Report

Prayag Parashar **Prince Yadav**

2022377 **2022379**

Problem Being Addressed

Music genre classification is a fundamental task in music information retrieval that assists in organizing vast amounts of music data, recommending systems, and music analysis. This project utilizes the GTZAN dataset, which is popular for benchmarking classification models but also known for its challenges like noise in genre labels and data integrity.

Relevant Literature

Several studies have used the GTZAN dataset for genre classification, employing various machine learning techniques:

1. Tzanetakis and Cook (2002) introduced the GTZAN dataset in their foundational paper, highlighting its use for feature extraction and genre classification.
2. Recent works have incorporated deep learning methods, notably Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), demonstrating significant improvements in classification accuracy over traditional methods like Support Vector Machines (SVMs) and K-Nearest Neighbors (KNN).

Methodology :

1. **Data Preparation:**

   Data Loading: Each audio file from the GTZAN dataset, which consists of tracks each 30 seconds long, is loaded into the system.

   Feature Extraction: From each audio track, Mel-frequency cepstral coefficients (MFCCs) are extracted. MFCCs are chosen because they effectively represent the power spectrum of a sound, capturing timbral and textual aspects which are important for distinguishing music genres.

   The calculation of the Mel-cepstrum is described by equation:

   $$f_{mel} = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

   Segmentation: Each track is divided into shorter segments, improving the granularity of analysis and increasing the amount of data available for training. This helps in building a more robust model by learning from various segments within the same track, which might exhibit slight variations in musical elements.

Normalization: Features are normalized to ensure that the model isn't biased towards variables with higher magnitude.

2. **Data Count**

Tracks: The GTZAN dataset contains 1,000 audio tracks evenly distributed across 10 genres, each genre comprising 100 tracks.

Segments: If each track is divided into, say, 5 segments, the effective dataset size becomes 5,000 samples.

3. **Model Architecture Design**

The Neural Network used in this project is a Multi-Layer Perceptron (MLP), designed as follows:

Input Layer: The input layer size depends on the number of features extracted per segment. For instance, if 13 MFCCs are calculated and each track is segmented into 5 parts, the input features for each segment form the initial input layer.

Hidden Layers: Multiple hidden layers can be used, each with ReLU (Rectified Linear Unit) activation functions. The choice and number of neurons in each hidden layer are usually determined based on the complexity of the problem and the amount of available data. A common configuration might include layers with decreasing numbers of neurons, such as 256, 128, and 64, to gradually learn more abstract representations.

Output Layer: The output layer consists of as many neurons as there are genres (10 in this case), using a softmax activation function to output the probability distribution across the genre classes.

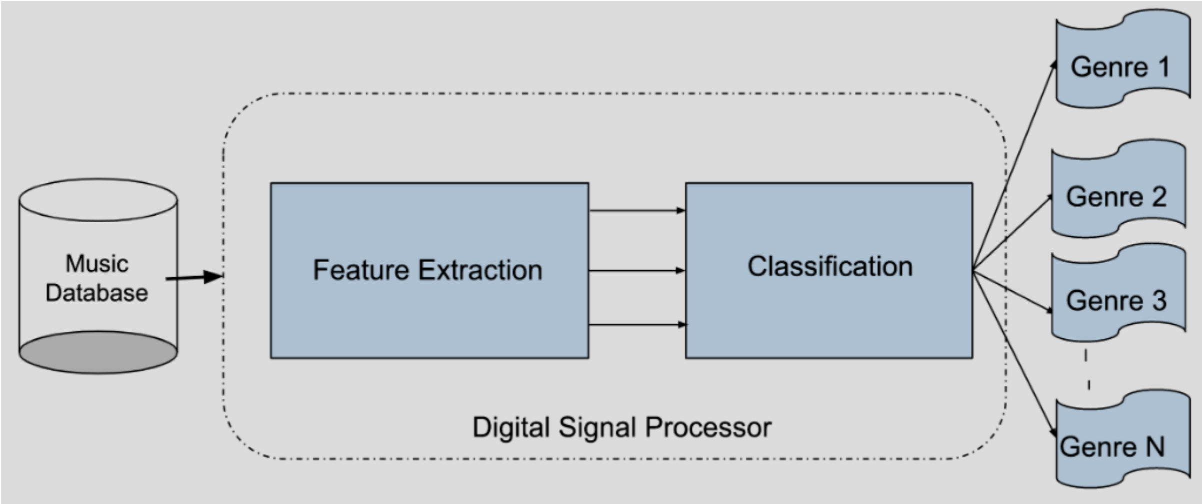
4. **Training Configuration**

Optimizer: The Adam optimizer is utilized for its efficiency in handling sparse gradients and adaptive learning rate capabilities.

