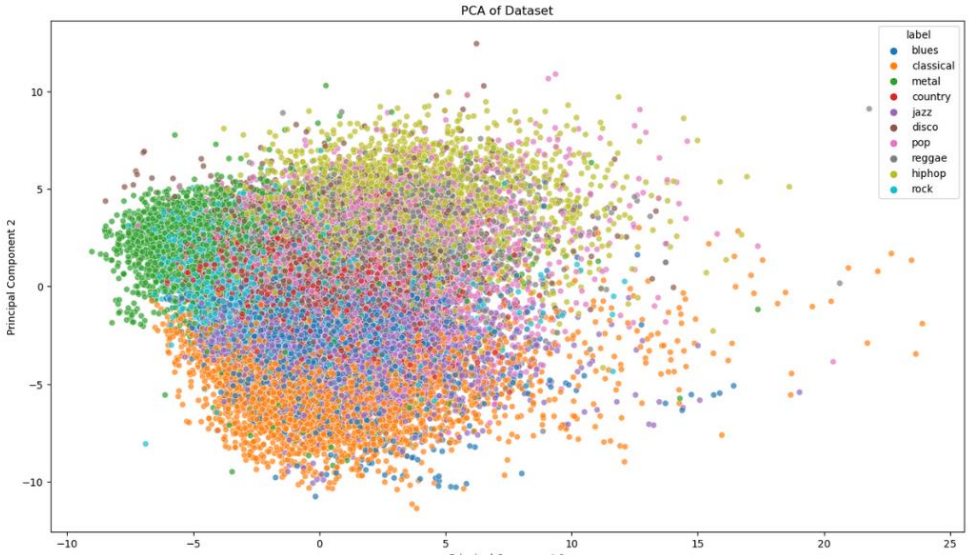

Figure 22. 2D Principal Component Analysis (PCA) of genres.

The PCA scatter plot shows a wide spread of data points, indicating significant variance captured by the first two principal components. However, the separation between different music genres is not distinctly visible, with notable overlap between genres such as metal, rock, and classical. This suggests that the linear combinations of features in PCA might not be sufficient for clear genre discrimination, highlighting its limitation in handling complex, non-linear relationships inherent in audio features.

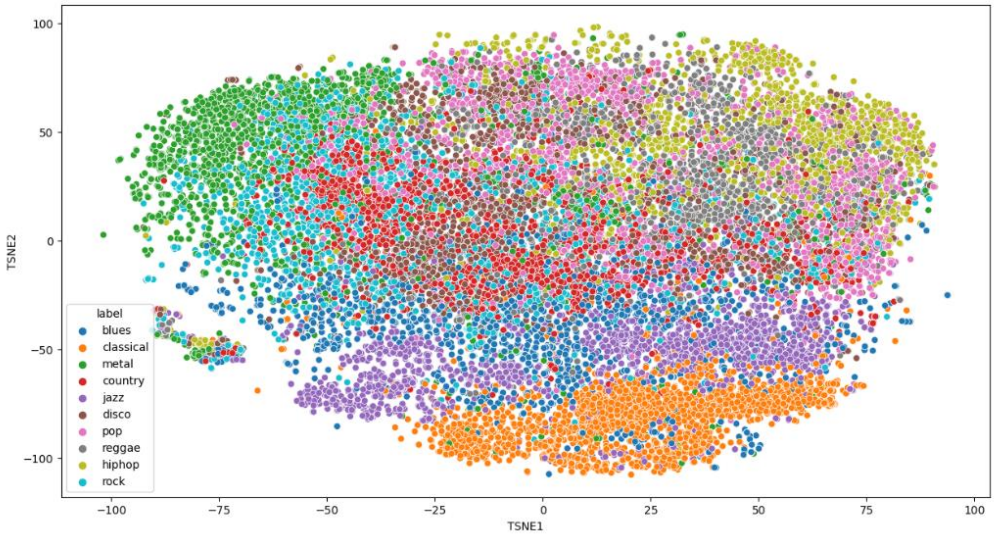

Figure 23. t-SNE Visualization of music genres.

In contrast, t-Distributed Stochastic Neighbor Embedding (t-SNE), a non-linear technique, provides a more defined clustering of music genres. The t-SNE visualization reveals distinct clusters for genres like classical, pop, and rock, with better separation and minimal overlap compared to PCA. This enhanced cluster formation indicates that t-SNE effectively captures the local structure and non-linear patterns in the data, making it more suitable for visualizing similarities within genres. However, some overlapping areas still exist, reflecting the inherent complexity and potential overlaps in music genre characteristics.

Comparatively, while PCA is efficient in reducing dimensions and preserving overall variance, it struggles with the non-linear relationships crucial for genre separation. t-SNE, on the other hand, excels in capturing these patterns, suggesting that non-linear classifiers might perform better for this task. These insights guide our feature

engineering and model selection processes, emphasizing the need for robust preprocessing steps such as feature normalization and advanced extraction techniques to enhance separability. Overall, the dimensionality reduction analysis provides a comprehensive understanding of the dataset's structure, informing our approach to model development in the music genre classification project.

CHAPTER 3. MODELLING

3.1 K-Nearest Neighbors

The first approach to our Music Genre Classification project is K-Nearest Neighbors (KNN), a non-parametric, supervised learning classifier which use proximity to make classifications about the grouping of an individual data point. Due to its simplicity and adaptability, choosing KNN helps establishing a strong baseline for the understanding of music data and future implementations of more sophisticated models.

