A Machine Learning Approach for Music Genre Classification

Yizhi Lu[†]

Project advisor: Hien Tran[‡]

Abstract.

This research aims to evaluate the effectiveness and efficiency of different machine learning (ML) algorithms in classifying music genres. In this paper, the ML algorithms that are considered include K-Nearest Neighbors (kNN), Perceptron Learning Algorithm (PLA), Multilayer Perceptron (MLP), and AdaBoost. To classify different genres of music, the ML algorithms use certain key features of music such as Beats Per Minute (BPM), key, mode, and lyrics. By comparing the accuracy of the algorithms, the paper will provide insight into the suitability of these algorithms for music genre classification tasks.

Key word. Classification, Music Genre, kNN, Perceptron, MLP, ANN, AdaBoost

1. Introduction.

In recent years, more music media and playback platforms have started to use content recommendation systems to attract users. The classification of genres of music is an important part of this task. Meanwhile, with the rise of machine learning, there is a growing interest in automating this process by using certain machine learning methods. However, the performance of several basic ML algorithms in classifying music genres remains underexplored. This research will investigate how well the kNN, Perceptron, MLP, and AdaBoost algorithms perform when applied to a certain dataset that contains musical features and their genre. The goal is to determine which machine learning algorithm is better suited for the classification of music genres.

As mentioned above, music genre classification is a crucial task for music platforms to improve their services. When it comes to a large amount of music, it becomes significantly important to adopt a highly automated method to perform this task. With the advent of machine learning in recent years, machine learning algorithms such as decision trees and Support Vector Machine (SVM) were used in [7]. More recently, studies have applied models such as the convolutional neural network (CNN) to the classification of music genres ([1] [3] [4]) and such models have been shown to have quite good efficiency ([2]). However, the accuracy of neural networks has been limited when it comes to different libraries of music [6].

Compared to previous studies on CNN models and other algorithms, fewer studies have focused on more simple algorithms such as kNN and Perceptron. This research will focus on such algorithms, explore their performance, and then discuss if they can function as well as other more complex models in music genre classification.

The goal of this research is as follows.

1. To compare the classification accuracy and computational efficiency of kNN, Perceptron, MLP, and AdaBoost when applied to music genre classification based on multiple kinds of feature.
2. To find the most suitable hyperparameters for certain algorithms, e.g., the value of k

[†]College of Computer Science and Technology, Zhejiang University, China (luyizhi1124@zju.edu.cn).

[‡]Department of Mathematics, North Carolina State University, Raleigh, NC, USA (tran@math.ncsu.edu).

in the kNN algorithm and the width and depth of the MLP algorithm.
3. Identify key insights and recommendations for selecting the most suitable algorithm for genre classification.
4. Analyze the importance of different features of music for genre classification and provide suggestions on feature selection.

The remainder of the paper is organized as follows. Section 2 details the dataset, Section 3 introduces the machine learning models utilized in this work; Section 4 presents the results of the machine learning models; Section 5 contains further discussions on the results; we concluded our work in Section 6 with some additional remarks.

2. Dataset.

2.1. Dataset Description. A dataset of music tracks labeled with their respective genres (e.g., pop, rock, classical) was used. The dataset is downloaded from *Kaggle* (www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre). It contains 50000 music samples (including corrupted data, which was removed). Each sample has 17 features and one genre (label).

The 17 features in the dataset are: *instance id*, *artist name*, *track name*, *popularity*, *acousticness*, *danceability*, *duration (ms)*, *energy*, *instrumentalness*, *key*, *liveness*, *loudness*, *mode*, *speechiness*, *tempo*, *obtained date*, and *valence*. A short definition of each feature of the dataset is given in Table 1. Before further processing, the *instance id* and *obtained date* features are removed from the dataset, as they do not exhibit any meaningful association with the musical genre labels. The remaining 15 features are then subjected to preprocessing and utilized for model training.

In this investigation, 10 different music genres are taken into account. They are *Alternative*, *Anime*, *Blues*, *Classical*, *Country*, *Electronic*, *Hip-Hop*, *Jazz*, *Rap*, and *Rock*.

Some examples chosen from the dataset are shown in Table 2. Note that only some of the features (shown in columns) are listed.

2.2. Data Preprocessing. Before preprocessing, the samples with corrupted data in the original set are removed. Corrupted data are defined as observations that contain anomalies, missing values, or implausible entries (e.g., numerical values marked as "?" or NaN, as well as negative durations). After this process, the size of the dataset was reduced to 40560.

2.2.1. Categorical Feature Encoding. Each sample in the dataset has 11 numerical features and 4 categorical features. Categorical features cannot be directly recognized by most machine learning models, so we need to convert them into numerical features by the encoding method.

In this paper, we use both *Label Encoding* and *One-Hot Encoding* to encode categorical features. An example is given in Table 3 to show the difference in the process between *Label Encoding* and *One-Hot Encoding*.

The basis of choosing between these two encoding methods is based on the number of categories of the feature and whether the feature contains comparable number information. *One-hot Encoding* will expand one categorical feature (with M categories) into M numerical features (with attributes 0 or 1), while *Label Encoding* will not expand the original feature number. Therefore, in order to avoid having too many features for training, only features

Table 1: Definition of features in the original dataset

Feature Name	Description	Type
instance id[1]	The unique ID for each sample	Numeric
artist name	Name of the singer(s)/artist(s)	Categorical
track name	Name of the song itself	Categorical
popularity	Measurement of sample's relative popularity	Numeric
acousticness	Measurement of to what extent the sample is acoustic	Numeric
danceability	Measurement of how danceable the sample is	Numeric
duration (ms)	Length of the sample	Numeric
energy	Measurement of how energetic the sample is	Numeric
instrumentalness	The proportion of instrumental accompaniment	Numeric
key	Sample's key name	Categorical
liveness	Measurement of sample's relative liveness	Numeric
loudness	Sample's standard loudness in LUFS[2]	Numeric
mode	Sample's mode name	Categorical
speechiness	The proportion of lyrics in the sample	Numeric
tempo	Sample's tempo in BPM[3]	Numeric
obtained date[1]	The date when the sample is included in the dataset	Categorical
valence	The measurement of the valence of the sample	Numeric

[1] Not used for training.
[2] LUFS (Loudness Units Full Scale) is a standard for measuring perceived loudness in audio, designed to reflect how the human ear perceives volume across different frequencies.
[3] BPM (Beats Per Minute) is a unit of measurement that indicates the tempo of a piece of music, representing the number of beats occurring in one minute.

Table 2: Example of some features in the original dataset

popularity	*danceability*	*duration*	*key*	*mode*	*music genre*
45	0.212	648827	G#	Major	Jazz
30	0.345	257829	G	Major	Electronic
50	0.747	251588	C#	Minor	Rap
54	0.675	274253	D#	Major	Rock
57	0.667	241040	E	Minor	Hip-Hop
numerical feature			**categorical feature**		**label**

with a small number of categories will be considered to be encoded by *One-Hot Encoding* (the number of categories of each categorical feature is shown in Table 4). For categorical features with comparable magnitudes, *Label Encoding* assigns values from 1 to N (assuming that there are N categories) after ordering them, thus preserving the relative order among categories. In contrast, *One-Hot Encoding* cannot achieve this. Therefore, for features with comparable numerical values, we will give priority to using Label Encoding.

For our dataset, *artist name* and *track name* were encoded by *Label Encoding*, while *key*

Table 3: An example of *Label* and *One-Hot* encoding

Before	After			
Categorical	**Label**	**One-Hot**		
A	1	1	0	0
B	2	0	1	0
C	3	0	0	1
A	1	1	0	0
C	3	0	0	1

and *mode* were encoded by *One-Hot Encoding* (see Table 4).

Table 4: Preprocessing categorical features

Feature	Category number	Encoding method	Reason
artist name	6363	Label	Too many categories
track name	34680	Label	Too many categories
key	12	One-Hot	Not ordered structure
mode	2	One-Hot	Not ordered structure

After this process, the number of features in the dataset will increase to 27. The categorical data will all become numerical data.

2.2.2. Normalization. The numerical features in the original dataset do not consistently follow approximately normal distributions. Figure 1 illustrates several representative cases. To improve data usability, normalization is applied prior to model training.

One commonly used method of normalization is *Z-score* normalization, defined by

$$x' = \frac{x - \mu}{\sigma}, \tag{2.1}$$

where x' is the normalized value, μ is the average value, and σ is the standard deviation. However, it does not perform well on not normally distributed data (such as this dataset). Therefore, we use the *Min-Max* normalization, which is defined by

$$x' = \frac{x - min}{max - min}, \tag{2.2}$$

where *min* and *max* represent the minimum and maximum values in the dataset, respectively.

This normalization method keeps the shape of the original distribution, while mapping all the values to $[0, 1]$. By doing this, we can maintain the space structure of the feature, which may affect the performance of the kNN model, where we need to calculate the *distance* between samples based on the value of the feature.

The *mean value* and *standard deviation* of all 11 numerical features after *Min-Max* normalization are shown in Figure 2.