Loss Function: Categorical crossentropy is employed as the loss function, which is suitable for multi-class classification problems.

Batch Size and Epochs: The model is trained in batches (e.g., 32 samples per batch) for a set number of epochs (e.g., 100 epochs). These parameters can be tuned based on the performance and computational resources.

Validation: Using K-fold cross-validation (e.g., 5 folds), the dataset is split into training and validation sets multiple times to ensure that every data point has been used for both training and validation. This helps in assessing the model's performance and its ability to generalize to unseen data.



Experimental Settings

Neural Network Architecture

The architecture of a neural network determines how well it can learn from the data by capturing relationships between input features and output classes. Here's a breakdown of each layer and its role within the Multi-Layer Perceptron (MLP) used in your project:

Input Layer:

Function: The input layer receives the feature set for each data sample, which, in this case, consists of Mel-frequency cepstral coefficients (MFCCs). This layer's size is equal to the number of features extracted per audio segment.

Role: It acts as the entry point for data, feeding normalized feature values into the network.

Hidden Layers:

Function: These layers are where most of the computation happens. Each hidden layer consists of a number of neurons, each connected to all neurons in the previous layer.

Neurons: Typically equipped with a non-linear activation function like ReLU (Rectified Linear Unit). ReLU is chosen for its ability to introduce non-linearity, helping the network learn complex patterns. It works by outputting the input directly if it is positive; otherwise, it outputs zero.

Role: Each layer's neurons combine inputs from the previous layer with a set of coefficients, or weights; these weights are adjusted during training. As data passes through each hidden layer, the network learns more refined features, with deeper layers capturing higher-level features.

Output Layer:

Function: The final layer in an MLP, having as many neurons as there are output classes (in this case, 10 genres). It uses the softmax activation function, which is a generalization of the sigmoid function to multiple classes.

Role: This layer converts the values from the last hidden layer into probabilities by outputting a probability distribution across the genre classes, thus making the final classification decision.

Training Configuration

Optimizer: Adam optimizer is used due to its efficiency in handling varying data scales and adaptive learning rate adjustments, which helps converge faster.

Loss Function: Categorical crossentropy is used for calculating the loss. It measures the difference between the true distribution (actual labels) and the predicted probability distributions output by the final layer, providing a robust objective for classification tasks.

K-fold Cross-validation

K-fold cross-validation is a technique used to validate the stability and reliability of the model and to ensure that it generalizes well to an independent dataset. Here's how it's implemented:

Partitioning: The dataset is randomly split into 'K' equal-sized subsets or folds. Common choices for K are 5 or 10, providing a good balance between training time and model validation thoroughness.

Iteration: For each fold:

Test Set: One fold is retained as the test set for validating the model.

Training Set: The remaining K-1 folds are used as the training set.

Model Training and Validation: The model is trained on the K-1 training folds, and the validation is done on the test fold. This process is repeated K times, with each of the K folds used exactly once as the test set.

Aggregation: The results from each fold can be averaged to produce a single estimation. This estimation provides insight into how the model is expected to perform in general when used on unseen data.

Results:

For Neural Network:

Fold 1:

```
Epoch 90/100
250/250 [==============================] - 4s 16ms/step - loss: 1.4446 - accuracy: 0.5618 - val_loss: 1.7067 - val_accuracy: 0.5170
Epoch 91/100
250/250 [==============================] - 4s 17ms/step - loss: 1.4532 - accuracy: 0.5593 - val_loss: 1.7248 - val_accuracy: 0.5180
Epoch 92/100
250/250 [==============================] - 4s 16ms/step - loss: 1.4388 - accuracy: 0.5675 - val_loss: 1.6870 - val_accuracy: 0.5250
Epoch 93/100
250/250 [==============================] - 4s 16ms/step - loss: 1.4233 - accuracy: 0.5696 - val_loss: 1.6891 - val_accuracy: 0.5240
Epoch 94/100
250/250 [==============================] - 4s 17ms/step - loss: 1.4008 - accuracy: 0.5737 - val_loss: 1.6990 - val_accuracy: 0.5165
Epoch 95/100
250/250 [==============================] - 4s 16ms/step - loss: 1.3735 - accuracy: 0.5861 - val_loss: 1.6961 - val_accuracy: 0.5185
Epoch 96/100
250/250 [==============================] - 3s 11ms/step - loss: 1.3907 - accuracy: 0.5775 - val_loss: 1.6659 - val_accuracy: 0.5190
Epoch 97/100
250/250 [==============================] - 3s 11ms/step - loss: 1.3949 - accuracy: 0.5832 - val_loss: 1.6376 - val_accuracy: 0.5375
Epoch 98/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3761 - accuracy: 0.5825 - val_loss: 1.6435 - val_accuracy: 0.5340
Epoch 99/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3704 - accuracy: 0.5866 - val_loss: 1.7000 - val_accuracy: 0.5205
Epoch 100/100
250/250 [==============================] - 3s 10ms/step - loss: 1.3449 - accuracy: 0.5950 - val_loss: 1.6316 - val_accuracy: 0.5315
63/63 [==============================] - 0s 4ms/step
```