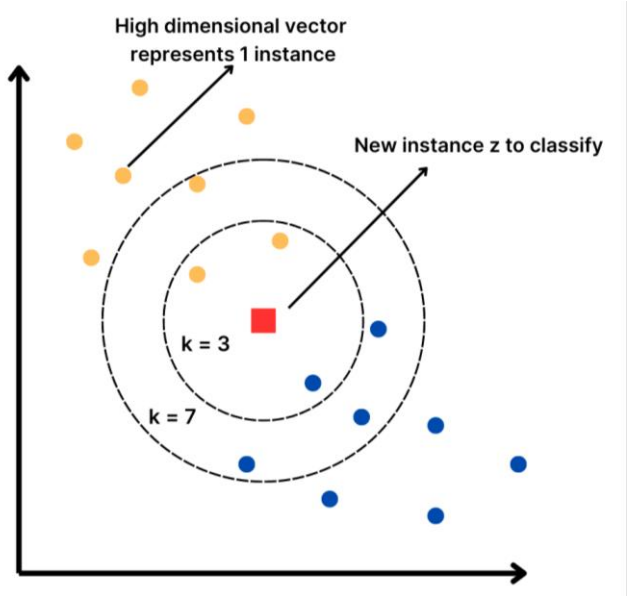

Figure 24. Illustration of K-Nearest Neighbors (KNN) Algorithm.

As shown in *Figure 24*, each instance in the training dataset D is represented by a vector in an n-dimensional space, e.g., $x_i = (x_{i1}, x_{i2}, \dots, x_{in})^T$ where each dimension represents an attribute. For a new instance z added to the space, the algorithm computes the distance between each instance x in D and z, then determines a set $NB(z)$ of the nearest neighbors of z and finally use the majority of the labels in $NB(z)$ to predict the label for z.

For the problem of Music Genre Classification, we evaluate the accuracy efficiency of KNN algorithm in our dataset by considering a range of hyper-parameters confessed below:

- ***n_neighbors*:** This parameter defines how many "neighbor" data points should be considered in the process of major voting. In practice, the number of *n_neighbors* can vary, but it is encouraged to be greater than 1 to avoid noise or error in only one nearest neighbor and not too large to avoid over-generalization.
- ***p*:** This parameter decide which distance metric we are going to use. The overall distance between two data points a and b in an n-dimensional space can be represented as:

$$d(a,b) = \sqrt[p]{\sum_{i=1}^{n}(a_i - b_i)^p}$$

  If we set p = 1 and p = 2, we will acquire the Manhattan Distance and the Euclidean distance, respectively. Different distance metrics can affect the neighbor-choosing process.

- ***weights*:** This parameter determines the weight assigned to each data point considered. Typically, in the "majority vote," every data point is given equal importance. However, this can sometimes introduce biases, leading to inconsistent predictions. Weighted "votes" address this issue by adjusting the importance of each vote according to specific rules. This adjustment can significantly impact the model's output, either positively or negatively, depending on the nature of the data. In the context of this problem, we consider the following values for weight metric:

- *'Uniform' weights:* All data points have the same weight. This is also the default setup implemented by Scikit-learn.

- *'Distance' weights:* Data points that are farther from the point being considered will have smaller weights. This ensures that closer data points have a greater influence on the prediction. The weight formula W based on the distance d can be expressed as follows:

$$W = \frac{1}{d^2}$$

The values of the above hyper-parameters considered in the implementation of this project is: *n_neighbors*: [1,8]; *p*: [1,2,3]; *weights*: ['uniform', 'distance'].

3.2 Support Vector Machine

The second approach to our Music Genre Classification project is Support Vector Machine (SVM), a supervised machine learning technique commonly known for classification purposes, and can also be utilized for regression tasks. SVM is especially proficient in dealing with datasets with a high number of dimensions, and is recognized for its ability to handle situations where the number of dimensions surpasses the number of samples. The primary advantage of SVM is its ability to identify the optimal hyperplane that effectively divides the data into distinct classes.

The main idea behind SVM is to find the hyperplane that can best separate classes by maximizing the distance between the hyperplane and the closest data points, also known as support vectors shown in *Figure 25*. This method of optimizing the margin enables SVM to accurately classify data points in a multi-dimensional space.

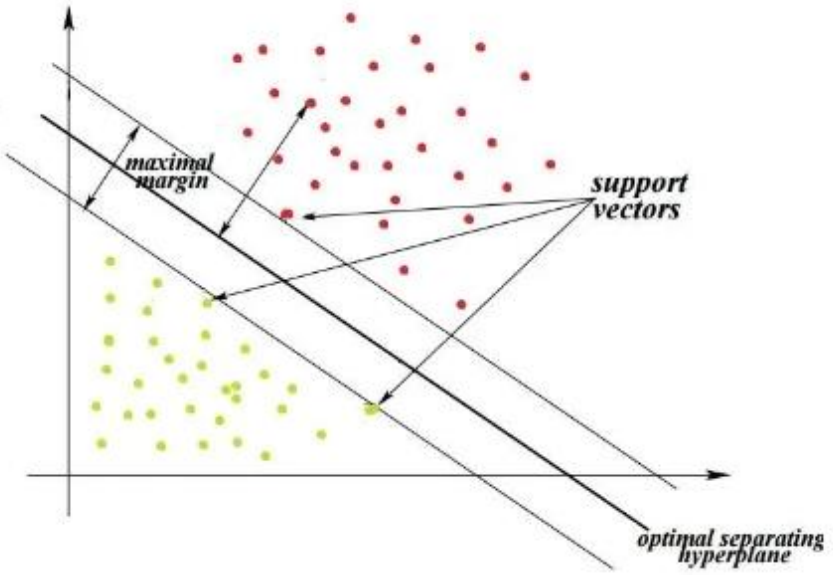

Figure 25. An example of how Support Vector Machine (SVM) works.

Support Vector Machines (SVM) employ kernel functions to convert non-linearly separable data into a higher-dimensional space as shonw in *Figure 26*, enabling better class separation through a hyperplane. This approach is essential when the original data cannot be separated by a linear boundary. By transforming the data in this manner, SVM can precisely classify and forecast results in intricate datasets.