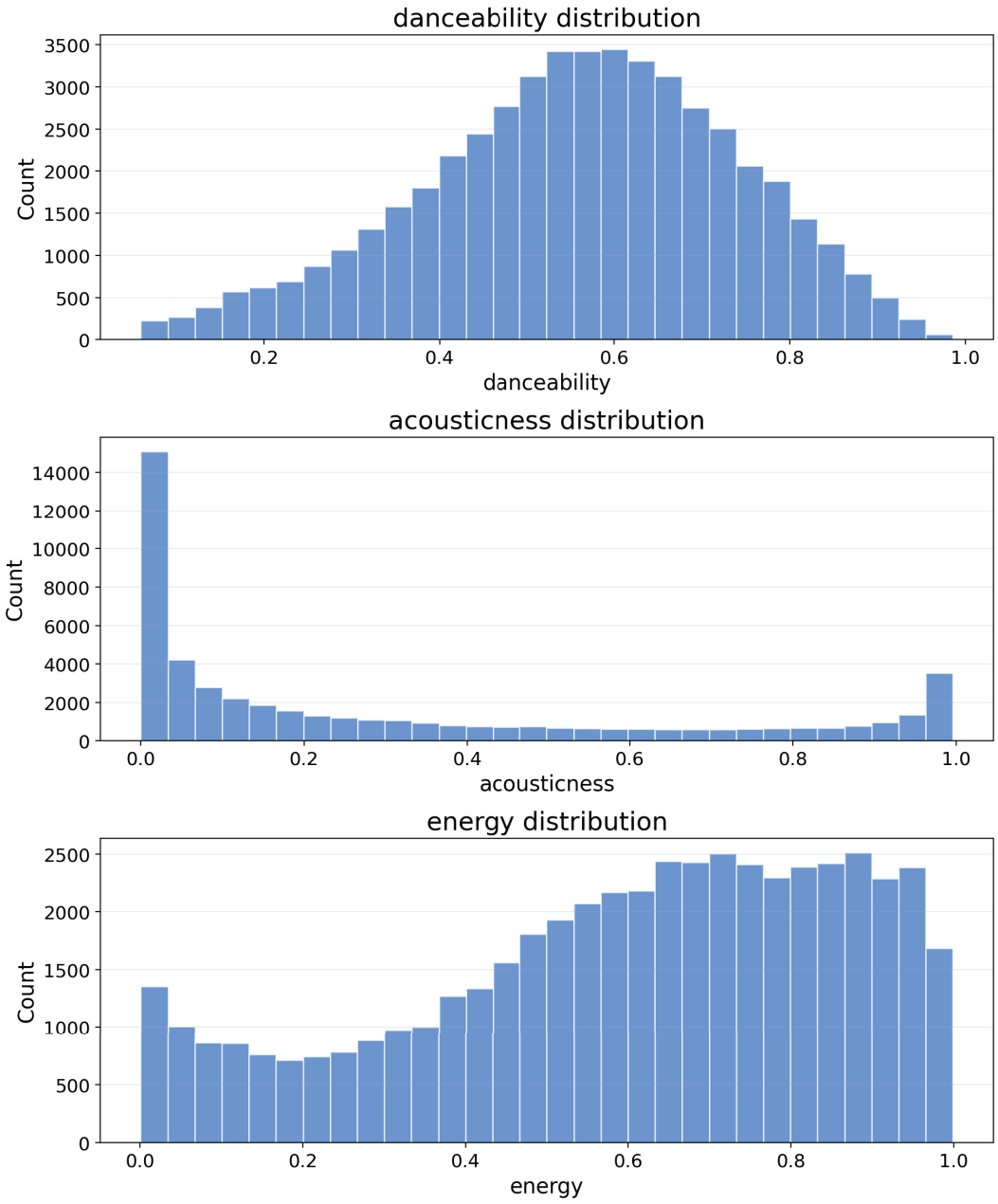

Figure 1: Original distribution of some numerical features in the dataset

2.2.3. Dataset Division. The dataset (of size 40560) was randomly divided into a training set (of size 32448, 80% of the original size) and a testing set (of size 8112, 20% of the original size).

Now, for simplicity of notation, assuming that we have N samples, i.e., a total of N music

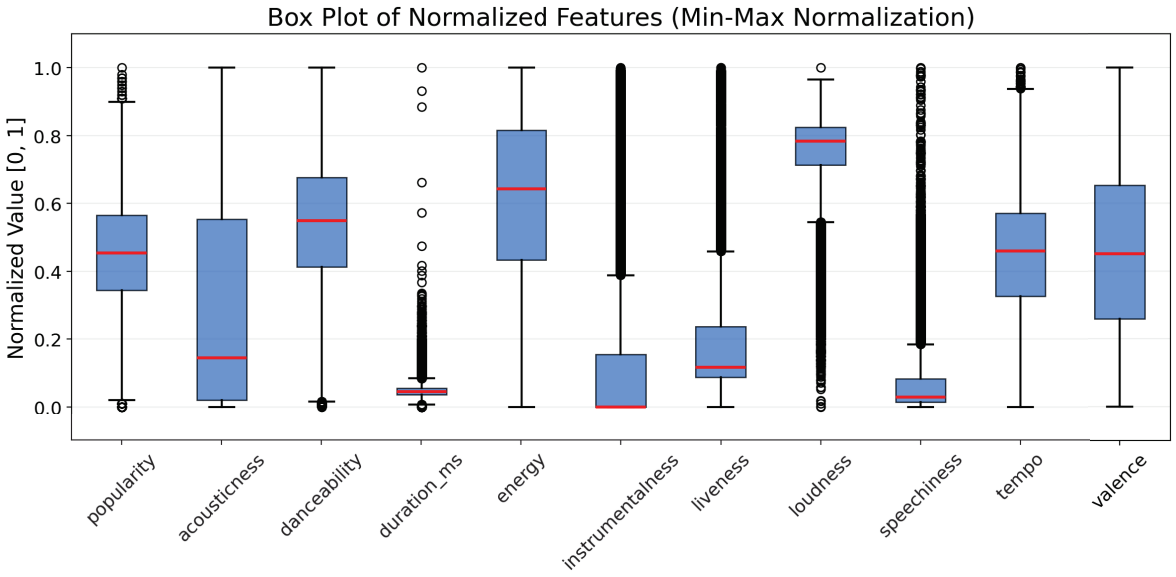

Figure 2: Boxplot of numerical features after normalization

samples in the dataset, let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^m$ (m is 27 in this case) denote the m-dimensional vector of features of one certain music sample, and let $y_1, \ldots, y_N \in \{1, 2, \ldots, 10\}$ denote the labels *music_genre*. For $t \in \{1, 2, \ldots, N\}$, let $\hat{y}_t$ be the predicted value of y_t by the machine learning models discussed in the following sections.

2.3. Binary and Multi-class Labeling. Among the models used in this research, some can handle only binary classification problems (e.g., Perceptron), and some can handle both binary and multi-class classification problems (e.g., kNN). Therefore, we will prepare two sets of label data to train different models. One is the set of origin labels, with values ranging from 1 to 10 since we have 10 classes in total, and a *One-Hot* version of the original set of labels.

For models that can only solve binary classification problems, we will utilize 2 methods enabling them to do multi-class classification tasks, which are *One vs. All (OvA)* and *One vs. One (OvO)*.

For the *One vs. All (OvA)* method, a total of P individual classifiers are trained, assuming the existence of P distinct classes in the dataset. Each classifier is trained to determine whether a given sample belongs to a specific class L_i, where $i \in \{1, \ldots, P\}$. Concretely, each classifier uses one column of the binary-encoded label matrix corresponding to the presence or absence of class L_i as its target output. Through this formulation, the original P-class classification problem is decomposed into P independent binary classification tasks, each of which can be effectively solved using the perceptron algorithm.

During the prediction stage, for each test data, all P models will give their scores (the actual way of calculating the score depends on the model type) and generally the class represented by the model with the highest score will be the final decision.

The One-vs-One (OvO) strategy addresses multi-class classification by decomposing it into a series of binary classification tasks. In contrast to the One-vs-All (OvA) approach, which constructs one classifier per class against all remaining classes, OvO trains a distinct classifier for each possible pair of classes. Given P classes, this results in a total of $\frac{P(P-1)}{2}$ binary classifiers. Each classifier $P_{i,j}$ is trained exclusively on the subset of data corresponding to classes i and j, and its decision function assigns a test instance to either class i or class j.

When predicting the label of a certain test data, all $\frac{P(P-1)}{2}$ models shall vote for one out of two classes represented by itself. Then the class with the most votes will be the final decision on the label prediction.

3. Machine Learning Models.

3.1. kNN. The k-Nearest Neighbors (kNN) algorithm is a non-parametric, instance-based learning method commonly used for classification tasks. Given a test data point, the algorithm identifies the K number of closest data points from the train dataset in the feature space (according to a certain distance metric, which will be discussed later in this section) and assigns the label that occurs the most frequently among these neighbors. The kNN algorithm has the ability to naturally perform multi-class classification tasks and serves as a widely adopted baseline in classification research.

Since music genre prediction is a multi-class classification problem and the kNN model can directly operate on multi-class datasets, it is intuitive to train a single kNN model on the entire training set. In addition, we explore the use of *One-vs-One (OvO)* and *One-vs-All (OvA)* schemes in combination with kNN as binary classifiers to investigate whether such strategies can improve prediction accuracy.

Generally, the default value of K for the kNN model is approximately equal to $\sqrt{N}$, where N is the size of the training dataset. Notating it as K_0, we have

$$K_0 = 2 \times \lfloor \frac{\lfloor \sqrt{N} \rfloor}{2} \rfloor + 1, \tag{3.1}$$

which ensures that K_0 is odd. With $N = 32448$ we have $K_0 = 181$.

However, the best choice of K may depend on different applications. To decide on the best choice of K for this research, we started with some intuitive guesses and then discussed in what range the value of K is that can improve the accuracy of the kNN model to the greatest extent.

We make sure all Ks are odd, which guaranties that there will be no draw when the kNN model predicts the binary label of the test data.