Fold 2:

```
Epoch 90/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4991 - accuracy: 0.5296 - val_loss: 1.7409 - val_accuracy: 0.4872
Epoch 91/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4863 - accuracy: 0.5310 - val_loss: 1.7401 - val_accuracy: 0.4847
Epoch 92/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4766 - accuracy: 0.5354 - val_loss: 1.7027 - val_accuracy: 0.4982
Epoch 93/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4751 - accuracy: 0.5436 - val_loss: 1.6904 - val_accuracy: 0.5058
Epoch 94/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4574 - accuracy: 0.5386 - val_loss: 1.6909 - val_accuracy: 0.5103
Epoch 95/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4555 - accuracy: 0.5461 - val_loss: 1.6658 - val_accuracy: 0.5078
Epoch 96/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4351 - accuracy: 0.5464 - val_loss: 1.6860 - val_accuracy: 0.5138
Epoch 97/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4170 - accuracy: 0.5516 - val_loss: 1.6939 - val_accuracy: 0.5163
Epoch 98/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4042 - accuracy: 0.5580 - val_loss: 1.6978 - val_accuracy: 0.5023
Epoch 99/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4040 - accuracy: 0.5599 - val_loss: 1.6194 - val_accuracy: 0.5283
Epoch 100/100
250/250 [==============================] - 2s 9ms/step - loss: 1.3906 - accuracy: 0.5644 - val_loss: 1.6604 - val_accuracy: 0.5108
63/63 [==============================] - 0s 4ms/step
```

Fold 3:

```
Epoch 90/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4521 - accuracy: 0.5465 - val_loss: 1.7408 - val_accuracy: 0.4887
Epoch 91/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4320 - accuracy: 0.5516 - val_loss: 1.7115 - val_accuracy: 0.4847
Epoch 92/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4538 - accuracy: 0.5415 - val_loss: 1.6881 - val_accuracy: 0.4947
Epoch 93/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4221 - accuracy: 0.5569 - val_loss: 1.6379 - val_accuracy: 0.4817
Epoch 94/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4117 - accuracy: 0.5554 - val_loss: 1.6606 - val_accuracy: 0.5013
Epoch 95/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4141 - accuracy: 0.5565 - val_loss: 1.6711 - val_accuracy: 0.4912
Epoch 96/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3946 - accuracy: 0.5589 - val_loss: 1.6533 - val_accuracy: 0.5053
Epoch 97/100
250/250 [==============================] - 2s 9ms/step - loss: 1.3835 - accuracy: 0.5682 - val_loss: 1.6846 - val_accuracy: 0.4887
Epoch 98/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3711 - accuracy: 0.5673 - val_loss: 1.6668 - val_accuracy: 0.4962
Epoch 99/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3690 - accuracy: 0.5689 - val_loss: 1.6610 - val_accuracy: 0.4962
Epoch 100/100
250/250 [==============================] - 2s 10ms/step - loss: 1.3583 - accuracy: 0.5734 - val_loss: 1.6550 - val_accuracy: 0.5093
63/63 [==============================] - 0s 4ms/step
```