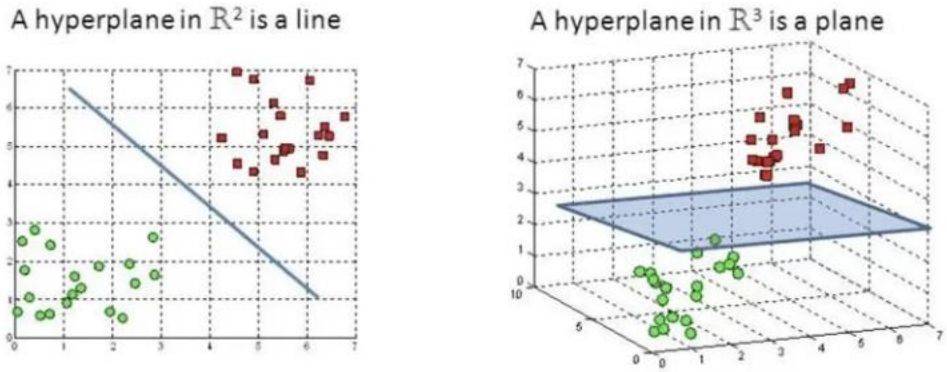

Figure 26. An example of how SVM works when employing kernel functions

Common kernel functions include:

- ***Linear Kernel*:** Suitable for linearly separable data.

$$k(x, z) = x^T z$$

- ***Polynomial Kernel*:** Useful for data that is not linearly separable but can be separated in a higher-dimensional space.

$$k(x, z) = (r + \gamma x^T z)^T$$

- ***Radial Basis Function (RBF) Kernel:*** Effective in scenarios where the decision boundary is complex and non-linear.

$$k(x, z) = \exp\left(-\gamma \left|\left|x - z\right|\right|^2\right)$$

- ***Sigmoid Kernel:*** Often used as an alternative to neural networks, the sigmoid kernel can introduce non-linearity similar to RBF. However, it may not perform as well in practice for high-dimensional spaces as RBF or polynomial kernels.

$$k(x, z) = \tanh(\gamma \mathrm{x}^T z + r)$$

In SVM, important hyper-parameters include the regularization parameter C and the parameters specific to the chosen kernel. The regularization parameter C plays a crucial role in balancing the desire to minimize training error with the need to avoid overfitting on the test data, ultimately impacting the margin of the classifier:

- ***Regularization Parameter C:*** A smaller C value in a SVM model increases the margin between classes, allowing for some misclassifications but promoting overall generalization. On the other hand, a larger C value aims to accurately classify all training examples, potentially leading to overfitting as the model may become too complex and unable to generalize well to unseen data as shown in *Figure 27*.
-

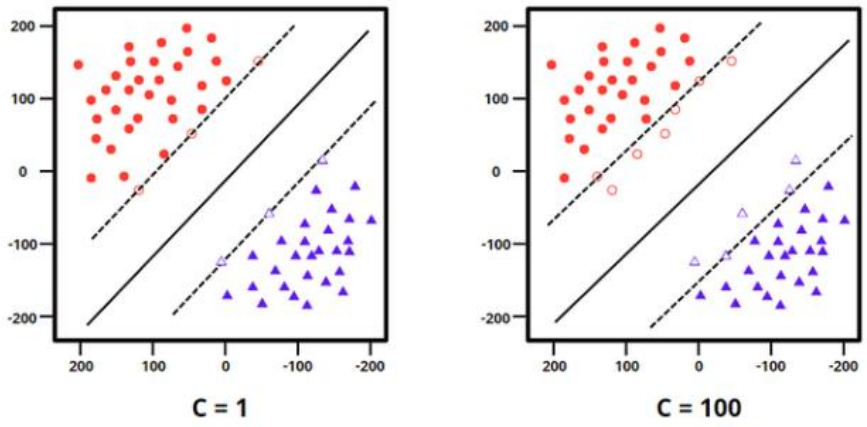

Figure 27. An example of the effect of parameter C.

- ***Kernel Parameters:*** For instance, the gamma value in the RBF and sigmoid kernels determines the extent to which a single training example impacts the model. This parameter plays a crucial role in fine-tuning the performance of the kernel methods.

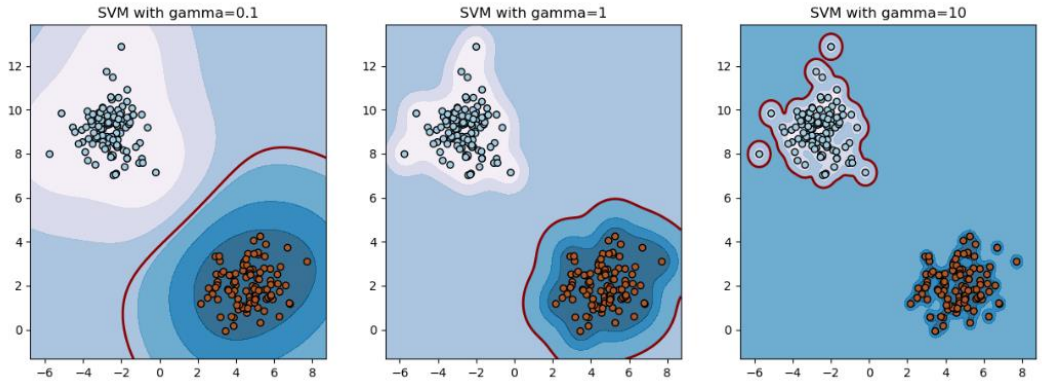

Figure 28. Example on the effects of gamma parameter.

The values of the above hyper-parameters considered in the implementation of this project are: *C:* [1, 10, 100, 200, 500]; *kernel:* ['poly', 'rbf'', 'sigmoid']; g*amma:* ['scale', 'auto'].