Four series of K values will be studied. As the value of N is 32448 and K_0 is 181, the ranges of K are:

- **Small K ($K \in (1, 30)$):** When K is small, the classifier relies on very few neighbors to determine the label of a test sample. This can result in a highly flexible model that can capture local variations, but is also prone to overfitting and noise sensitivity. (For example, when $K = 1$, the prediction is entirely dependent on the nearest neighbor, which can easily misclassify if the neighbor is an outlier)

- **Medium K ($K \in (30, 300)$):** This range includes the heuristic choice K_0. Here, the classifier balances between local detail and global stability. The model is expected to generalize well, as the influence of individual noisy points diminishes while still preserving enough locality to distinguish between classes.
- **Large K ($K \in (300, 3000)$):** With higher values of K, the prediction increasingly reflects global patterns in the dataset rather than local neighborhoods. This reduces variance, but may lead to underfitting, as subtle class boundaries are smoothed out.
- **Extremely large K ($K \in (3000, 30000)$):** Here, the classifier essentially considers a substantial fraction of the training set for every prediction. As K approaches N, the algorithm tends toward always predicting the majority class, leading to severe underfitting and poor performance in multi-class problems. This setting is primarily of theoretical interest to illustrate the degeneration of kNN at very large K.

We decide on the scale of the value of K based on comparing with K_0, calculated by (3.1). The average ratio $\frac{K}{K_0}$ for the 4 different ranges from *Small* to *Extremely large* are approximately around 0.1, 1, 10 and 100 respectively. The purpose of doing this is to study the proper order of magnitude of K, relative to N.

The research will first apply different K to the kNN multi-class classifier model and discuss which ones (K on what scale) are the most suitable for this task. The performance of each K is decided by its accuracy in predicting the testing dataset. In addition, *OvA* and *OvO* will be tested in the selected Ks and their performance will be compared.

Let M be the size of the testing dataset (8112 in this case), then for $t \in \{1, \dots, M\}$, $\boldsymbol{x}'_t$ is the current test sample which needs to be classified by the kNN model. What the model will do is calculate the *distance* between $\boldsymbol{x}_t$ and every data point in the training dataset.

There are three commonly used distance measures in the kNN algorithm: L_1, L_2, and *Mahalanobis* distance. Suppose that we have two samples $\boldsymbol{x}, \boldsymbol{y}$ from the dataset and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m, \boldsymbol{x} = (x_1, \dots, x_m), \boldsymbol{y} = (y_1, \dots, y_m)$; the distances between $\boldsymbol{x}$ and $\boldsymbol{y}$ are defined respectively by

$$D_{L_1}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i| \tag{3.2}$$

$$D_{L_2}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{3.3}$$

$$D_{\text{Mahal}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^\top \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{y})}, \tag{3.4}$$

where Σ is the covariance matrix of the dataset.

To show the difference in performance when using these three distances, we implement three straightforward multi-class kNN variants and apply them to the dataset before and after normalization, then compare their prediction accuracy.

For the remainder of this research, we use the distance formula with the best performance in the previous stage (which turns out to be L_1 distance; see Section 4).

For each sample $\boldsymbol{x}'_t$ in the testing dataset, we calculate the *distance* between it and all training samples and then find the K training samples that are closest to it (the K-nearest

samples). The model then checks the labels of these K samples to decide the label of $\boldsymbol{x}'_t$. For a multi-class kNN model, we choose the majority label among the K-nearest samples as the final decision. This process is shown in Algorithm 3.1.

Algorithm 3.1 kNN with OvA method

```
Define the value of K
Define the covariance matrix of the training data matrix as Σ.
for i in 1 → M do
  for j in 1 → N do
    Calculate D_{x'_i → x_j}
  end for
  Get the K nearest data points x_{n_1}, ..., x_{n_K}.
  Find the majority among the labels of x_{n_1}, ..., x_{n_K}, notate as L_i
  Set ŷ_i to L_{t_0}.
end for
```

Based on the algorithm introduced above, a series of K with promising accuracy will be selected to be used in further studies on *OvA* and *OvO*.

In the *OvA* method, specifically, the score S_i of a test data $\boldsymbol{x}'_t$ with respect to a class $L_i, i \in \{1, \ldots, P\}$, is defined by

$$S_i = \frac{\text{Number of neighbors belong to } L_i}{K},$$

and L_k with the highest score ($k \in \{1, \ldots, P\}$) will be considered as the most-likely music genre of $\boldsymbol{x}'_t$, so the model will set $\hat{y}_t$ to L_k. This is used in the prediction stage of *OvA* method.

The performance of these different methods will be discussed in a later section.

3.2. Perceptron. The Perceptron Learning Algorithm is one of the earliest supervised learning algorithms for binary classification. It is a linear classifier that seeks to find a hyperplane that separates the data into two classes. Given a vector of features and a corresponding label, the perceptron computes a weighted linear combination of features and applies a bias to produce its prediction. The weight vector and bias are iteratively updated whenever a misclassification occurs (see equations (3.6) and (3.7)). This simple update mechanism allows the perceptron to progressively adjust its decision boundary to better separate the classes.

We have a total of 15 different features in this study. After the data preprocessing stage, non-numeric features have been turned into numeric features by *One-Hot encoding*, thus increasing the total number of the feature dimensions directly used in the model training stage. In this study, as mentioned above, the number of features after preprocessing is 27 and is denoted by S. We denote the k-th feature as $f_k, k \in \{1, \ldots, S\}$. For a sample in the training set $\boldsymbol{x}_i, i \in \{1, \ldots, N\}$, let $f_{i,j}, j \in \{1, \ldots, S\}$ be the notation of the numeric value of the j-th feature of $\boldsymbol{x}_i$. Similarly, $f_{t,j}, j \in \{1, \ldots, S\}$ is the corresponding value of $\boldsymbol{x}'_t$ in the testing dataset.

The perceptron model defines the *weight* vector $\vec{w}$ and a *bias* b. The k-th value of w_k is the weight of f_k. Before the training stage, we set the value E of the *epoch* to 1000. The *Sign*

Function, sgn, is used as the activation function, and, correspondingly, the value of the label matrix will be set to 1 or -1.

The Perceptron Learning Algorithm starts with a random guess of small values in the *weight* vector $\vec{w}$ and *bias* b within a range of $(-\frac{1}{2}, +\frac{1}{2})$. For each training sample $\boldsymbol{x}_i$ and its corresponding label y_i, the algorithm computes its prediction by the formula

$$\hat{y}_i = \text{sgn}(\vec{w} \cdot \boldsymbol{x}_i) + b. \tag{3.5}$$

If the prediction is not correct (that is, $\hat{y}_i \neq y_i$), the weights and bias are updated by

$$\vec{w}_{\text{new}} = \vec{w}_{\text{old}} + y_i \boldsymbol{x}_i \tag{3.6}$$

$$b_{\text{new}} = b_{\text{old}} + y_i \tag{3.7}$$

This process is repeated through the training samples until either there are no misclassifications or until the user specified value of *epoch* is reached.

The Algorithm 3.2 shows the whole process of the Perceptron Learning Algorithm applied in this research.

Algorithm 3.2 Perceptron Learning Algorithm (ACC denotes the accuracy)

```
Define E, i.e., the number of epochs.
for t in 1 → P do
  for i in 1 → E do
    ACC_max = 0
    for 1 → N do
      Update w_i and b_i by (3.6)-(3.7).
    end for
    Calculate ACC_i using current w_i and b_i.
    if ACC_i > ACC_max then
      ACC_max = ACC_i
      w = w_i and b = b_i
    end if
  end for
  for i in 1 → M do
    Calculate F_{t,i} by (3.8).
  end for
end for
```

It is apparent that only one Perceptron model can perform a binary classification task. Therefore in this part of the study, the two different methods *One vs. All (OvA)* and *One vs. One (OvO)* will be applied to break down the multi-classification task, similar to what we have done with kNN model.

For *OvA* method, we define the *score* of a test point $\boldsymbol{x}'_t$ in the testing dataset by

$$F_t = \sum_{i=1}^{S} (w_i \cdot f_{t,i}) - b_i, \tag{3.8}$$

where w_i and b_i are defined by the current perceptron model. This *score* is used to decide the predict label of the test point $\boldsymbol{x}'_t$ when using the *OvA* method. For the k-th model and the test data point $\boldsymbol{x}'_t$, let us denote its output by $F_{k,t}$. After all outputs have been calculated, define k_0 as the value of k that maximizes $F_{k,t}$ for each $\boldsymbol{x}'_t$ in the testing dataset. Then $\hat{y}_t$ will be set to L_{k_0}.

The structure of *the One vs. One (OvO)* method is the same as the one introduced above.

Finally, we calculate the accuracy of the perceptron learning algorithm by comparing each $\hat{y}_i$ and y_i for both *OvA* and *OvO* and compare between these two methods.

3.3. MLP. We applied a Multi-layer Perceptron (MLP) (or Artificial Neural Network (ANN)) model in this research. To explore the suitable number of hidden layers, we will train a series of fully connected neural network (FCANN) models with different number of hidden layers. Models with depth 2, 3, 4, and 5 will be tested (with, respectively, 1, 2, 3 and 4 hidden layers and 1 output layer). Apart from that, some different activation functions will also be applied to our models, and their performance will be observed. We will also try a series of different numbers of neurons for the hidden layer in the model with depth 2. The performance of each model will be assessed later in the section.

To control variables, we set the default width of other models to be 20 (means 20 neurons in each hidden layer) and set 10 neurons in the output layer (since we have 10 classes in total). The default activation function for the hidden layer is *ReLU*. We use *Softmax* as a transfer function to the last layer in order to normalize the output. On top of that, we use *mean square error (MSE)* as the *loss* function to assess the model's performance, which will be discussed in more detail later.

For each model, we use a 10-fold cross validation for validation checking during the training stage. The model will stop training at the epoch when it fails to improve on the validation set for 6 consecutive epochs.

We will take the depth model 2 as an example to explain the MLP algorithm. Let $n_k, (k = 1, 2)$ be the number of neurons in the k-th layer.

For $i = 1, \ldots, n_k$, let $h_i^{(k)}$ be the output of the i-th neuron in the k-th hidden layer and let $b_i^{(k)}$ be its bias. Denote the weights of x_j $(j = 1 \ldots, p)$ in the i-th neuron of the k-th hidden layer by $w_{ij}^{(k)}$ and let $\vec{w}_i^{(k)} = \left(w_{i1}^{(k)}, \ldots, w_{ip}^{(k)}\right)^\top$.