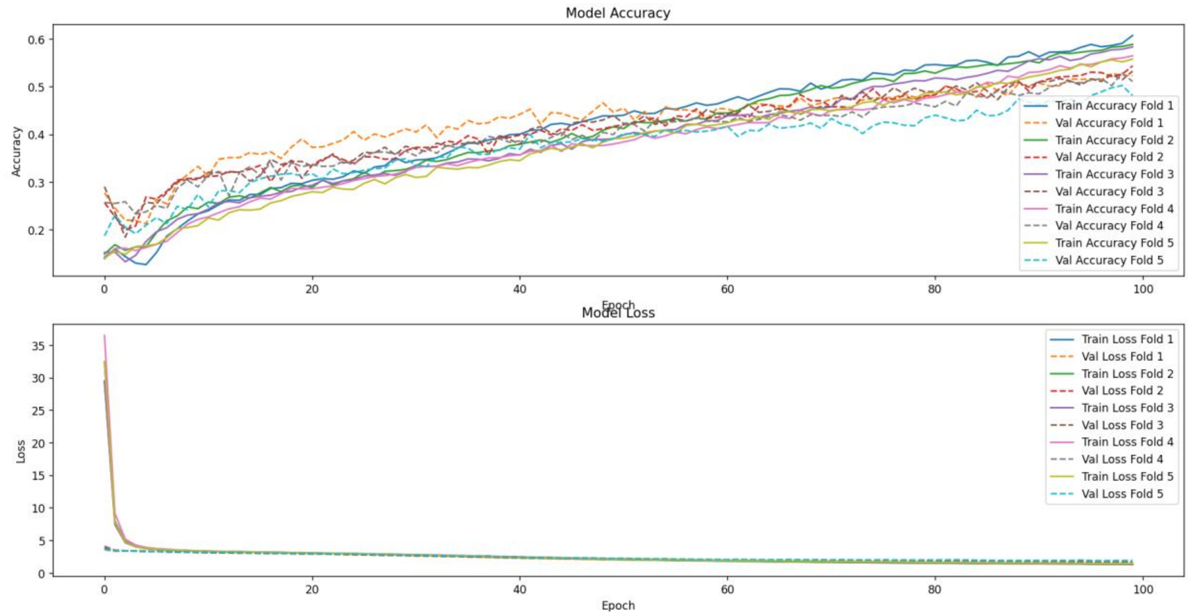
Fold 4:

```
Epoch 90/100
250/250 [==============================] - 4s 16ms/step - loss: 1.4663 - accuracy: 0.5340 - val_loss: 1.7460 - val_accuracy: 0.4957
Epoch 91/100
250/250 [==============================] - 4s 15ms/step - loss: 1.4447 - accuracy: 0.5434 - val_loss: 1.7055 - val_accuracy: 0.5063
Epoch 92/100
250/250 [==============================] - 4s 15ms/step - loss: 1.4449 - accuracy: 0.5458 - val_loss: 1.7100 - val_accuracy: 0.5138
Epoch 93/100
250/250 [==============================] - 4s 15ms/step - loss: 1.4273 - accuracy: 0.5523 - val_loss: 1.7013 - val_accuracy: 0.5078
Epoch 94/100
250/250 [==============================] - 4s 15ms/step - loss: 1.4100 - accuracy: 0.5566 - val_loss: 1.7210 - val_accuracy: 0.5158
Epoch 95/100
250/250 [==============================] - 4s 15ms/step - loss: 1.4080 - accuracy: 0.5580 - val_loss: 1.7179 - val_accuracy: 0.5173
Epoch 96/100
250/250 [==============================] - 4s 15ms/step - loss: 1.3831 - accuracy: 0.5647 - val_loss: 1.7013 - val_accuracy: 0.5123
Epoch 97/100
250/250 [==============================] - 4s 15ms/step - loss: 1.3831 - accuracy: 0.5693 - val_loss: 1.7504 - val_accuracy: 0.5018
Epoch 98/100
250/250 [==============================] - 4s 15ms/step - loss: 1.3553 - accuracy: 0.5799 - val_loss: 1.7117 - val_accuracy: 0.5168
Epoch 99/100
250/250 [==============================] - 4s 15ms/step - loss: 1.3626 - accuracy: 0.5803 - val_loss: 1.6544 - val_accuracy: 0.5128
Epoch 100/100
250/250 [==============================] - 4s 15ms/step - loss: 1.3391 - accuracy: 0.5864 - val_loss: 1.6668 - val_accuracy: 0.5238
63/63 [==============================] - 1s 6ms/step
```