3.3 Neural Network

Music, with its intricate patterns, rhythms, and textures, can contains a lot of non-linear relationships and some traditional machine learning models can struggle to capture that. So we think about a deep learning approach that can automatically learn hierarchical feature representations directly from raw audio signals or spectrograms,

capturing nuance and subtle variations in music that are critical for distinguishing between genres. This ability to learn from large amounts of data (our dataset is pretty large, up to 30000 instances) and to capture intricate patterns makes deep learning particularly well-suited for tasks like music genre classification.

The neural network model is designed using the TensorFlow and Keras libraries. The architecture consists of the following layers are shown in *Figure 29*:

- ***Input Layer:*** A 'Flatten' layer to convert the input data into a one-dimensional array suitable for dense layers.
- ***Hidden Layers:*** Five 'Dense' layers with 300 - 300 - 300 - 300 - 300 neurons respectively, activated using the ReLU function, which helps the network learn non-linear patterns.
- ***Output Layer:*** A 'Dense' layer with 10 neurons (corresponding to the number of music genres) and a softmax activation function. The softmax function will assign a probability to each class, with the sum of probabilities equaling 1.

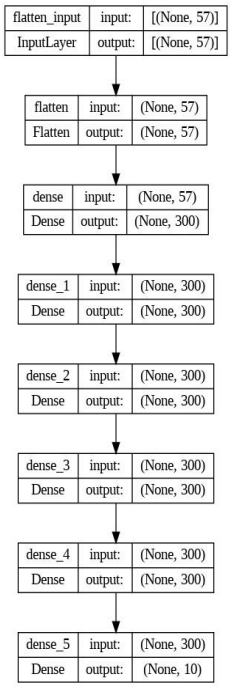

Figure 29. Multi-layer Perceptron (MLP) network architecture.

The network uses a variant of the gradient descent optimization algorithm named Adam. The learning rate is set very low (1e-4), which means the steps taken towards the minimum of the loss function will be small, potentially providing more precise convergence at the risk of taking more time to train.

The core idea of this neural network is to leverage a multi-layer perceptron (MLP) architecture to learn the underlying patterns in audio features for music genre classification. By training on a labeled dataset, the model adjusts its parameters to minimize the classification error, aiming to achieve high accuracy in predicting the genre of new, unseen music tracks. The combination of dense layers and appropriate activation functions enables the network to capture complex relationships in the data.

3.4 Stacking Ensemble

Now we already have three different models: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and a Neural Network (NN). Each of these models possesses unique characteristics that make them suitable for different types of data and learning tasks. KNN excels in capturing local patterns, SVM thrives in high-dimensional spaces, while NN can model complex, non-linear relationships. By combining these independent models, stacking leverages their complementary strengths to produce more robust predictions.

By combining these models, stacking takes advantage of their diverse strengths. While KNN might excel in one part of the data, SVM might perform better in another, and NN might capture intricate patterns missed by the others. This diversity helps in covering different aspects of the data distribution, leading to a reduction in overall error. Individual models might suffer from high bias (underfitting) or high variance (overfitting). Stacking helps balance these issues by averaging out the errors of individual models. For instance, a high-bias model can be compensated by a low-bias but high-variance model, and vice versa...

Schematically, our stacking ensemble model looks like this:

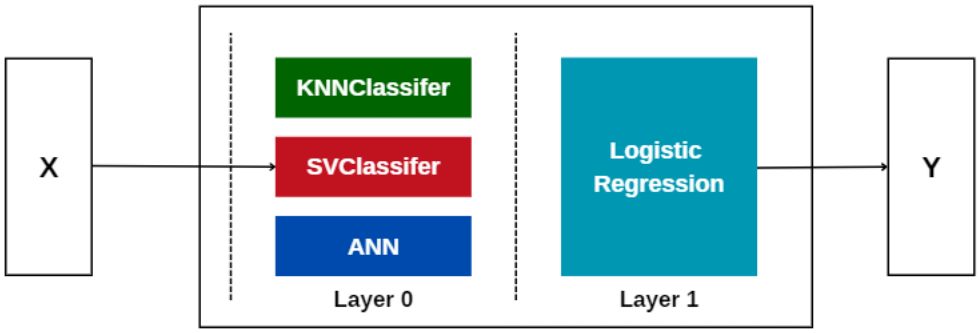

Figure 30. Stack Model: fitting step 1.

Properly combined models in a stacked ensemble tend to generalize better to unseen data. This is because the weaknesses of one model are mitigated by the strengths of another, leading to improved performance on test data. The first step involves training several base models (KNN, SVM, NN) on the training data. Each model will generate predictions on this data. These predictions are then used as inputs to a meta-model (also called a second-level model or combiner model, in our work we used Logistic Regression), which learns how to best combine these predictions to make the final decision.

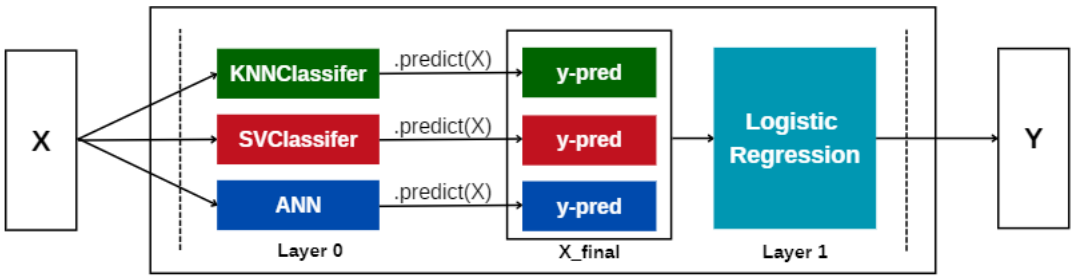

Figure 31. How the stacking ensemble makes prediction.