Assuming φ is the activation function of the layer, we have $h_i^{(k)}$ being defined by

$$h_i^{(k)} = \varphi\left(\sum_{j=1}^{p} w_{ij}^{(k)} x_j + b_i^{(k)}\right) = \varphi\left(\boldsymbol{x}^\top \boldsymbol{w}_i^{(k)}\right) + b_i^{(k)}. \tag{3.9}$$

For the depth model 2, a series of different activation functions will be adopted to the hidden layer and their performance will be assessed. The following functions will be tested:

- ReLU: $\mathrm{ReLU}(x) = \max(0, x)$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

We do not modify the value in the input layer (first layer) and we use *Softmax* as the activation function in the output layer (last layer).

The aim of the MLP algorithm is to solve an optimization problem, which in this case is to minimize the *error* (also called *loss*) of the model. We calculate the *error* J using the method known as *mean square error (MSE)*, as mentioned above.

$$J = \frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t - y_t)^2. \tag{3.10}$$

The key of MLP is to find the relatively best weight choice to minimize J. At the beginning of the training process, $w_{ij}^{(k)}, i \in \{1, \ldots, n_k\}$, $j \in \{1, \ldots, p\}$, and $k \in \{1, 2\}$ are randomly initiated within a fixed range. Then in every iteration, where the maximum number of epochs is set to 1000 in practice, the algorithm uses the *Gradient Descent* to update $w_{ij}^{(k)}$ and check if it performs well enough. Let $\eta \in (0, 1)$ be the *learning rate* (set to 0.01 in practice), the *Gradient Descent* method can be written as

$$\boldsymbol{w}_i^{(k)} := \boldsymbol{w}_i^{(k)} - \eta \nabla_{\boldsymbol{w}_i^{(k)}} J, \tag{3.11}$$

where $\nabla_{\boldsymbol{w}_i^{(k)}} J$ is derived by calculating the gradient of J with respect to $\boldsymbol{w}_i^{(k)}$ using *Back Propagation (BackProp)*. In practice, we use *Levenberg-Marquardt backpropagation*, which uses first derivative approximation to the Hessian matrix to speed up the training progress, since the MLP model in this research is relatively small (for devices to memorize first derivatives). It is noted that cross entropy loss function is used mostly in classification problem with neural networks. However, this loss function is more suitable in batch training such as the stochastic gradient descent method. Higher-order optimization algorithms, such as the Levenberg-Marquardt algorithm, which uses the Hessian matrix, are better suited for quadratic loss function such as the MSE used in this paper [5].

After each update of $w_{ij}^{(k)}$, we use *cross validation* to check if the current output performs well enough. As we mentioned above, once the model passes the validation checks six times, the algorithm will stop updating and save the current model.

The whole process of MLP adopted in this research is shown in Algorithm 3.3.

3.4. AdaBoost. We also try to build a decision forest based on *Adaptive Boosting (AdaBoost)* to predict and classify music genre and assess its performance.

4. Results.

4.1. kNN.

4.1.1. Distance Measuring Methods. To show the difference between kNN using the distance formulas (3.2) (3.3) and (3.4), we implemented these kNN models on the training dataset and compared their prediction accuracy. We also compared models on data before and

Algorithm 3.3 MLP algorithm with two hidden layers.

Define the set of activation functions $\mathcal{F}$.
Define the maximal iteration T.
Define the threshold of convergence ε.
Define the learning rate $\eta \in (0, 1)$.
Initiate $w_{ij}^{(k)}(0)$ randomly.
for φ in $\mathcal{F}$ **do**
  for τ in $1 \to T$ **do**
    $e = 0$.
    for k in $2 \to 1$, step by -1 **do**
      for i in $1 \to n_k$ **do**
        Compute J using (3.10) with $w_{ij}^{(k)}(\tau - 1)$ for $k \in \{1, 2\}$, $j \in \{1, \ldots, p\}$, $i \in \{1, \ldots, n_k\}$.
        for s in $1 \to T$ **do**
          Update $\boldsymbol{w}_i^{(k)}(\tau - 1)$ by (3.11).
          Update J by (3.10).
        end for
        $\boldsymbol{w}_i^{(k)}(\tau) = \boldsymbol{w}_i^{(k)}(\tau - 1) - \eta \nabla_{\boldsymbol{w}_i^{(k)}(\tau-1)} J$.
        $e = e + \|\boldsymbol{w}_i^{(k)}(\tau) - \boldsymbol{w}_i^{(k)}(\tau - 1)\|_2$.
      end for
    end for
  end for
  if $e < \varepsilon$ **then**
    $w_{ij}^{(k)}(T) = w_{ij}^{(k)}(\tau)$, for $k \in \{1, 2\}$, $j \in \{1, \ldots, p\}$, $i \in \{1, \ldots, n_k\}$.
    break
  end if
end for
return $w_{ij}^{(k)}(T)$, $k \in \{1, 2\}$, $j \in \{1, \ldots, p\}$, $i \in \{1, \ldots, n_k\}$ for each $\varphi \in \mathcal{F}$.

after normalization to show how normalization affects model accuracy by affecting distance calculating.

The k used in this step is set to $k_0 = 181$ defined by (3.1) where the size of the training set is 32448. The results can be seen in Table 5.

Table 5: Performance of models with different distance measuring formulas

Formula	**Accuracy (%)**	
	Not normalized	*Normalized*
L_1	74.51	68.17
L_2	68.66	57.15
Mahalanobis	57.52	57.52

From the table, we can see that all models perform better on original data than normalized data, regardless of the distance measurement formula. In addition, the kNN model using the L_1 distance has the best accuracy compared to the L_2 and *Mahalanobis* distance. Therefore, we will use L_1 distance for the rest of this section to train our kNN models. The reason for this will be discussed in a later section.

4.1.2. Choices of K. In this paper, we apply a series of values of k on different scales and observe the prediction accuracy of the models on the testing set. Models with higher accuracy are considered to have better performance.

The accuracy of models with different values of k (*Small*, *Medium*, *Large* and *Extremely Large*, respectively) is shown in Figure 3. The value of k with the highest accuracy is depicted in the figure.

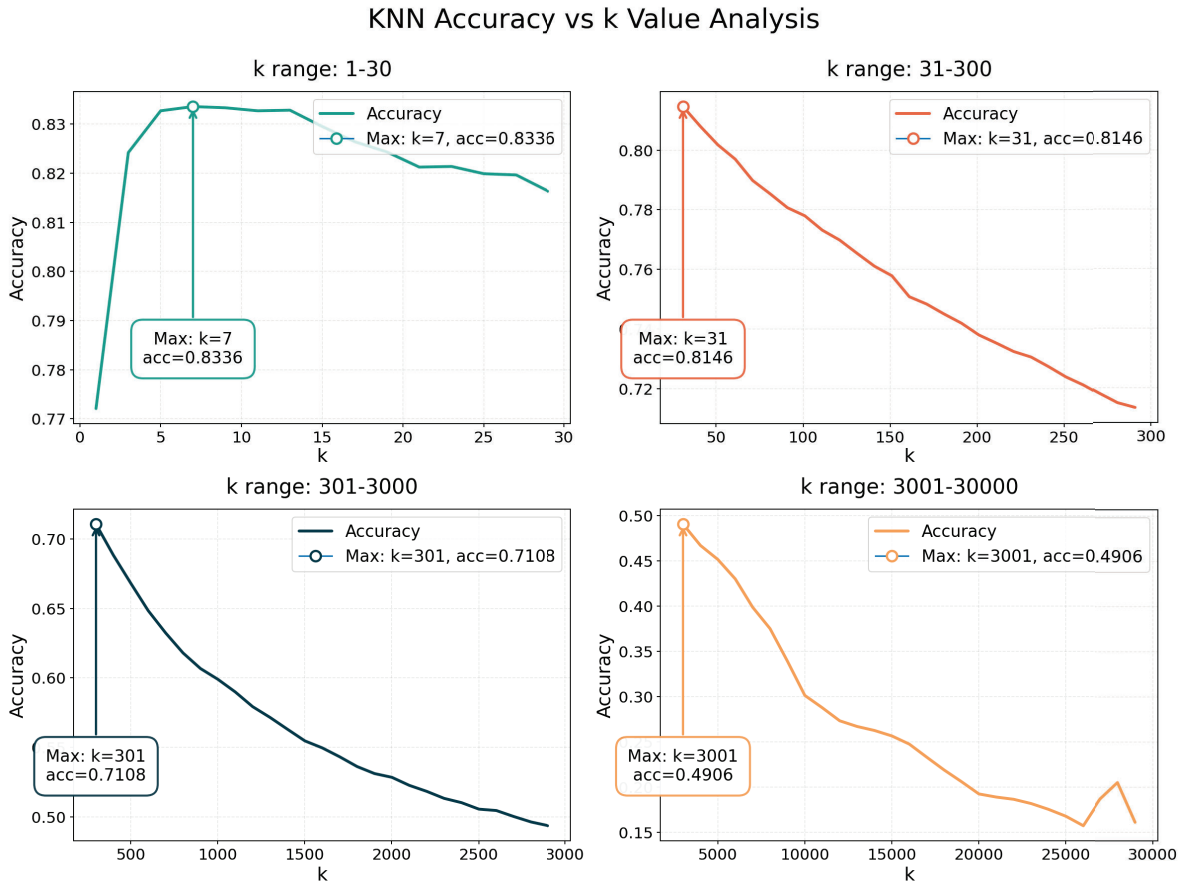

Figure 3: Accuracy of kNN models with different k value. The x-axis of each subplot is the value of k, the y-axis are accuracy, ranging from 0 to 1. Notice that the value range of the y-axis may differ between each subplot.

4.1.3. Binary and Multi-class kNN. From Figure 3 we can see that the value of k in the range of $[5, 13]$ has the relatively highest accuracy. We choose $K \in \{5, 7, 9, 11, 13\}$ to continue our research on the comparison between straightforward multi-class classification kNN models and binary classification kNN models with *OvA* and *OvO*.

The performance of these different types of kNN models' is still assessed by their average prediction accuracy in the testing dataset with the k value chosen above. The results are shown in Table 6. The name of a straight-forward multi-class classification kNN model is represented by *MC* for short.

Table 6: Accuracy of different kNN model types

(%)	*Model type*		
K	**MC**	**OvA**	**OvO**
5	83.27	83.26	83.28
7	83.36	83.37	83.25
9	83.33	83.36	83.01
11	83.27	83.28	82.92
13	83.28	83.25	82.80
avg.	83.30	83.30	83.05

4.2. Perceptron.

4.2.1. One vs. All Perceptron. First we use the *OvA* method to enable our Perceptron models to solve multi-classification tasks in this paper, which means that a total of 10 models are trained, each identifying a specific music genre. The accuracy of each model can be seen in Table 7 and Figure 4. Note that the accuracy shown below is just the accuracy of a single model regarding one label and does not represent the overall accuracy.

This indicates that Perceptron models work well in binary classification tasks (deciding whether or not one test data point belongs to a certain class), achieving an average accuracy of 92.73%.

However, in this paper, the average overall accuracy of the Perceptron Learning Algorithm with *OvA* is only at 58.81%, which is poor in reaching our target.

4.2.2. One vs. One Perceptron. To improve its overall performance, we instead tried the *OvO* method.

In the *OvO* method, a total of $\frac{P(P-1)}{2}$ models were trained, where P is the total count of classes. That is to say, we have a model for every single pair of different classes (train on a subset of training dataset consisting of data points belonging to only that pair of classes). In our case, the number of classes is 10, therefore, 45 models are trained. The accuracy of each model is shown in Table 8.

It can be seen that the mean performance of each model is slightly higher than that in *OvA* method. However, the overall accuracy of *OvO* is much better than that in *OvA*, which is around 77.19% (compared to that of *OvA*' 58.81%). This indicates that *the OvO* method is the better way to model the multi-class classification problem for this dataset.

Table 7: Accuracy of Perceptron models (OvA)

Model ID	Model Class	Accuracy (%)
1	Alternative	89.98
2	Anime	94.44
3	Blues	91.10
4	Classical	98.73
5	Country	90.18
6	Electronic	99.04
7	Hip-Hop	93.23
8	Jazz	90.11
9	Rap	88.88
10	Rock	91.57
Average Accuracy[1]: 92.73%		
Overall Accuracy[2]: 58.81%		

[1] Average Accuracy: The mean accuracy of all binary classifiers in their respective sub-tasks, calculated across all individual models.

[2] Overall Accuracy: The classification performance on the original dataset after integrating the binary classifiers using OvA strategy for multi-class classification.

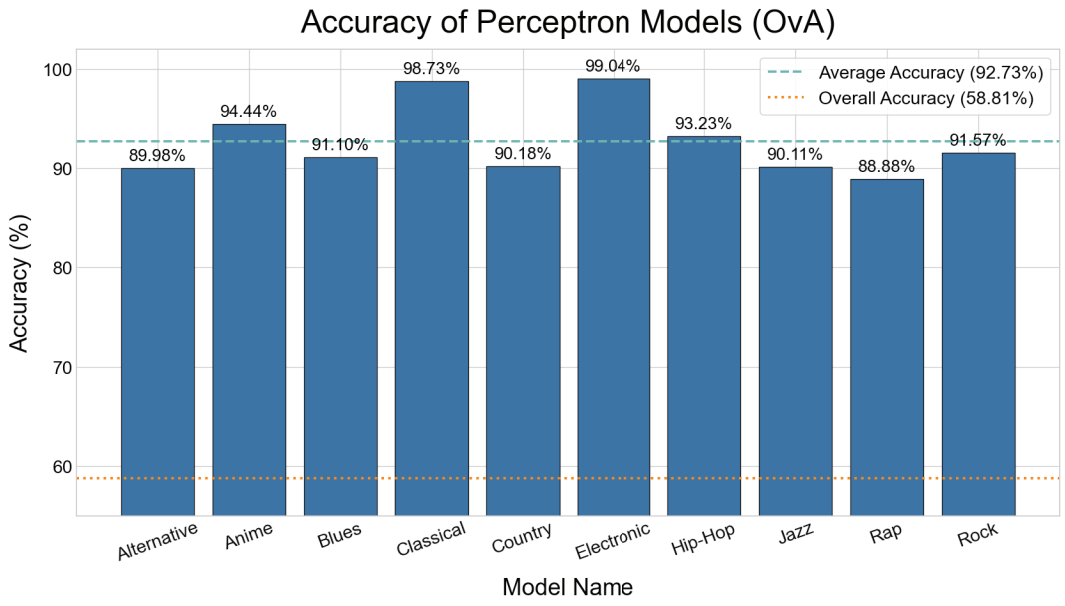

Figure 4: Accuracy of each Perceptron model in OvA

Table 8: Accuracy of Perceptron models (OvO)

Label ID	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	-	98.27	93.82	99.25	88.36	99.43	91.90	92.52	91.64	89.30
2	98.27	-	99.72	99.74	99.91	98.89	99.89	94.25	99.89	99.63
3	93.82	99.72	-	96.10	88.47	99.94	96.90	97.85	96.91	93.79
4	99.25	99.74	96.10	-	99.14	100.00	99.42	99.51	99.75	98.28
5	88.36	99.91	88.47	99.14	-	99.91	92.95	97.56	91.14	86.08
6	99.43	98.89	99.94	100.00	99.91	-	99.86	97.69	99.92	99.82
7	91.90	99.89	96.90	99.42	92.95	99.86	-	98.90	75.62	89.55
8	92.52	94.25	97.85	99.51	97.56	97.69	98.90	-	98.94	98.53
9	91.64	99.89	96.91	99.75	91.14	99.92	75.62	98.94	-	86.03
10	89.30	99.63	93.79	98.28	86.08	99.82	89.55	98.53	86.03	-
Average Accuracy[1]: 95.91%					**Overall Accuracy**[2]: 77.19%					

[1] Average Accuracy: The mean accuracy of all binary classifiers in their respective sub-tasks, calculated across all individual models.

[2] Overall Accuracy: The classification performance on the original dataset after integrating the binary classifiers using OvO strategy for multi-class classification.

4.2.3. Feature Weights. Since Perceptron models are linear classifiers and work by weighing each feature value, we can use it to analyze the importance of each feature in the dataset.

For each model, we calculate the *weight* for each feature. The higher the absolute value of the weight, the more important is the corresponding feature for genre classification. The categorical features *key* and *mode* have been translated into numeric form using the *One-Hot* method, which means that they are treated as multiple features in model training.

We can investigate the relative importance of different features by calculating the *average weight* for each feature by

$$\bar{w} = \frac{1}{P} \sum_{i=1}^{P} |w_i| \,,$$

where P is the number of total models that we have trained.

For *key* and *mode*, we adopt the overall average. The average *weight* of each feature can be seen in Table 9.

We will discuss the importance of different features in the subsequent section.

4.3. MLP. First, we compare the performance between models with depth 2, 3, 4 and 5. The corresponding activation functions of the first (input) and last (output) layers of both models are *purelin* and *softmax*, respectively. We will fix the activation function of the hidden layers of each model to *ReLU*.

Then we use the depth model 2 (with one hidden layer) to analyze the performance difference between different activation functions and different width (number of neurons in the hidden layer). The activation functions that will be tested in this research are *Tanh*,

Table 9: Average *weight* of each feature

Feature ID	1	2	3	4	5
Feature Name	*popularity*	*acousticness*	*danceability*	*duration*	*energy*
Weight	13.50	0.90	2.68	3.51	1.35
Feature ID	6	7	8	9	10
Feature Name	*instrumentalness*	*liveness*	*loudness*	*speechiness*	*tempo*
Weight	0.67	0.28	0.78	2.80	0.33
Feature ID	11	12	13	14	15
Feature Name	*valence*	*artist name*	*track name*	*key*	*mode*
Weight	0.20	2.68	1.74	1.30	7.85

Sigmoid, and *ReLU*.

For simplicity of notation, the conditions of all models used in this investigation are shown in Table 10 and each is numbered by an ID. All models have an input layer with width 27 because the feature dimension is 27 after preprocessing.

Table 10: Properties of MLP models

ID	**Depth**	**Structure (neuron numbers)**	**Activation function**
1	2	(27)-20-10	ReLU
2	2	(27)-20-10	Tanh
3	2	(27)-20-10	Sigmoid
4	2	(27)-10-10	ReLU
5	2	(27)-30-10	ReLU
6	2	(27)-40-10	ReLU
7	3	(27)-20-20-10	ReLU
8	4	(27)-20-20-20-10	ReLU
9	5	(27)-20-20-20-20-10	ReLU

We observe the training efficiency (by counting the epochs when the model meets validation check) and predict the accuracy of each model and assess their performance. The outcome is shown in Table 11. The data in the table can also be seen in Figure 5 to gain more intuitive insight into the variation of the model's performance with respect to the model *width* and *depth*.

4.4. AdaBoost. We want to find suitable hyperparameters that enable the model to converge fast (with fewer training cycles). To decide the best choices of *learning rate* and *learning cycles* of AdaBoost models, we trained a series of models with different hyperparameters and analyzed their performance (by their prediction accuracy on the validation set).

The first five models are trained to determine the proper order of magnitude of the learning rate. Candidates for the learning rates are given by $lr \in \{1 \times 10^{-5},\ 1 \times 10^{-4},\ 1 \times 10^{-3},\ 1 \times 10^{-2},\ 1 \times 10^{-1}\}$. In this step, the number of learning cycles is intuitively fixed to 300. The

Table 11: Performance of each MLP model

ID	Stopped value			Accuracy (%)
	Epoch	Loss	Gradient	
1	35	0.0141	3.78e-3	89.37
2	26	0.0137	5.86e-3	89.81
3	48	0.0135	3.81e-3	89.92
4	49	0.0154	1.19e-3	90.40
5	36	0.0135	6.22e-4	89.85
6	29	0.0131	2.95e-3	88.98
7	27	0.0144	8.92e-3	89.15
8	26	0.0122	1.04e-2	89.24
9	27	0.0123	8.47e-3	89.67

results are shown in Figure 6. It is clear that a higher learning rate tends to have better validation accuracy, so we can locate the new learning rate range in $(0.1, 1)$.

Next, five more models are trained to further determine learning rates in a more precise range. This time we reduced the number of learning cycles to 50 to test the convergence speed to different learning rates. The new candidates for learning rates are $lr \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The results are shown in Figure 7. A local maximum occurs at $lr = 0.5$, so it can be assumed that the best choice of learning rate falls around 0.5.

Finally, in order to make the final decision on both learning rates and learning cycles, other 30 models are trained on the current range of learning rates lr and learning cycles lc. Candidates for this step are $lr \in \{0.40,\ 0.45,\ 0.50,\ 0.55,\ 0.60\}$ and $lc \in \{500,\ 1000,\ 1500,\ 2000,\ 2500,\ 3000\}$. The results are shown in Figure 8, where the accuracy of the validation is presented by different colors on the heat map. From the data shown in the figure we can see that the best validation accuracy is 77.22%. Models with higher learning rates converge faster and all models converge at 1000 learning cycles and perform slightly worse with more training cycles.

5. Discussion. Based on the data we have obtained from the training and testing stage, we can analyze the comprehensive performance of models trained by different algorithms. Furthermore, we will try to gain insight into the classification of music genres based on the analysis.

In addition, we have tried a series of different hyperparameters for each model and compared their performance. In this section, we will analyze the results and provide some discussions.

5.1. Feature Importance. We use the weight of features produced by Perceptron models to analyze their importance. Generally speaking, the larger the absolute value of the weight, the more important the feature is. This is because all feature values are normalized in the data preprocessing stage, and therefore the same feature value with larger weight will have a more significant impact on the output of the Perceptron model, which means that the feature is more important than other.

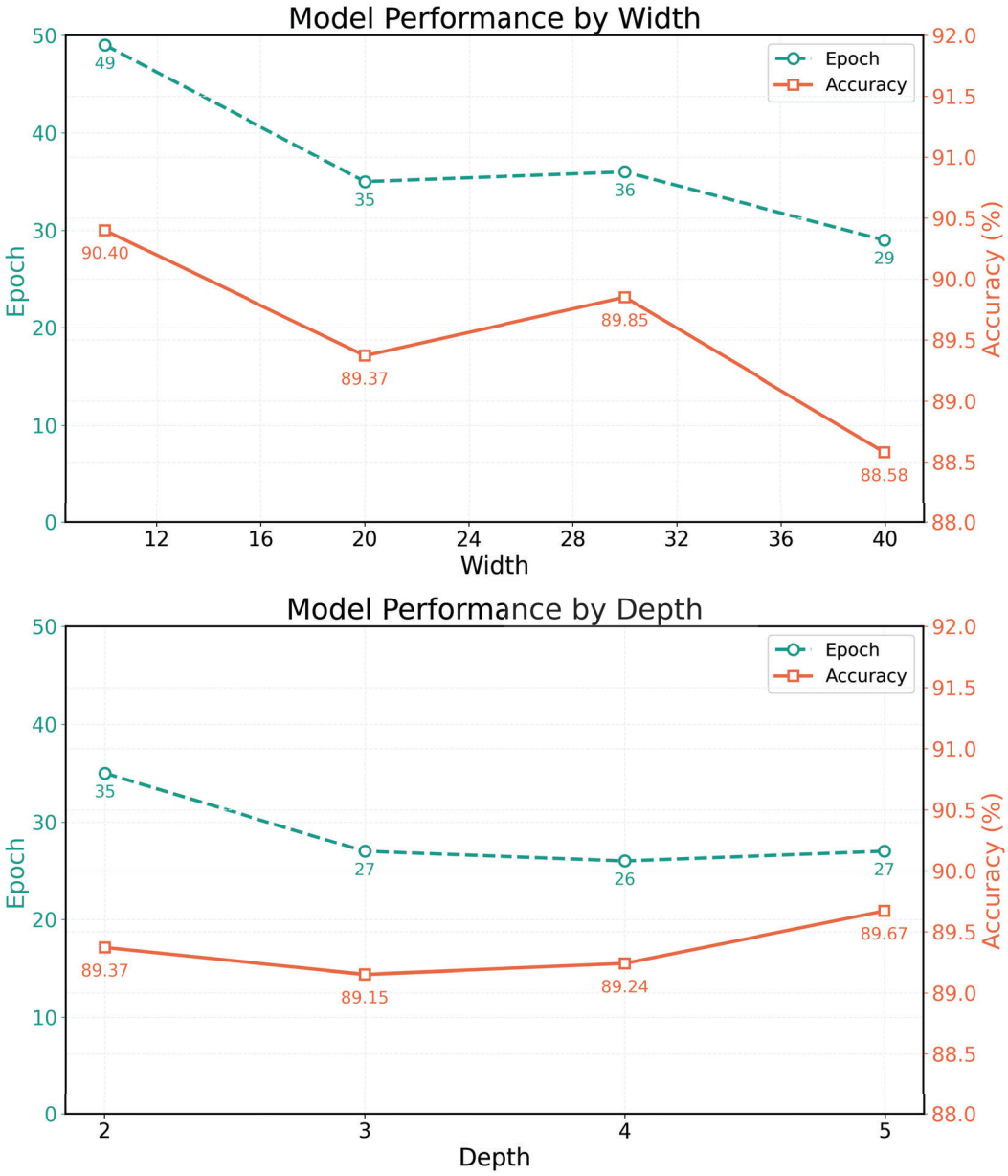

Figure 5: Model performance by *Width* and *Depth*

Intuitively, the following features should have a more significant effect on determining music genres: *popularity*, *acousticness*, *danceability*, *energy*, *instrumentalness*, *mode*, *speechiness*. This means that they are expected to have a larger *weight* (absolute value) in the outcome of the Perceptron.

From Table 9, we can see that some features have significantly greater weight than oth-

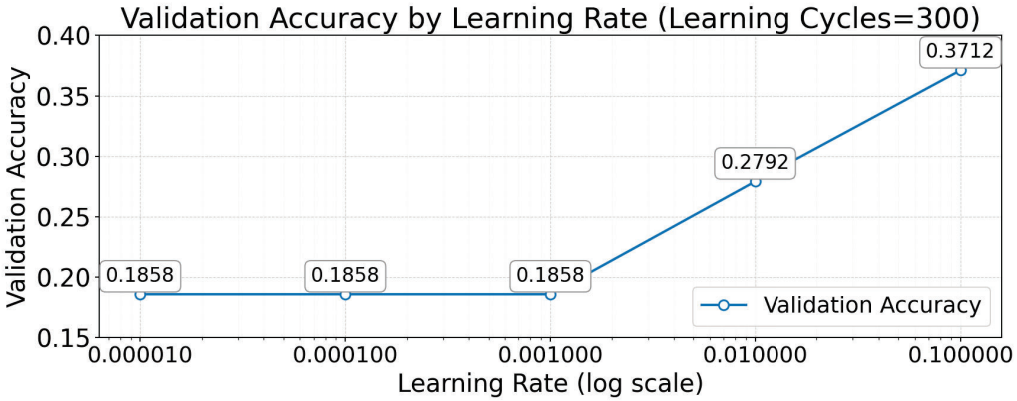

Figure 6: Validation accuracy by learning rate, fig 1.

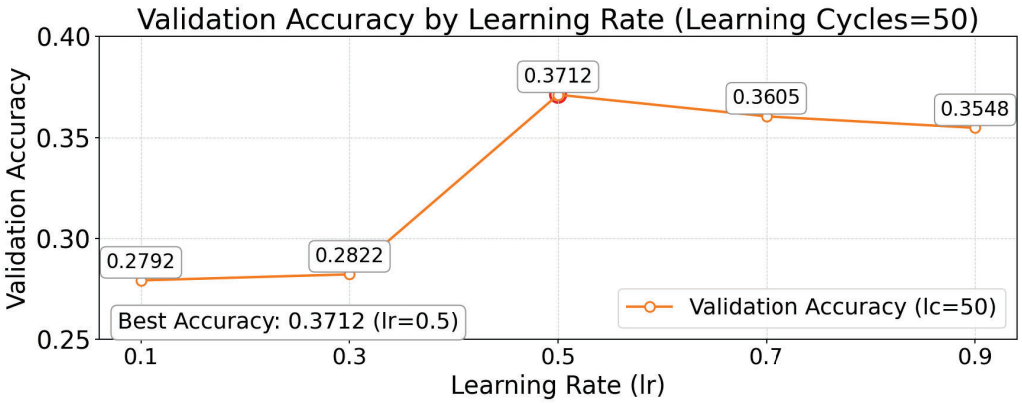

Figure 7: Validation accuracy by learning rate, fig 2.

ers. In Figure 9, we sort the features by their weight proportion, which can represent their *importance*. A pie chart is also drawn to show some of the most important features based on this analysis.

From the histogram in Figure 9 we can intuitively divide the features into two groups by their importance. Since there are 15 features taken into consideration, let $\frac{1}{15} \times 100\% = 6.7\%$ be a threshold, features whose importance percentage (weight proportion) is greater than 6.7% (whose original weight value is greater than 2.70, as shown in Figure 9) are considered *important features*, otherwise *ordinary features*. Consequently, we can divide the features into two subsets, *important features* and *ordinary features*.

The *important features* are the following.

1. popularity
2. mode

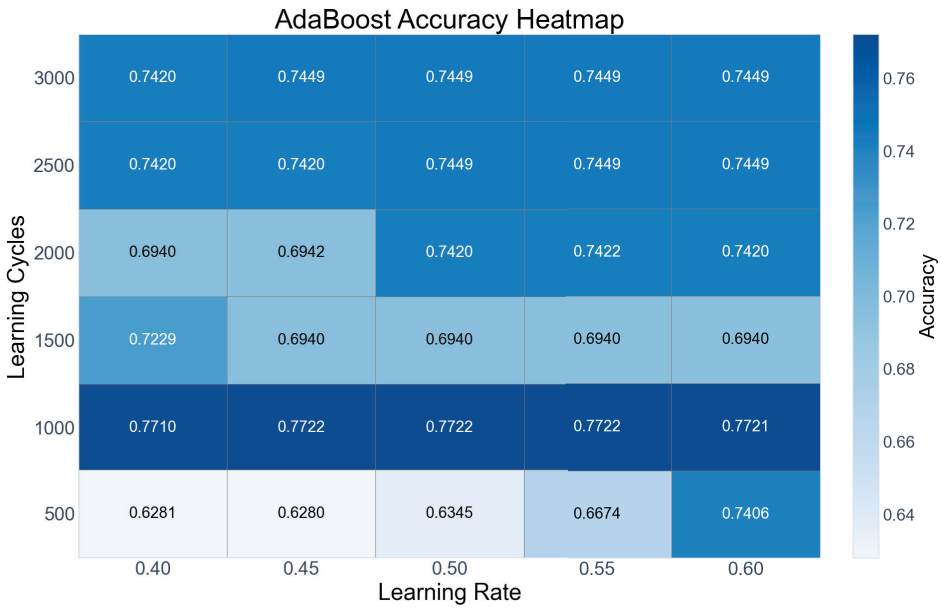

Figure 8: Heat map of validation accuracy

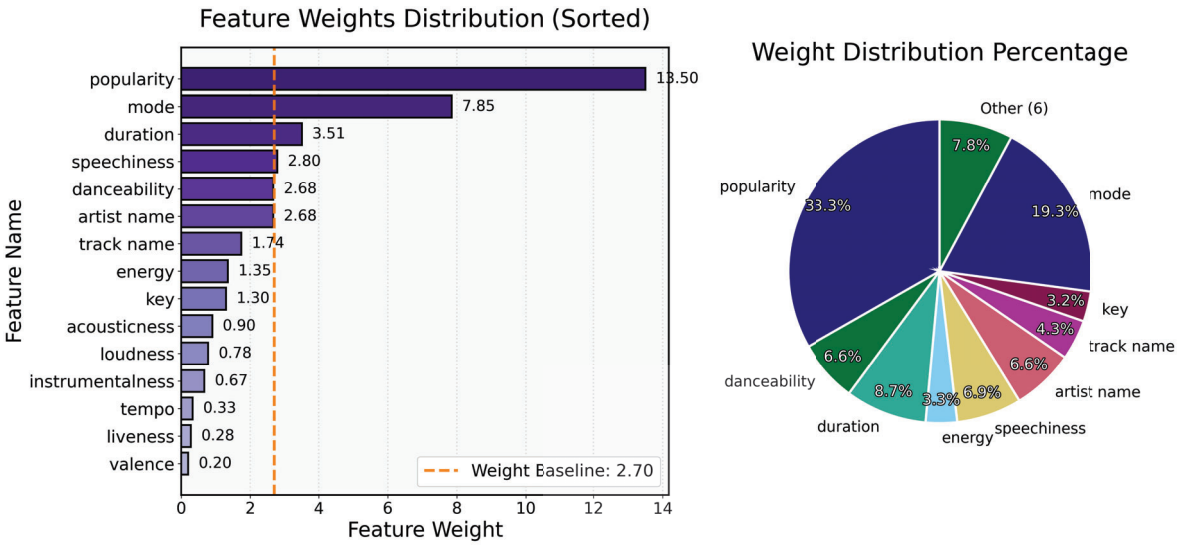

Figure 9: Importance (Perceptron weight) of all features

3. duration
4. speechiness

The *ordinary features* are the following.

1. danceability

2. artist name
3. track name
4. energy
5. key
6. acousticness
7. loudness
8. instrumentalness
9. tempo
10. liveness
11. valence

The outcome is basically consistent with intuition, which means that the Perceptron model trained in this research is quite trustworthy.

The *important features* are usually features with these two properties: (*a*) vary significantly between different genres of music, and (*b*) have a typical value for a certain genre. Some music genres may enjoy higher popularity than others (like *Anime* compared to *Blues*), making *popularity* an important feature. Some features which directly describe musical elements are also proved important, such as *speechiness* (*Rap* can have significantly denser lyric than *Classical*, which basically have none) or *duration* (again, *Classical* music is usually much longer than regular pop songs). Feature *mode* is also important, since music theory thinks that songs in *Major* mode tend to be brighter or happier while songs in *Minor* mode are gloomier, which can distinguish different music genres.

The *ordinary features* are usually features belonging to at most one of (*a*) and (*b*), but do not distinguish music genres as clearly as important features. For example, usually *Classical* music scores high in *instrumentalness*, but other song genres can also be high in *instrumentalness*, while their acoustic version can score very low. This means that *instrumentality* does not have a typical value for most genres of music and that it does not have property (*b*). Similarly, songs belonging to the same genre can have very different *valences* and *loudness*.

Apart from that, some features indicates some very common attributes that can be shared by all different genres. For example, the feature *key* belongs to the this category (based on music theory, transposing a music piece from one key to another will not change its character; changing the key is a common way to modify the pitch of a song to make it easier to sing without altering how it sounds).

It is also noteworthy that, although in practice songs by the same artist (corresponding to the *artist name* feature) or from the same record (corresponding to the *track name* feature) often belong to the same genre, the weights of these two features are relatively low in the results. This may be attributed to two factors. First, artist or track names can be misleading, as the naming may reflect characteristics or elements associated with other genres. Second, in this study, these features are treated as categorical variables and transformed into numerical representations using label encoding. During this process, meaningful semantic information is likely to be lost, reducing their discriminative power and consequently lowering their feature importance.

Still, we observe that some feature's *weights* are different from what we expected. Features like *duration* are more important than we thought intuitively, while *acousticness* and *liveness* are less important.

This outcome indicates that when trying to decide which feature of music should be taken into consideration to classify music genre, some certain features should be treated more seriously, i.e., having a greater coefficient. To be specific, this paper indicates that the *important features* listed above should have a greater effect than the *ordinary features* listed above.

5.2. Choices of Hyperparameters.

5.2.1. kNN. In this section, we will first discuss the performance of the three distance measuring formulas (3.2), (3.3), and (3.4). Then we will analyze the data shown in Figure 3 and discuss the value of k whose performance is relatively the best. In addition, we analyze the difference in accuracy of *multiclass kNN*, *One vs. All* , and *One vs. One*.

From Table 5 we can see that models with the L_1 distance outperform other types of models. Since the distributions of many features of the training set are not an approximately normal distribution (see Figure 1), the data points are actually not uniformly distributed in the feature space. This means that there are a lot of "outsiders" which are far from the center but may contain some information about some certain pattern of a music genre and affect kNN models when predicting labels. Therefore, if we use the L_2 distance to calculate the distance between these "outsiders" and test data points, we even enlarge the distance by computing the square of difference in value along each feature and thus reducing the impact of these faraway but important outside data points. Due to the same reason, *Mahalanobis* distance also has this shortage. So, the L_1 distance is the best distance measurement formula for this task.

It is also worth mentioning that the performances of models with *Mahalanobis* distance on both datasets before and after normalization are the same. This is because the *Mahalanobis* distance normalizes the data implicitly by adding the covariance term.

In the training stage, we have divided the value of k into 4 different scales: *small*, *medium*, *large*, and *extremely large*. The corresponding value ranges are $k \in (1, 30)$, $k \in (30, 300)$, $k \in (300, 3000)$ and $k \in (3000, 30000)$. There are 15 trials of k value for *small* and 30 trials for the other three ranges. All ks are ensured to be odd.

In Figure 3, the maximum *accuracy* (around 0.83) appears when $k \in [5, 13]$. In general, the average accuracy when k is *small* is significantly higher than when k is medium, large, or extremely large.

The accuracy value first increases sharply when k increases from 1 to 5, reaches its maximum at $k = 7$. After that, the accuracy drops continuously as k increases. Therefore, we can indicate that the best value of k is when k is relatively *small*, which means that k is around $\frac{\sqrt{N}}{10}$. In this paper, $k \in [5, 13]$ is the best range.

Furthermore, we use $k \in [5, 13]$ to analyze kNN models with different implementations of (*multiclass kNN*, *One vs. All* and *One vs. One*). The results are shown in Table 6, where we can see that the average accuracy of these kNN models is fairly close to each other, which indicates that changing the model ensemble methods does not greatly improve the accuracy in this task.

5.2.2. Perceptron. Similarly to kNN, we tested two implementations of Perceptron models, with *OvA* and *OvO*, respectively. The results of these two versions are shown in Table 7 and

8. We can see that the accuracy of the Perceptron model with *OvO* (77.19%) is significantly higher than that of *OvA* (58.81%). Therefore, undoubtedly *OvO* is the better method for combining multiple binary classification Perceptron models into one multi-class classification model.

A reason for this is that since Perceptron is a binary linear classifier, its geometric intuition is actually a hyperplane in the feature space of the dataset, dividing the dataset into two subsets regarding two classes. In the *OvA* method, we use all data from the original training set (containing 10 classes) to train every Perceptron model. Having multiple classes of data at one time may lead to poor binary divisibility of the feature space, thus causing low accuracy of the model. On the other hand, *OvO* uses a subset of data that contains only two classes at one time, which hopefully ensures the divisibility of the training set directly accessed by the Perceptron model.

However, this argument turned out not to be the main cause of the result. As indicated by Table 7, it shows that each model trained by *OvA* performs well enough on its own sub-task (achieving an average accuracy at 92.73%, slightly lower than 95.91% of *OvO*). If the guess on binary divisibility is correct, these models should have performed worse. But somehow the overall accuracy is still poor. This means that the problem is actually about how we assemble the scores of all models and conclude the final decision in *OvA*. In fact, the score of the Perceptron model trained by *OvA* is only the product value between the vector of features of the test data and the weight matrix in the correct class (label). This value is not numerically stable and may differ in scale between different models due to the actual training process. So, it is not wise to make a final decision simply by comparing the value of scores, which unfortunately happens to be what we have done in the *OvA* method.

However, this is no longer a problem in *OvO*, where we have multiple models to vote against each label and provide choices to choose from. This avoids the problem of numerical stability caused by directly using the matrix product. As we can see in Table 8, the overall accuracy maintains a satisfactory level of 77.19%.

5.2.3. MLP. In Table 11, we can infer several conclusions, listed below:

- Changing the activation function (models #1, #2 and #3) does not significantly affect accuracy, but can differ training efficiency (the stopped value of the epoch indicates the time when the model stops training).
- Among models with different *width* (models #1, #4, #5 and #6), model #4 (with width 10) has the best prediction accuracy. Wider models tend to have worse test performance and smaller stopped value of epoch (see Figure 5).
- Among models with different *depth* (models #1, #7, #8 and #9), model #9 (with depth 5) has the best accuracy. Increasing the depth of the models also slightly reduces the stopped value of the epoch (see also Figure 5).

Generally speaking, both increasing the *width* and the *depth* of models do not tend to behave significantly better in testing datasets (and stopping training earlier). This is because larger models are easier to overfit the training dataset in the early stage of training and stop getting better on the validation or testing dataset. Since our task is a relatively simple classification problem, models with simple structure are good enough to obtain high accuracy. Since larger models usually take longer time to train, in practice we can train small models

with higher efficiency while ensuring accuracy.

5.2.4. AdaBoost. As shown in Figure 8, we can see that the best accuracy is 77.22%.

We can see that increasing the number of training cycles does not necessarily improve the performance of the model. There are a few reasons for this result. Since AdaBoost trains the model by adding weak learners to the former model, once it has already captured all relatively useful features in the dataset, the new added weak learners will recognize the noisy data in the training set, eventually causing the model to overfit the train data. Apart from that, Adaboost changes the weights of the sample data in each learning cycle based on the result from the last cycle, which may cause the sample weights to be either extremely large or extremely small after a certain number of training cycles, causing the model to pay too much attention to abnormal data and reduces final performance.

5.3. Model Accuracy. One of the goals of this paper is to assess how well basic algorithms like kNN and Perceptron perform in music genre classification tasks, compared to relatively more complex models like MLP.

We can see the overall accuracy of the kNN model in Table 6. Assuming that the best range of k is used (based on the discussion in the section above), we can expect the model to have an accuracy around 83.37%. For the perceptron model, if the *OvA* method is adopted, it can achieve an accuracy of 77.19%, as shown in Table 8. In Table 11, we can see that the best accuracy of the MLP model is around 90.40%. For AdaBoost, as shown in Figure 8, the best accuracy is 77.22%. Table 12 lists the accuracy of the best models and their hyperparameters studied in this investigation.

Table 12: Accuracy of different models

Model	Performance	Description
kNN	83.37%	K = 7, OvA
Perceptron	77.19%	OvO
MLP	90.40%	D = 2, W = 10, ReLU
AdaBoost	77.22%	lr = 0.50, lc = 1000

It should be noted that genre overlap may occur in practice, meaning that a single piece of music can belong to multiple categories (e.g., a rock track used as an anime soundtrack may be classified as both *anime* and *rock*). However, the models trained in this study are designed to predict a single label as the best choice. This formulation may therefore affect the reported accuracy to some extent.

5.4. Models Trained by Important Features. After analyzing the importance of each feature in the training dataset, we remove some of the least important features from the dataset based on the above analysis and use the remaining features to train some new models. In this stage, we choose the relatively best hyperparameters shown in Table 12 for each algorithm.

As shown in Figure 9, the average importance percentage of all features is 6.7%. Take half of this baseline value $\frac{1}{2} \times 6.7\% = 3.4\%$ as a threshold. Features whose importance percentage

is less than 3.4% are removed from the dataset. The remaining and removed features are listed in Figure 10.

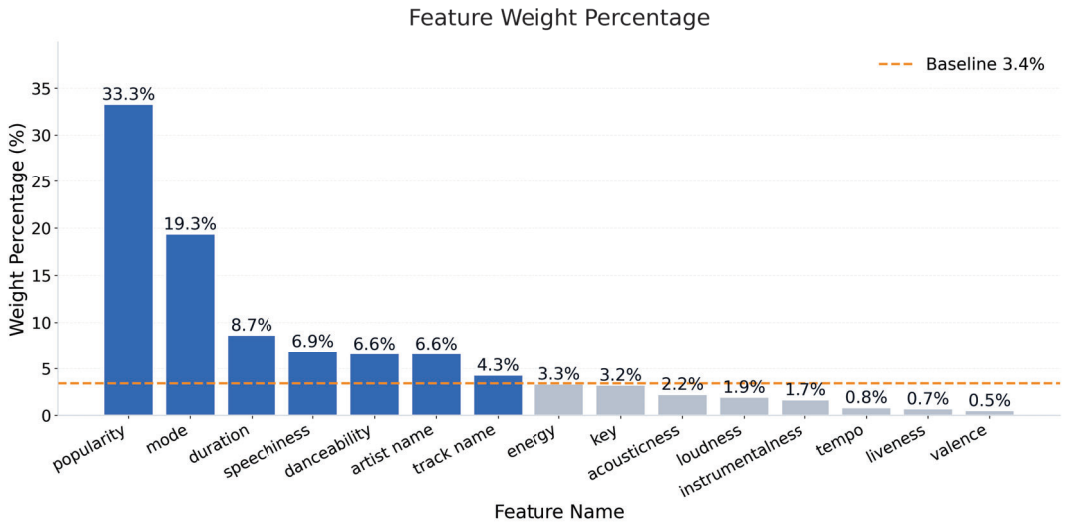

Figure 10: Remaining and removed features based on importance percentage

We train four new models (kNN, Perceptron, MLP, and AdaBoost) with their hyperparameters set to the best choices given Table 12. The accuracy of these models is shown in Table 13.

Table 13: Accuracy of different models trained by important features only

Model	Acc	Original Acc	Improved by
kNN	**88.50%**	83.37%	5.13%
Perceptron	**77.76%**	77.19%	0.57%
MLP	**90.72%**	90.40%	0.32%
AdaBoost	**77.22%**	77.22%	0.00%

From Table 13, we can observe that the effect of feature reduction varies between different algorithms. For kNN, accuracy improves significantly by more than 5%, suggesting that removal of noisy or less informative features helps alleviate the curse of dimensionality and allows the model to better capture the neighborhood structure of the data. The Perceptron also benefits slightly from the reduced feature set, indicating that pruning redundant features can simplify the decision boundary.

MLP only shows a marginal increase in performance, likely because neural networks can implicitly handle irrelevant features during training and do not necessarily rely on the scale of input space. For AdaBoost, the accuracy remains unchanged, which implies that the algorithm is relatively robust to the inclusion of weaker features due to its ensemble nature.

Overall, these results highlight that feature selection can have markedly different impacts

depending on the learning algorithm. Specifically, reducing feature space can be particularly beneficial for simpler or distance-based models.

6. Conclusion. This paper aimed to evaluate the effectiveness and efficiency of different machine learning (ML) algorithms in classifying music genres. The study focused on four basic ML algorithms: K-Nearest Neighbors (kNN), Perceptron Learning Algorithm (PLA), Multilayer Perceptron (MLP) and AdaBoost. In addition, we provided some useful methods to combine binary classification models into multi-class classification models. The study also highlighted the importance of certain features in the classification of music genres and provided recommendations to select the most suitable algorithm and set of features for this task.

This work has shown that automatic music genre classification can be achieved using simple models with high accuracy and low training costs.

REFERENCES

[1] Hareesh Bahuleyan. Music genre classification using machine learning techniques. *arXiv preprint arXiv:1804.01149*, 2018.

[2] Snigdha Chillara, AS Kavitha, Shwetha A Neginhal, Shreya Haldia, and KS Vidyullatha. Music genre classification using machine learning algorithms: a comparison. *Int Res J Eng Technol*, 6(5):851–858, 2019.

[3] Anirudh Ghildiyal, Komal Singh, and Sachin Sharma. Music genre classification using machine learning. In *2020 4th international conference on electronics, communication and aerospace technology (ICECA)*, pages 1368–1372. IEEE, 2020.

[4] Ndiatenda Ndou, Ritesh Ajoodha, and Ashwini Jadhav. Music genre classification: A review of deep-learning and traditional machine-learning approaches. In *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–6. IEEE, 2021.

[5] Buse M Ozyildirim and Mariam Kiran. Levengerg-Marquardt multi-classification using hinge loss function. *Neural Networks*, 143:564–571, 2021.

[6] Nikki Pelchat and Craig M Gelowitz. Neural network music genre classification. *Canadian Journal of Electrical and Computer Engineering*, 43(3):170–173, 2020.

[7] Carlos N Silla, Alessandro L Koerich, and Celso AA Kaestner. A machine learning approach to automatic music genre classification. *Journal of the Brazilian Computer Society*, 14:7–18, 2008.