Fold 5:

```
Epoch 90/100
250/250 [==============================] - 2s 9ms/step - loss: 1.5343 - accuracy: 0.5198 - val_loss: 1.9284 - val_accuracy: 0.4677
Epoch 91/100
250/250 [==============================] - 2s 9ms/step - loss: 1.5020 - accuracy: 0.5237 - val_loss: 1.9150 - val_accuracy: 0.4627
Epoch 92/100
250/250 [==============================] - 2s 10ms/step - loss: 1.5147 - accuracy: 0.5281 - val_loss: 1.9102 - val_accuracy: 0.4702
Epoch 93/100
250/250 [==============================] - 2s 9ms/step - loss: 1.5005 - accuracy: 0.5340 - val_loss: 1.9359 - val_accuracy: 0.4682
Epoch 94/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4673 - accuracy: 0.5364 - val_loss: 1.9028 - val_accuracy: 0.4742
Epoch 95/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4526 - accuracy: 0.5478 - val_loss: 1.9871 - val_accuracy: 0.4637
Epoch 96/100
250/250 [==============================] - 2s 9ms/step - loss: 1.4619 - accuracy: 0.5412 - val_loss: 1.9068 - val_accuracy: 0.4812
Epoch 97/100
250/250 [==============================] - 3s 10ms/step - loss: 1.4339 - accuracy: 0.5516 - val_loss: 1.8834 - val_accuracy: 0.4892
Epoch 98/100
250/250 [==============================] - 3s 10ms/step - loss: 1.4220 - accuracy: 0.5560 - val_loss: 1.8991 - val_accuracy: 0.4977
Epoch 99/100
250/250 [==============================] - 3s 10ms/step - loss: 1.4389 - accuracy: 0.5518 - val_loss: 1.9058 - val_accuracy: 0.5023
Epoch 100/100
250/250 [==============================] - 2s 10ms/step - loss: 1.4236 - accuracy: 0.5576 - val_loss: 1.9062 - val_accuracy: 0.4812
63/63 [==============================] - 0s 3ms/step
```

```
Average Accuracy across the folds: 0.5140178561210632
Average Loss across the folds: 1.7022186517715454
```



Averaged Classification Report:

```
Precision:
6: 0.77
2: 0.32
1: 0.72
4: 0.58
5: 0.38
9: 0.40
3: 0.53
0: 0.43
7: 0.76
8: 0.50
macro avg: 0.54
weighted avg: 0.54
```

```
Recall:
6: 0.75
2: 0.52
1: 0.80
4: 0.40
5: 0.39
9: 0.32
3: 0.39
0: 0.36
7: 0.77
8: 0.49
macro avg: 0.52
weighted avg: 0.52
```

```
F1-score:
6: 0.75
2: 0.39
1: 0.74
4: 0.47
5: 0.38
9: 0.35
3: 0.44
0: 0.38
7: 0.76
8: 0.49
macro avg: 0.52
weighted avg: 0.52
```

```
Accuracy:
accuracy: 0.52
```

Comparision with

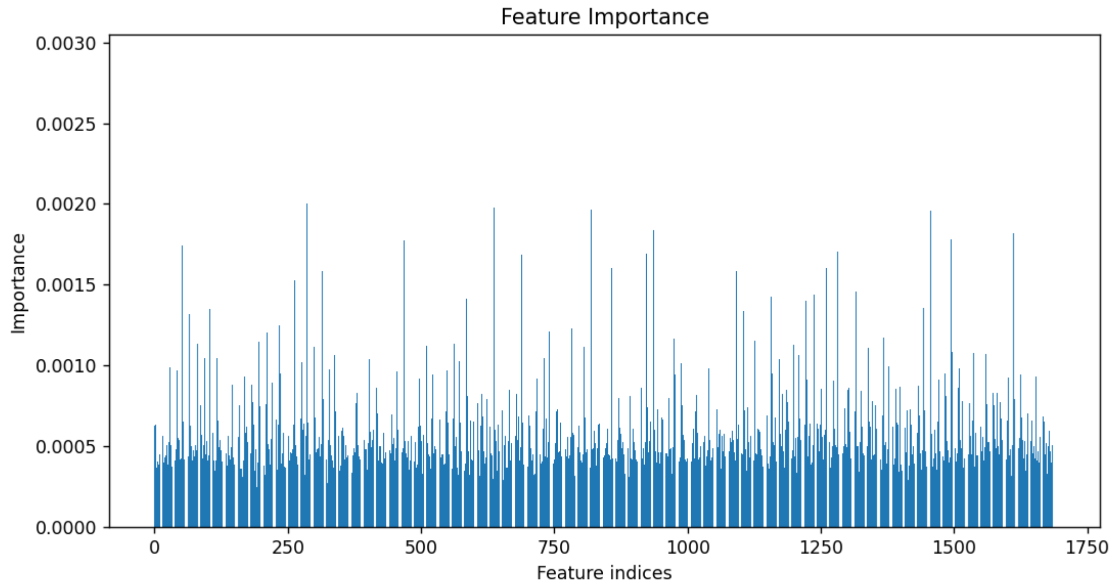
Random Forest

```
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.52      0.57       319
           1       0.83      0.92      0.87       308
           2       0.51      0.38      0.43       289
           3       0.43      0.41      0.42       298
           4       0.54      0.29      0.37       329
           5       0.57      0.70      0.63       282
           6       0.56      0.89      0.69       280
           7       0.46      0.85      0.60       283
           8       0.59      0.49      0.53       304
           9       0.46      0.25      0.32       304

    accuracy                           0.56      2996
   macro avg       0.56      0.57      0.54      2996
weighted avg       0.56      0.56      0.54      2996

Accuracy: 0.5614152202937249
```