To avoid overfitting, the training of the meta-model typically involves a cross-validation procedure (Instead of fitting the whole dataset for each base model). It splits the training data into k-folds, trains each base model on k-1 folds, and makes predictions on the remaining fold. Collect the out-of-fold predictions for each base model. These predictions form a new dataset, which is then used to train the meta-model. When making predictions on new data, each base model makes a prediction,

and these predictions are then used as inputs to the meta-model, which produces the final prediction.

We set pass_through=True, now the final meta-model not only learns from the predictions of the base models but also has access to the original features as shown in *Figure 32*. This can potentially improve the performance of the stacked ensemble model by providing more information to the final meta-model for making decisions.

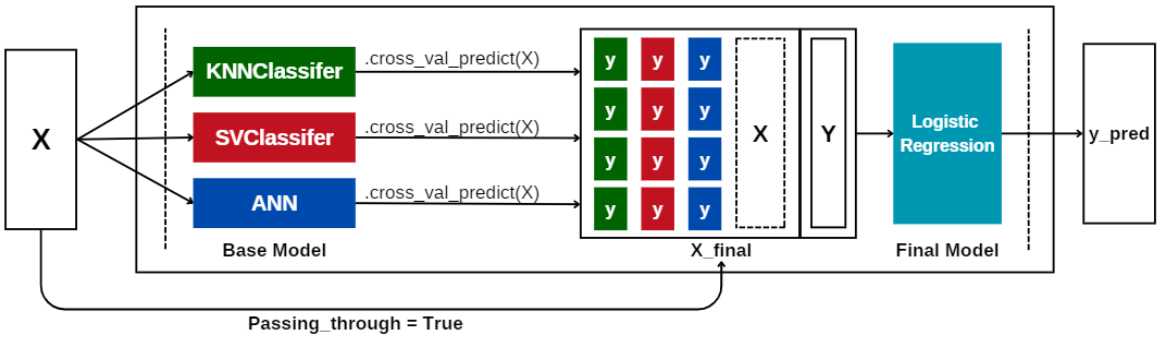

Figure 32. Passthrough X to X_final to add information for final learning

By combining the strengths of each model, we expect stacking to create a robust predictive model that is typically more accurate than any of the individual models alone.

CHAPTER 4. EVALUATION

4.1 Performance of Models

4.1.1 Evaluating Models by metrics

We evaluate the performance of our models using widely recognized metrics for classification tasks. These include Accuracy, which measures the proportion of correctly classified instances, Precision, which quantifies the accuracy of the positive predictions, Recall, which assesses the model's ability to identify all relevant instances, and the F1 Score, which provides a harmonic mean of Precision and Recall, balancing the trade-off between them. By employing these metrics, we ensure a comprehensive assessment of our models' effectiveness in various aspects of classification. The table of experimental result based on listed metrics is shown below:

Table 1. Performance of models based on proposed metrics.

Models	**Accuracy**	**Precision**	**Recall**	**F1**
K-Nearest Neighbors	0.80	0.83	0.81	0.81
Support Vector Machine	0.70	0.71	0.70	0.70
Neural Network	0.80	0.79	0.79	079
Stacking Ensemble	0.84	0.85	0.85	0.85

4.1.2 Models Comparison

The Stacking Ensemble model consistently outperforms the other models across all metrics—Accuracy, Precision, Recall, and F1 Score. This indicates that it is the most reliable and effective model among the four evaluated. K-Nearest Neighbors is the second-best model, particularly noted for its high precision and recall. The Neural Network performs moderately well across all metrics but does not surpass the Stacking Ensemble or K-Nearest Neighbors. The Support Vector Machine has the lowest performance in all evaluated metrics.

Despite its simplicity and fast speed, the K-Nearest Neighbors (KNN) algorithm delivers remarkably good performance. Its ease of implementation and efficiency contribute to its high accuracy and balanced precision and recall. On the other hand, while the Support Vector

Machine (SVM) is a powerful method, it may not be the best choice in this context. SVM is inherently designed for binary classification and does not support multiclass classification natively. Addressing multiclass problems with SVM requires decomposing them into multiple binary classification tasks, which can be complex and less efficient.

Neural Network exhibits strong performance across various metrics, although we just tested for a random architecture and haven't tuned it yet. However, the computational demands can be a limitation in environments with restricted resources or the need for rapid model deployment. So unless we find the best architecture of the neural network that significantly outperforms the KNN, we would still choose KNN as our best single model for its excellent performance and speed.

The Stacking Ensemble method stands out with the highest performance metrics across the board, showcasing its potential in combining the strengths of multiple models such as KNN, SVM, Neural Network, and a final model (Logistic Regression in this case). However, this method necessitates a relatively long training time due to the complexity of integrating multiple models. Until computational resources advance sufficiently to mitigate this barrier, KNN might be the more practical choice for a modest decrease in performance. This approach prioritizes efficiency and speed, avoiding the extended wait times associated with the ensemble model's predictions.

4.2 Overfitting problem and Fine-tuning the hyper-parameters

4.2.1 Cross Validation

The very first idea when we were trying to implement any machine learning models is to make the most reasonable prediction is: to fit a model on a "training" data set and evaluate its performance on a separate held-out "testing" data (which is supposed to simulate how our model will perform in the real world). But there's absolutely a problem with the simple train-test split approach. When we use a single train-test split, 80/20 for example, it's possible that our X_test and y_test splits won't be representative of the type of data our models will encounter in production. This is a problem because it means that our model's performance on the test split might not be

a reliable estimate of performance in production (in other words, an overfitting problem).

The solution to this is to use cross-validation, which involves creating multiple train-test splits, training and evaluating your model on each of these splits separately and calculating the average performance across all testing splits.

Figure 33. The way cross-validation compares models (source: Matt Chapan on Medium).

As the figure shows, a cross-validation process always starts by creating the different train-test splits. In our case, we're using 5-fold cross-validation, so we'll create 5 versions of the training and testing data. Next, for each split, we train the model on the Train split and calculate its performance on the Test split. Finally, we compute the average score across all Test splits. This gives us a much more reliable/realistic picture of our model's ability, which is less likely to be biased by the particular splitting strategy we used.

We think about cross-validation as a way to reduce the risk of overfitting, as the model is trained and validated on different subsets of the data multiple times. This ensures that the model's performance is not overly optimistic and is likely to generalize well to new, unseen data. Moreover, cross-validation provides a more comprehensive measure of model accuracy by averaging the results over multiple folds, which accounts for variability in the data and ensures that the model is reliable. The thing that we appreciate about this technique is that it maximizes the use of the available

data, particularly useful, by allowing every data point to be used for both training and validation purposes.

4.2.2 Gird Search and Best Parameters

In the context of cross-validation, GridSearchCV from sklearn library is a powerful tool used to perform an exhaustive search over a specified parameter grid for a given model, combining cross-validation with hyper-parameter tuning. By systematically working through multiple combinations of parameter values, GridSearchCV helps identify the optimal set of parameters that yield the best performance for the model.

For the K-Nearest Neighbors (KNN) model, the primary parameters tuned were the number of neighbors (n_neighbors) and the distance metric (metric). The n_neighbors parameter determines how many neighbors are considered for classifying a point, while the metric parameter defines the distance function used to calculate the closeness between points. The parameter grid for KNN included:

- *n_neighbors:* [1,8]
- *p:* [1,2,3]
- *weights:* [’uniform’,’distance’]

After performing the exhaustive grid search on of the dataset using 5-fold cross-validation, We found out {*n_neighbors:* 2, *weights:* ‘distances’, *p:* 1} to be the optimal parameters for our whole dataset.

In the case of the Support Vector Classifier (SVC), several parameters were optimized, including the regularization parameter (C), kernel type (kernel), and kernel coefficient (gamma) for certain kernels. The C parameter controls the trade-off between achieving a low training error and a low testing error, while the kernel parameter specifies the type of kernel to be used, such as linear, polynomial, radial basis function (rbf), or sigmoid. The gamma parameter defines the influence of individual training examples in determining the decision boundary.

The parameter grid for SVC includes:

- *C:* [1, 10, 100, 200, 500]
- *kernel:* ['poly', 'rbf', 'sigmoid']
- *gamma:* ['scale', 'auto']

Through the process, we've found that {*C:* 200, *kernel:* 'rbf', *gamma:* 'scale'} is the most optimal configuration of this algorithm for this project.

So in our hyperparameter tuning process here, we utilized GridSearchCV, We implemented cross-validation for each model to systematically evaluate different combinations of these parameters. The process involved defining the parameter grid for each model, setting up the GridSearchCV object with the model, parameter grid, and StratifiedKFold cross-validation strategy, fitting the GridSearchCV object to the training data to perform the exhaustive search, and extracting the best parameters based on cross-validated performance metrics.

4.2.3 Early Stopping for Artificial Neural Network

While fine-tuning K-Nearest Neighbors (KNN) and Support Vector Classification (SVC) models, we utilized GridSearchCV with cross-validation to optimize hyperparameters effectively. This method, however, becomes highly computationally intensive when applied to more complex models such as neural networks. The time complexity and resource demands of GridSearchCV on neural networks can be prohibitive, making it an impractical choice for larger datasets or deeper architectures. Seeking a more efficient solution, we turned to the early stopping technique provided by Keras.

Early stopping in Keras is a form of regularization used to prevent overfitting in neural networks. It works by monitoring the model's performance on a validation dataset and halting training when the performance stops improving. Specifically, during training, Keras tracks the validation loss, and if it doesn't improve for a specified number of

epochs (patience), the training process is terminated as shown in *Figure 34*. This approach ensures that the model does not waste computational resources by training indefinitely once it has reached an optimal point, thus significantly reducing the computational load compared to exhaustive methods like GridSearchCV.

The primary advantage of early stopping is its efficiency. By dynamically deciding when to stop training based on actual performance rather than predefined iterations, it can save considerable time and computational power. Additionally, early stopping inherently mitigates the risk of overfitting (which makes it a great alternative to cross-validation), as it ceases training once the model begins to overfit the training data, evidenced by the stagnation or increase in validation loss.

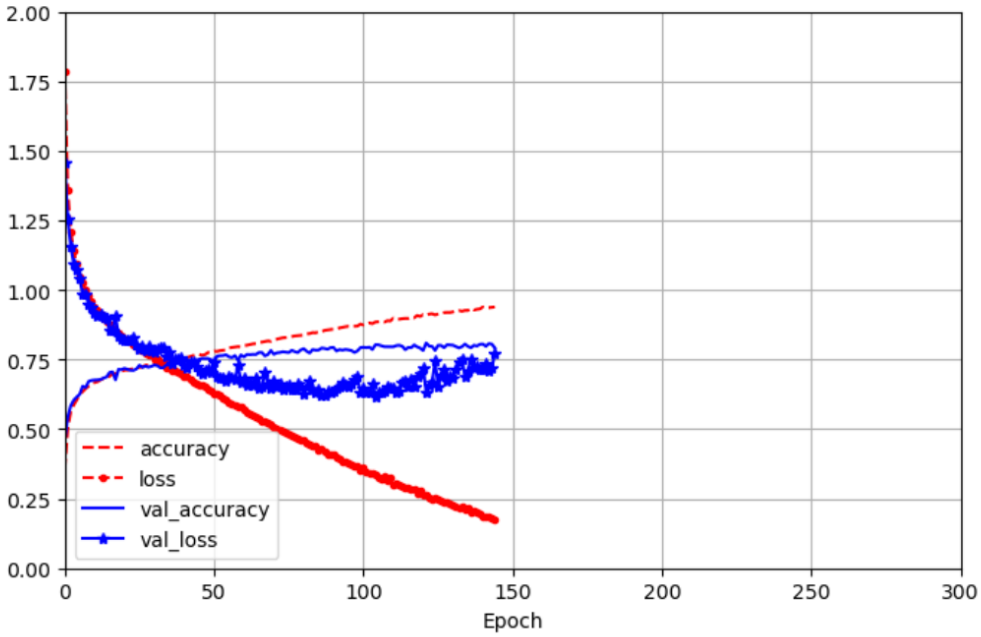

Figure 34. The callbacks early stopping from Keras terminate the training process if it senses the abnormal increase in val_loss, which indicates the sign of overfitting.

However, early stopping also has its drawbacks. Unlike GridSearchCV, which provides a systematic search over a predefined hyperparameter space, early stopping does not explore different configurations exhaustively. As a result, it might miss the optimal combination of hyperparameters that could have been found through a more extensive search. Moreover, while early stopping is effective in preventing

overfitting, it does not address the initial selection of hyperparameters, which might still require some level of manual tuning or an additional layer of automated search.

In our perspective, integrating both GridSearchCV and early stopping can be a powerful enhancement for neural network optimization, provided we have sufficient computational resources. This kind of hybrid approach would allow us to systematically explore various hyperparameter configurations while also utilizing early stopping to efficiently manage training time and prevent overfitting. This strategy could lead to finding the best parameters for the neural network, balancing thorough exploration with computational efficiency.

For now, our current Keras model utilizing early stopping is performing well. It effectively reduces overfitting and achieves good performance without the extensive computational overhead required by GridSearchCV. Looking ahead, the integration of both methods represents a promising avenue for further enhancing our neural network tuning process. We will definitely try this method if we have sufficient computational resources and time.

4.2.4 Models Performance after Regularization Techniques

Table 2 Model Performance after Regularization Techniques

Models	**Accuracy**	**Precision**	**Recall**	**F1**
K-Nearest Neighbors	0.86	0.86	0.86	0.86
Support Vector Machine	0.81	0.81	0.81	0.81
Neural Network	0.78	0.78	0.78	0.78
Stacking Ensemble	0.88	0.88	0.88	0.88

After conducting grid searches to find the best hyperparameters, we observed notable improvements in the performance of our machine learning models as observed in *Table 2*. The accuracy of the k-Nearest Neighbors (KNN) model increased from 0.80 to 0.86, and the Support Vector Machine (SVM) improved from 0.70 to 0.81, which is quite significant. However, the performance of the neural network decreased by

approximately 1%. This was expected, as we did not perform a thorough hyperparameter optimization for the neural network due to computational resource limitations. Instead, we employed early stopping to prevent overfitting. Although the new neural network model performs slightly worse on the test set compared to the previous version, we believe it will generalize better in real-world applications. If we had more time and resources, we could conduct a grid search for the neural network to develop a more robust model. Nonetheless, the current performance is acceptable. Additionally, the performance of our stack ensemble model improved from 0.84 to 0.88. Despite its higher time complexity, the stack ensemble delivered the best results. Therefore, we have decided to use the stack ensemble as our final classifier for optimal prediction accuracy.

CHAPTER 5. DEPLOYMENTS

The Music Genre Classification web application is designed to provide an accurate prediction of a song's genre through a sophisticated multi-step process. When a user uploads a song, the application first divides the track into multiple 30-second segments. This segmentation allows the system to analyze diverse portions of the song, ensuring a comprehensive capture of its audio features across its entire duration. Each of these segments is then processed through a series of pre-trained machine learning models: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Neural Network (NN), and a Stacking (Ensemble Learning) model. The workflow is described in the figure below:

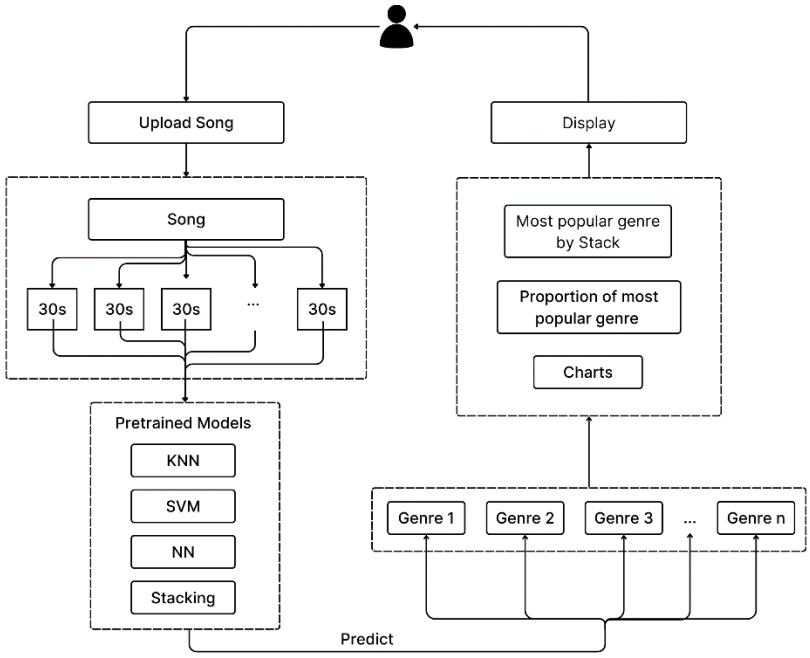

Figure 35. Workflow of Music Genre Classification Application.

As each 30-second segment is analyzed by these models, they collectively predict the genre for that specific segment. The stacking model then aggregates these individual predictions to determine the most frequent genre predicted across all segments, identifying the most popular genre for the entire song. This multi-model approach ensures efficient and accurate genre classification by accounting for different aspects and segments of the song.

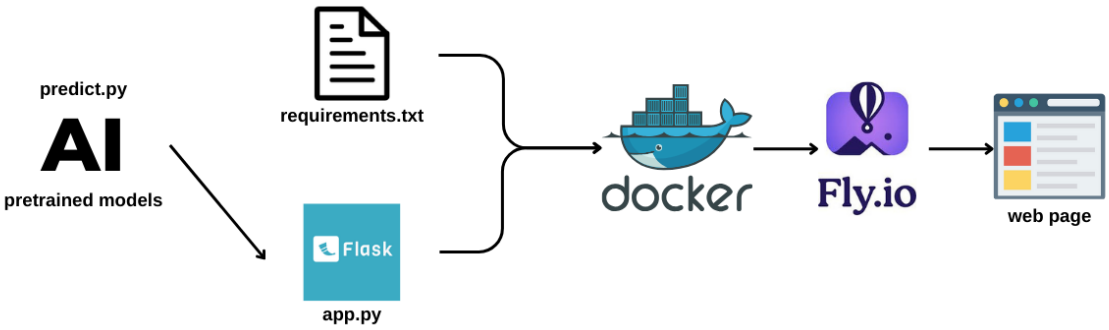

Figure 36. Deploying a Flask Application with Pretrained Models using Docker and Fly.io.

We developed a Docker image to package all dependencies and configurations, ensuring consistent performance across various environments. This Dockerized application was deployed on Fly.io, making it accessible online with an enhanced user interface compared to the local version as shown in *Figure 36*.

Once the genre prediction process is complete, the web interface displays the results to the user.

CHAPTER 6. FURTHER ENHANCEMENT

6.1 Feedback

The Music Genre Classification project aims to automate and improve the categorization of music tracks. Feedback from initial users, including some of our friends, has highlighted several strengths and areas for improvement. Users appreciated the intuitive graphical interface of the deployed website, and the prediction accuracy was relatively high for most of their attempts. However, some issues were noted, particularly in classifying Vietnamese traditional songs. Users also requested a more powerful model to support a broader range of genres, as they felt the current genre set was too limited. These issues stem from our dataset, which includes diverse songs from around the world but lacks sufficient representation from our own country. Additionally, the GTZAN dataset we used is restricted to only ten genres. This valuable feedback will guide us in enhancing our machine learning model to be more robust and versatile, capable of addressing these challenges.

6.2 Future Research Directions

Future research could expand the genre classification model by collecting and annotating data for more niche and emerging genres, including sub-genres and hybrid genres, enhancing its precision for diverse musical tastes. Accuracy and robustness could be improved through advanced machine learning techniques like deep learning models designed for audio analysis, such as convolutional neural networks, and leveraging transfer learning from large pre-trained audio datasets. Enhanced feature engineering, including new temporal and harmonic features and improved feature extraction and selection methods, could provide more informative inputs, streamlining the model. User interface and experience improvements, such as redesigning the graphical interface for better engagement and incorporating user feedback mechanisms, along with developing mobile and web applications, could broaden accessibility. Integration with music streaming services through APIs or plugins could enhance recommendation systems, and studying the model's impact on

user satisfaction and engagement could provide valuable data for further refinement. Cross-domain applications in music therapy, education, and automated DJing could leverage the model for personalized music experiences. Enhanced deployment and scalability, including optimizing Docker images for efficiency and implementing scalable cloud solutions for real-time classification, would make the tool more robust and reliable for commercial use.

REFERENCES

Khoat Than. (2024, Semester 2). IT3190E - Machine Learning. Lecture presented at Hanoi University of Science and Technology, March 2024.

Tiep Vu Huu, "Machine Learning cơ bản", published March 27th, 2018.

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron - Released September, published in 2019, 39-129

Article Medium.

Sturm, Bob L. "The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use." arXiv preprint arXiv:1306.1461 (2013).

Ghosh, Partha & Mahapatra, Soham & Jana, Subhadeep & Jha, Ritesh. (2023). A Study on Music Genre Classification using Machine Learning. International Journal of Engineering Business and Social Science. 1. 308-320. 10.58451/ijebss.v1i04.55.

A. Sandra Grace, "Song and Artist Attributes Analysis For Spotify," 2022 International Conference on Engineering and Emerging Technologies (ICEET), Kuala Lumpur, Malaysia, 2022, pp. 1-6, doi: 10.1109/ICEET56468.2022.10007360.

Andrana, "Work w/ Audio Data: Visualise, Classify, Recommend" notebook on Kaggle.

McFee, Brian & Raffel, Colin & Liang, Dawen & Ellis, Daniel & Mcvicar, Matt & Battenberg, Eric & Nieto, Oriol. (2015). librosa: Audio and Music Signal Analysis in Python. 18-24. 10.25080/Majora-7b98e3ed-003.

Lucas B.V. de Amorim, George D.C. Cavalcanti, Rafael M.O. Cruz. The choice of scaling technique matters for classification performance (23 Dec 2022).

Alexandropoulos, Stamatios-Aggelos & Aridas, Christos & Kotsiantis, Sotiris & Vrahatis, Michael. (2019). Stacking Strong Ensembles of Classifiers. 10.1007/978-3-030-19823-7_46.

Hossin, Mohammad & M.N, Sulaiman. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. International Journal of Data Mining & Knowledge Management Process. 5. 01-11. 10.5121/ijdkp.2015.5201.

Bates, Stephen, Trevor Hastie, and Robert Tibshirani. "Cross-validation: what does it estimate and how well does it do it?." Journal of the American Statistical Association (2023): 1-12.