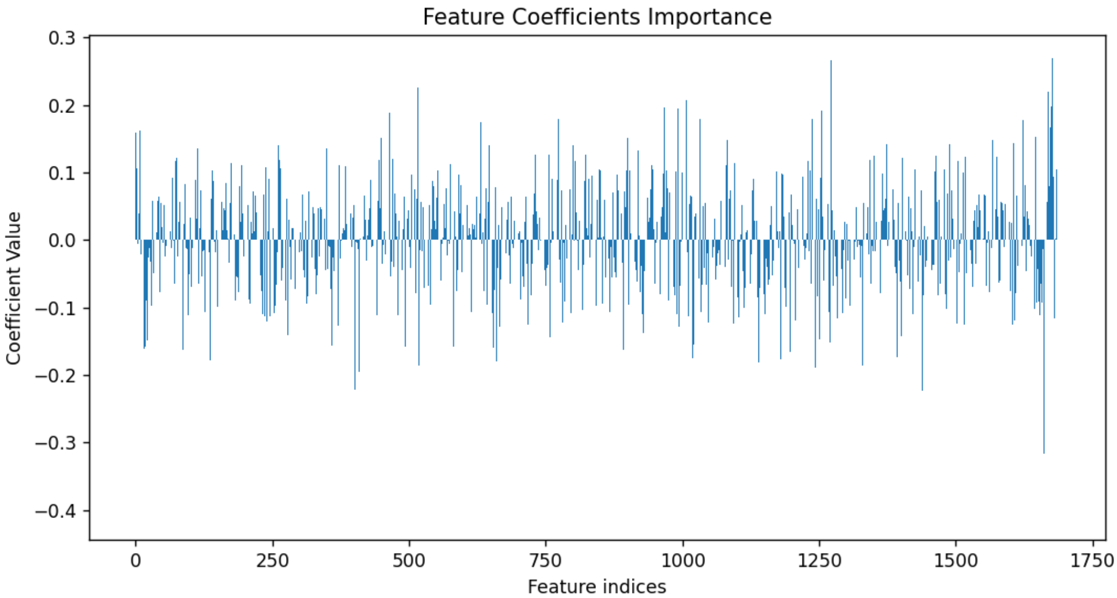
Logistic Regression:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.33      0.33       319
           1       0.69      0.79      0.74       308
           2       0.24      0.22      0.23       289
           3       0.20      0.16      0.18       298
           4       0.23      0.21      0.22       329
           5       0.36      0.39      0.37       282
           6       0.57      0.68      0.62       280
           7       0.53      0.53      0.53       283
           8       0.24      0.23      0.24       304
           9       0.21      0.21      0.21       304

    accuracy                           0.37      2996
   macro avg       0.36      0.37      0.37      2996
weighted avg       0.36      0.37      0.36      2996

Accuracy: 0.37016021361815754
```



Bagging:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.47      0.39       319
           1       0.72      0.84      0.78       308
           2       0.18      0.21      0.19       289
           3       0.28      0.27      0.28       298
           4       0.31      0.22      0.26       329
           5       0.45      0.42      0.43       282
           6       0.58      0.63      0.61       280
           7       0.50      0.61      0.55       283
           8       0.39      0.25      0.31       304
           9       0.21      0.14      0.17       304

    accuracy                           0.41      2996
   macro avg       0.39      0.41      0.40      2996
weighted avg       0.39      0.41      0.39      2996

Accuracy: 0.40520694259012013
```

Resources:

https://www.tensorflow.org/

https://www.tensorflow.org/guide/keras

https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification

https://ar5iv.labs.arxiv.org/html/2309.04861

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning