Comparison of Different Machine Learning Models on Classification of Music Genres

Project Category: Machine learning

HAO Jiadong
DUAN Mingfei
HUANG Yufeng

Abstract

Music genre classification has been an active research area in recent years, with increasing amount of digital music data available. In this paper, we compared nine different music genre classifiers and evaluated their performance on a large and famous dataset of music tracks called FMA (Free Music Archive). We applied various feature extraction techniques and machine learning algorithms to classify music tracks into nine top-genres. Our results showed that the performance of the classifiers varies significantly depending on the machine learning algorithms used. We also presented a comprehensive reflection on the results and proposed some possible future improvements.

1. Introduction

Music genre classification is a challenging problem and has remained to be a hot topic in recent years. The ability to automatically categorize large amounts of music tracks into different genres has numerous practical applications, including music recommendation systems, playlist generation, music search engines and music streaming services [1]. For instance, music recommendation systems like Spotify and Pandora rely heavily on genre classification algorithms to suggest songs to users. Additionally, music search engines like Shazam use genre classification to identify songs and provide information on them [2]. Moreover, genre classification models can also be used in music analysis and research, such as studying the evolution of musical genres over time [3]. Hence, a powerful machine learning model for music genre classification is in great need.

Over the years, researchers have been trying various approaches to automate the music classification process, ranging from traditional machine learning techniques [3] to more recent deep learning methods [4][5].

In this paper, our team trained 9 classifiers which are SVM, Logistic Regression, K-NN, Naïve Bayes, Quadratic Discriminant Analysis, Random Forest, Multilayer perceptron, CatBoost and XGBoost, and further compared their performance. We also discussed some insights and some extrapolations based on the results.

2. Dataset

The dataset used is the Free Music Archive (FMA), a large, open-source music dataset published in 2017 containing 106,574 music tracks from 16,341 artists and 14,854 albums, arranged in a hierarchical taxonomy of 161 genres [6].

We mainly focus on four files, which are:

File name	Description
tracks.csv	Metadata about each track in the dataset, such as the track ID, the album, the artist, and the title. It also includes information about the license under which each track is released, the duration, and number of listens.
features.csv	Various audio features extracted from each track, such as tempo, key, and spectral characteristics, generated by the Marsyas feature extraction library and are stored as numerical values.
echonest.csv	Additional metadata and features, obtained by the Echo Nest API, including features such as danceability, energy, and loudness, as well as information about the popularity and familiarity of each track.
genres.csv	An excerpt of built-in genre hierarchy, that is all 163 genres with name and parent (top-level genres).

In the genres.csv, the column "top_level" indicates the top-level genres of a particular genre. In our experiment, we only considered the 9 top-level genres to simplify the problem and save our time and computational power.

id	parent	top_level	title	#tracks
38	None	38	Experimental	38,154
15	None	15	Electronic	34,413
12	None	12	Rock	32,923
1235	None	1235	Instrumental	14,938
25	12	12	Punk	9,261
89	25	12	Post-Punk	1,858
1	38	38	Avant-Garde	8,693

Fig.1 The file genres.csv storing the genre hierarchy

The figure below shows the statistics of all 16 top-genres existing in the genres file. However, during the data preprocessing phase, we dropped some records hence some of the top-genres may have very few instances, causing extreme imbalance in the dataset. Hence, we manually merged those top-genres with very few records into a new group called "Other" so that finally we only considered 9 top-genres (details see in 3.2.4).

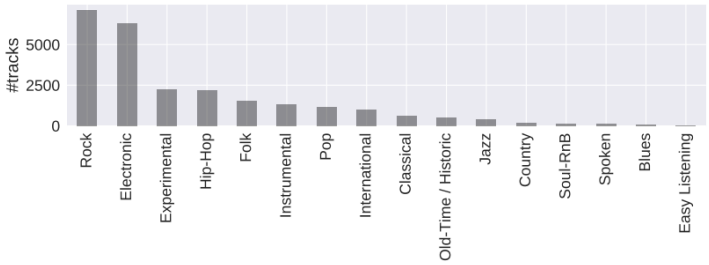

Fig.2 Statistics of the number of tracks in each top-level genre

3. Experiment and Methods

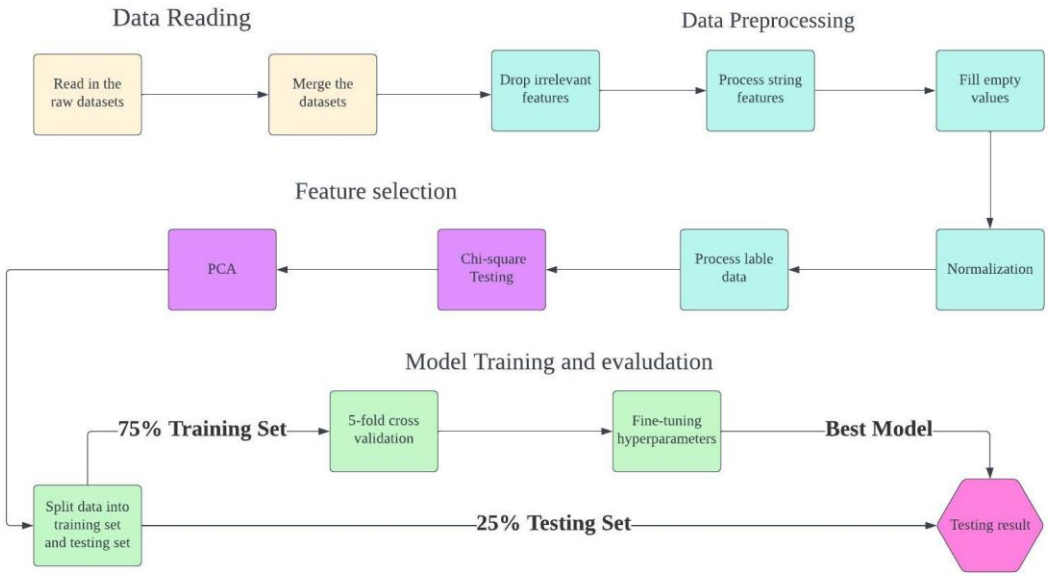

Fig.3 Our machine-learning pipeline

3.1 Data Reading

The tracks.csv, features.csv, and echonest.csv provided the training features while the genres.csv provided a mapping between the sub-genre of a track and its top-genre. We further inner-joined the three files containing the features so that only tracks that exist in all three files are retained, reducing the dataset size to 13554.

3.2 Data Preprocessing

3.2.1 Dropping irrelevant features by domain knowledge

By analyzing all the training features, we found that some of the training features are totally irrelevant to our genre classification problem, like the location of the artist. Including them may downgrade the performance of our models and dramatically increase the time and computational cost. Hence, we choose to drop all these irrelevant features by domain knowledge at the very first stage of the data preprocessing. At the end of this step, we reduced the number of features from 806 to 776.

```
dropped = ["artist_location", "tags", "artist_url", "artist_website",
           "license_image_file_large", "track_url", "license_url",
           "license_title", "license_image_file", "track_file",
           "track_explicit", "artist_name_x", "album_title",
           "track_date_recorded", "track_date_created", "track_lyricist",
           "track_information", "track_copyright_c","track_copyright_p",
           "artist_name_y", "track_explicit_notes", "track_composer",
           "track_publisher", "album_url", "release","album_name",
           "album_date", "album_id", "track_image_file", "track_language_code"]
```

Fig.4 Dropping list of irrelevant features

3.2.2 Processing the string features

After dropping all irrelevant features, there were two features remaining to be in string format that needed to be converted to numeric values before being fed to the machine learning models, which were “track_duration” and “track_title”.

For “track_duration”, which is in the format “xx:yy”, where xx is for minutes and yy is for seconds, we just converted them to the number of seconds.

For “track_title”, the Word2Vec algorithm was used to convert the feature into numeric values. Word2Vec algorithm is a commonly used neural network-based algorithm in natural language processing to convert text data into numerical format. It can represent words as vectors, where words with similar meanings are placed close to each other in a high-dimensional vector space [7]. In our case, we used this algorithm to convert the track titles into numerical vectors, which can be taken in by the models.

We first used Word2Vec to create feature vectors for each word that appears in the track titles, which allowed us to capture the semantic meaning of each word in the track titles. Then, to calculate the semantic meaning of each title, we used the mean value of the words in the title, which was then used as a series of features in our machine learning models [8].

By using Word2Vec to convert track titles into vectors, we were able to improve the result of our models, as it helped to capture the semantic meaning of the track titles, allowing the models to better understand the underlying sentiment and emotional tone of the songs. This approach proved to be effective in our analysis, as it allowed us to accurately predict the sentiment of each track based on the track’s title.

After this step, the number of features increased to 874.

3.2.3 Fill in empty values and normalize the data

We used the mean values to fill the empty values and further normalized the data with a MinMaxScaler to prepare the features ready for training.

3.2.4 Process label data

To simplify our problem, we only considered single-genre instances and ignored all the instances with multiple genres and with no genre, reducing the number of training instances to 10192. Furthermore, we mapped all the sub-genres to their top-genre as mentioned above. Below shows the statistics about the 9 top-genres after mapping.

Rock	3493
Electronic	2514
Folk	1036
Hip-Hop	987
Pop	533
Other	442
Old-Time / Historic	430
Jazz	403
Classical	354

dtype: int64

Fig.5 Statistics about the 9 top-genres

3.3 Feature Selection and Dimensionality Reduction

3.3.1 Chi-square testing

We calculated the chi-square score of all training features and filtered out all insignificant features with chi-square score less than 2, reducing the number of training features to 744.

3.3.2 PCA

Principal Component Analysis (PCA) was used to compress the dimensions of the dataset while retaining at least 90% of the variance. This was done by transforming the high-dimensional features into lower-dimensional principal components. After performing PCA, the number of features was significantly reduced to 121, allowing for a more comprehensive understanding of the dataset's structure and reducing the likelihood of redundancy. By selecting the most important principal components based on their variance contributions, we were able to make more accurate predictions and analyses.

3.4 Model Training and Evaluation

We split the whole dataset into a 75% training set and a 25% testing set. For the training set, we used 5-fold cross-validation to fine-tune the hyperparameters of each of the nine machine learning models. Then, we applied the 9 best models to the testing set to see their performance.

For the fine-tuning strategy, we adopt the “GridSearchCV” in “sklearn.model_selection” package, which loops through all the combinations of predefined hyperparameters and gives back the best model on the training set [9].

For the evaluation metric, since we had a relatively imbalanced dataset, accuracy may not be suitable because it can be biased towards the majority class, leading to misleading performance evaluations. And standard F1 score also failed because it doesn’t fit multi-class classification. Hence, we chose to use the macro F1 score as our main evaluation metric, which gives equal importance to each class by averaging the F1 scores of each class.

3.4.1 Support Vector Machine

Support Vector Machine (SVM) is a classical machine learning technique that is usually used to solve big data classification problems. Its principle is to find a hyperplane that separates the data into two classes with the maximum margin [10].

In our experiment, we took two hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
C	Regularization parameter. It represents the strength of the regularization that is inversely proportional to C. Smaller values specify stronger regularization.	0.1, 1, 10
Kernal	The kernel type to be used in the algorithm.	'linear', 'poly', 'rbf', 'sigmoid'

```
Hyperparameters:  {'C': 0.1, 'kernel': 'linear'}
Mean cross-validation score (macro f1):  0.6036994453583988
Hyperparameters:  {'C': 0.1, 'kernel': 'poly'}
Mean cross-validation score (macro f1):  0.36368495295496805
Hyperparameters:  {'C': 0.1, 'kernel': 'rbf'}
Mean cross-validation score (macro f1):  0.45756772447520044
Hyperparameters:  {'C': 0.1, 'kernel': 'sigmoid'}
Mean cross-validation score (macro f1):  0.45779583106451066
Hyperparameters:  {'C': 1, 'kernel': 'linear'}
Mean cross-validation score (macro f1):  0.6394094774869735
Hyperparameters:  {'C': 1, 'kernel': 'poly'}
Mean cross-validation score (macro f1):  0.5838703536200239
Hyperparameters:  {'C': 1, 'kernel': 'rbf'}
Mean cross-validation score (macro f1):  0.6559864950945296
Hyperparameters:  {'C': 1, 'kernel': 'sigmoid'}
Mean cross-validation score (macro f1):  0.5015692636771256
Hyperparameters:  {'C': 10, 'kernel': 'linear'}
Mean cross-validation score (macro f1):  0.6354135776033457
Hyperparameters:  {'C': 10, 'kernel': 'poly'}
Mean cross-validation score (macro f1):  0.6791019587414532
Hyperparameters:  {'C': 10, 'kernel': 'rbf'}
Mean cross-validation score (macro f1):  0.7193830628454315
Hyperparameters:  {'C': 10, 'kernel': 'sigmoid'}
Mean cross-validation score (macro f1):  0.4606634845757334
--------------------------------------------------------------------------------
Best hyperparameters:  {'C': 10, 'kernel': 'rbf'}
Best performance in 5-fold cross validation (macro f1):  0.7193830628454315
```

Fig.6 Tuning of SVM

After the fine-tuning, we found that {'C': 10, 'kernel': 'rbf'} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.2 Logistic Regression

Logistic Regression is a linear prediction model that applies step-by-step coverage to do value forecast and classification [11]. It plays an important role in statistics because it can avoid uncertainty by considering the relevance of all variables together [12].

In our experiment, we considered 3 hyperparameters when doing the fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
C	Regularization parameter. It represents the strength of the regularization that is inversely proportional to C. Smaller values specify stronger regularization.	0.1, 1, 10, 50

Solver	Type of the kernel	'saga', 'sag'
Multi_class	Combine with the solver to adjust the loss.	'ovr', 'multinomial'

```
Hyperparameters:  {'C': 0.1, 'multi_class': 'ovr', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.5573989697632264
Hyperparameters:  {'C': 0.1, 'multi_class': 'ovr', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.5573989697632264
Hyperparameters:  {'C': 0.1, 'multi_class': 'multinomial', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.5798290880846863
Hyperparameters:  {'C': 0.1, 'multi_class': 'multinomial', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.5798290880846863
Hyperparameters:  {'C': 1, 'multi_class': 'ovr', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6179940233029388
Hyperparameters:  {'C': 1, 'multi_class': 'ovr', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.6177223138322564
Hyperparameters:  {'C': 1, 'multi_class': 'multinomial', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6331059093215099
Hyperparameters:  {'C': 1, 'multi_class': 'multinomial', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.6331059093215099
Hyperparameters:  {'C': 10, 'multi_class': 'ovr', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6230225007890734
Hyperparameters:  {'C': 10, 'multi_class': 'ovr', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.6230225007890734
Hyperparameters:  {'C': 10, 'multi_class': 'multinomial', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6332638823856073
Hyperparameters:  {'C': 10, 'multi_class': 'multinomial', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.6334918876869164
Hyperparameters:  {'C': 50, 'multi_class': 'ovr', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6210510418385785
Hyperparameters:  {'C': 50, 'multi_class': 'ovr', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.621444290693242
Hyperparameters:  {'C': 50, 'multi_class': 'multinomial', 'solver': 'saga'}
Mean cross-validation score (macro f1):  0.6328328213435876
Hyperparameters:  {'C': 50, 'multi_class': 'multinomial', 'solver': 'sag'}
Mean cross-validation score (macro f1):  0.6324059575855888
--------------------------------------------------------------------------------
Best hyperparameters:  {'C': 10, 'multi_class': 'multinomial', 'solver': 'sag'}
Best performance in 5-fold cross validation (macro f1):  0.6334918876869164
```

Fig.7 Tuning of Logistic Regression

After the fine-tuning, we found that {'C': 10, 'multi_class': 'multinomial', 'solver': 'sag'} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.3 K-NN

K-NN is a non-parametric algorithm that predicts the class of a new data point by finding the K-closest training samples and classifying it based on the majority class among its neighbors [13]. It is a lazy learning method that highly depends on the value of K.

In our experiment, we took 3 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
n_neighbors	Number of neighbors	3, 5, 7
weights	Weight function used in prediction.	'uniform', 'distance'
leaf_size	The number of data points stored in the leaf nodes of the KD-tree data structure, which affects the speed of the construction and query, as well as the memory required to store the tree.	10, 20, 30

```
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 3, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6224429172911906
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6353352849684487
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 5, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6094034554790662
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 5, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6379511561781869
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 7, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.59886506171967
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 7, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6232917520196285
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 3, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6224429172911906
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 3, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6353352849684487
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 5, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6094034554790662
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 5, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6379511561781869
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 7, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.59886506171967
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 7, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6232917520196285
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 3, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6224429172911906
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 3, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6353352849684487
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 5, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.6094034554790662
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 5, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6379511561781869
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 7, 'weights': 'uniform'}
Mean cross-validation score (macro f1):  0.59886506171967
Hyperparameters:  {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 7, 'weights': 'distance'}
Mean cross-validation score (macro f1):  0.6232917520196285
--------------------------------------------------------------------------------
Best hyperparameters:  {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 5, 'weights': 'distance'}
Best performance in 5-fold cross validation (macro f1):  0.6379511561781869
```

Fig.8 Tuning of KNN

After the fine-tuning, we found that {'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 5, 'weights': 'distance'} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.4 Naïve Bayes

Naïve Bayes is a mechanism for using the information in sample data to estimate the posterior probability of each class y given an object x. It predicts the class of an instance based on the Bayes' theorem and the assumption that all features are independent [14].

In our experiment, we took 2 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
alpha	Value for Laplace Smoothing	0.1, 0.5, 1.0
fit_prior	Whether to take prior probabilities into account.	True, False

```
Hyperparameters:  {'alpha': 0.1, 'fit_prior': True}
Mean cross-validation score (macro f1):  0.056100666544975 63
Hyperparameters:  {'alpha': 0.1, 'fit_prior': False}
Mean cross-validation score (macro f1):  0.5790154593647915
Hyperparameters:  {'alpha': 0.5, 'fit_prior': True}
Mean cross-validation score (macro f1):  0.056100666544975 63
Hyperparameters:  {'alpha': 0.5, 'fit_prior': False}
Mean cross-validation score (macro f1):  0.5779476145484745
Hyperparameters:  {'alpha': 1.0, 'fit_prior': True}
Mean cross-validation score (macro f1):  0.056100666544975 63
Hyperparameters:  {'alpha': 1.0, 'fit_prior': False}
Mean cross-validation score (macro f1):  0.5781739619642365
--------------------------------------------------------------------------------
Best hyperparameters:  {'alpha': 0.1, 'fit_prior': False}
Best performance in 5-fold cross validation (macro f1):  0.5790154593647915
```

Fig.9 Tuning of Naïve Bayes

After the fine-tuning, we found that {'alpha': 0.1, 'fit_prior': False} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.5 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis models the probability density function of each class using a quadratic function and assigns the class with the highest probability to a new data point [15].

In our experiment, we took 3 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
reg_param	Regularizes the per-class covariance estimates	0.0, 0.1, 0.5, 1.0
store_covariance	Whether regularizing the per-class covariance estimates	True, False
tol	The absolute threshold for a singular value to be considered significant.	1e-3, 1e-4, 1e-5

```
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': True, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': True, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': True, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': False, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': False, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.0, 'store_covariance': False, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.6070307452152996
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': True, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': True, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': True, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': False, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': False, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.1, 'store_covariance': False, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.19051140893477328
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': True, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': True, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': True, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': False, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': False, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 0.5, 'store_covariance': False, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': True, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': True, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': True, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': False, 'tol': 0.001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': False, 'tol': 0.0001}
Mean cross-validation score (macro f1):  0.05610066654497563
Hyperparameters:  {'reg_param': 1.0, 'store_covariance': False, 'tol': 1e-05}
Mean cross-validation score (macro f1):  0.05610066654497563
--------------------------------------------------------------------------------
Best hyperparameters:  {'reg_param': 0.0, 'store_covariance': True, 'tol': 0.001}
Best performance in 5-fold cross validation (macro f1):  0.6070307452152996
```

Fig.10 Tuning of Quadratic Discriminant Analysis

After the fine-tuning, we found that {'reg_param': 0.0, 'store_covariance': True, 'tol': 0.001} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.6 Random Forest

Random Forest is an ensemble model that builds multiple decision trees on randomly sampled subsets of features and data and aggregates their predictions to improve the accuracy and reduce the overfitting of the model.

In our experiment, we took three hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
n_estimators	The number of trees in the forest	200,500
max_depth	The maximum depth of the tree.	10, 20
max_features	The number of features to consider when looking for the best split	'sqrt','log2'

```
Hyperparameters:  {'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.43087765291863456
Hyperparameters:  {'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.4331841544070921
Hyperparameters:  {'max_depth': 10, 'max_features': 'log2', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.3497731296324221
Hyperparameters:  {'max_depth': 10, 'max_features': 'log2', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.3509074959735617
Hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.476486599562587
Hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.47920879604569055
Hyperparameters:  {'max_depth': 20, 'max_features': 'log2', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.44404057266585034
Hyperparameters:  {'max_depth': 20, 'max_features': 'log2', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.44136383587836675
--------------------------------------------------------------------------------
Best hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
Best performance in 5-fold cross validation (macro f1):  0.47920879604569055
```

Fig.11 Tuning of Random Forest

After the fine-tuning, we found that {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.7 Multiple Layer Perceptron

The Multiple Layer Perceptron (MLP) is a feedforward neural network that consists of multiple layers of nodes, where each node is a non-linear function of a linear combination of its inputs, and it is trained using backpropagation to minimize the error between the predicted and actual outputs [16].

In our experiment, we took 3 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
hidden_layer_sizes	The i^{th} element represents the number of neurons in the i^{th} hidden layer.	(50,), (100,), (500,), (1000,), (50,30,), (100,50,), (500,200,), (50,30,20,)

activation	Activation function for the hidden layer.	'relu', 'logistic'
solver	The solver for weight optimization.	'lbfgs', 'adam'

```
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6218509156591677
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6555820148235988
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (100,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.647048240356242
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (100,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6781780006380916
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (500,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6730883193624306
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (500,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.7181692966109023
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (1000,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6771458847395049
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (1000,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.7154637062304394
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50, 30), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6248464523068918
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50, 30), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6373521948328634
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (100, 50), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6477211750769787
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (100, 50), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6735736052456822
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (500, 200), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6792188245675594
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (500, 200), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.7022652888844366
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50, 30, 20), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6149542715360308
Hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (50, 30, 20), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6201769974708611
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6025810677240011
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6494221459276248
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6264131169810951
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (100,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6475885440136243
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (500,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6306185529769344
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (500,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6388136511630205
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (1000,), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6265238011874281
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (1000,), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6354125530753137
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50, 30), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6166923313981109
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50, 30), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6361725475760851
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (100, 50), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6336296231732591
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (100, 50), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6545140765544816
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (500, 200), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.6117696975353886
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (500, 200), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.6593808828399527
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50, 30, 20), 'solver': 'lbfgs'}
Mean cross-validation score (macro f1):  0.5921833944476147
Hyperparameters:  {'activation': 'logistic', 'hidden_layer_sizes': (50, 30, 20), 'solver': 'adam'}
Mean cross-validation score (macro f1):  0.5651201151324623
--------------------------------------------------------------------------------
Best hyperparameters:  {'activation': 'relu', 'hidden_layer_sizes': (500,), 'solver': 'adam'}
Best performance in 5-fold cross validation (macro f1):  0.7181692966109023
```

Fig.11 Tuning of Multiple Layer Perceptron

After the fine-tuning, we found that {'activation': 'relu', 'hidden_layer_sizes': (500,), 'solver': 'adam'} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.8 CatBoost (not implemented in sklearn)

CatBoost is an algorithm for gradient boosting on decision trees which is developed by Yandex researchers in 2017 [17].

CatBoost works by building a set of decision trees consecutively, where each tree tries to reduce the loss compared to the previous trees. To build each tree, CatBoost uses a greedy algorithm that splits the data based on the feature that minimizes the loss function. However, unlike other boosting algorithms, CatBoost uses oblivious decision trees, where each split is based on the same feature for all the data points at a given level of the tree. This makes the prediction faster and more robust to noise. [17].

In our experiment, we took 2 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
iterations	Number of iterations	50,100,500
learning_rate	Learning rate in gradient descent	0.01, 0.1

```
Hyperparameters:  {'iterations': 50, 'learning_rate': 0.01}
Mean cross-validation score (macro f1):  0.365432136578692
Hyperparameters:  {'iterations': 50, 'learning_rate': 0.1}
Mean cross-validation score (macro f1):  0.4558863813009725
Hyperparameters:  {'iterations': 100, 'learning_rate': 0.01}
Mean cross-validation score (macro f1):  0.38546197541256083
Hyperparameters:  {'iterations': 100, 'learning_rate': 0.1}
Mean cross-validation score (macro f1):  0.5122087572014571
Hyperparameters:  {'iterations': 500, 'learning_rate': 0.01}
Mean cross-validation score (macro f1):  0.4668479408999766
Hyperparameters:  {'iterations': 500, 'learning_rate': 0.1}
Mean cross-validation score (macro f1):  0.6207619387305623
--------------------------------------------------------------
Best hyperparameters:  {'iterations': 500, 'learning_rate': 0.1}
Best performance in 5-fold cross validation (macro f1):  0.6207619387305623
```

Fig.13 Tuning of CatBoost

After the fine-tuning, we found that {'iterations': 500, 'learning rate': 0.1} performs best. Therefore, we applied this combination to the final model in the testing process.

3.4.9 XGBoost (not implemented in sklearn)

XGBoost (Extreme Gradient Boosting) is a powerful and scalable gradient boosting algorithm that uses decision trees as base learners to make predictions which was invented by Tianqi Chen in 2014.

The principle of XGBoost can be explained in the following steps [18]:

1. Initialize the model with the mean value of the target variable.
2. Train a decision tree to predict the residuals of the previous model.
3. Add the new decision tree model to the previous model and update the predictions.
4. Repeat steps 2 and 3 until a specified number of trees (n_estimators) have been built, or until the performance on the validation set stops improving.
5. Make predictions by summing the predictions of all the decision trees.

In our experiment, we took 2 hyperparameters into consideration when doing fine-tuning. Here shows the hyperparameter list and the fine-tuning result:

Parameters	Description	Value
max_depth	maximum depth of each decision tree	5, 7, None

n_estimators	the number of decision trees	200, 500

```
Hyperparameters:  {'max_depth': 5, 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.6169120288510787
Hyperparameters:  {'max_depth': 5, 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.6290077173840094
Hyperparameters:  {'max_depth': 7, 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.6209578651726337
Hyperparameters:  {'max_depth': 7, 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.6278184693935199
Hyperparameters:  {'max_depth': None, 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.622257014677521
Hyperparameters:  {'max_depth': None, 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.6303818011735993
--------------------------------------------------------------------------------
Best hyperparameters:  {'max_depth': None, 'n_estimators': 500}
Best performance in 5-fold cross validation (macro f1):  0.6303818011735993
```

Fig.14 Tuning of XGBoost

By adjusting the maximum depth and number of estimators, we found that {'max_depth': None, 'n_estimators': 500} performs best. Therefore, we applied this combination to the final model in the testing process.

4. Result

After getting the best settings for all 9 kinds of machine-learning models, we apply those models to the testing set to compare their performance. Below shows a comparison table and confusion matrixes of the 9 models.

Model	Training Time	Testing Time	Accuracy	Precision	Recall	F1-Score (macro)
SVM	5.7801	2.2243	77.55%	0.7848	0.7179	0.7452
Logistic Regression	7.4726	0.0015	71.82%	0.6632	0.6204	0.6299
K-NN	0.004	0.1600	69.15%	0.6773	0.6361	0.6488
Naïve Bayes	0.0159	0.0030	62.05%	0.5947	0.6162	0.5749
QDA	2.5069	0.0282	71.04%	0.7492	0.6049	0.6491
Random Forest	67.0917	0.4688	66.33%	0.7394	0.4719	0.4929
Multiple Layer perceptron	39.6285	0.0159	76.73%	0.7455	0.7115	0.7261
CatBoost	43.3773	0.0633	72.53%	0.7706	0.6057	0.6369
XGBoost	64.0567	0.0705	72.45%	0.7421	0.6095	0.6424

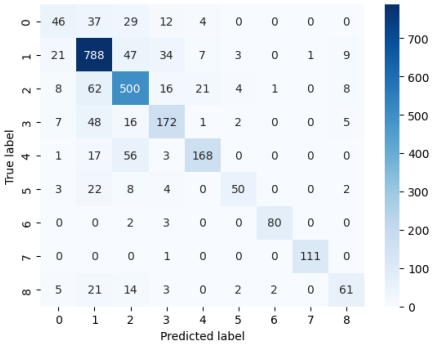

Fig.15 Confusion matrix of SVM

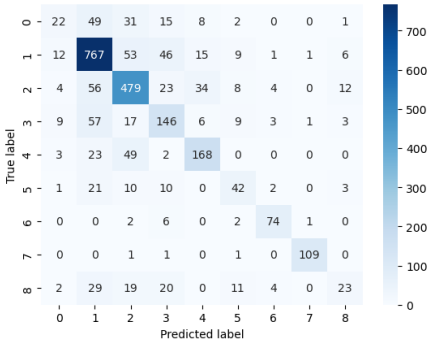

Fig.16 Confusion matrix of Logistic Regression

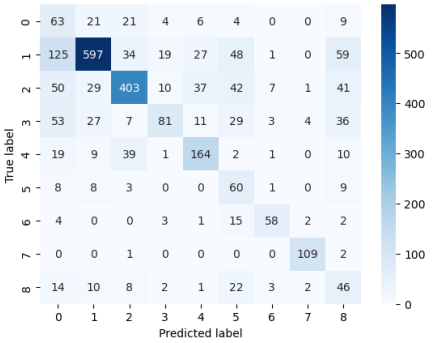

Fig.17 Confusion matrix of Naïve Bayes

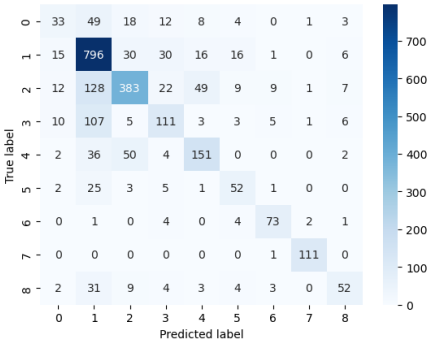

Fig.18 Confusion matrix of K-NN

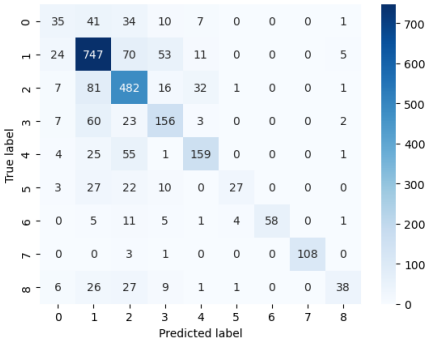

Fig.19 Confusion matrix of Quadratic Discriminant Analysis

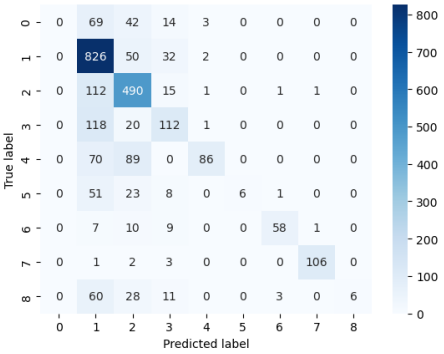

Fig.20 Confusion matrix of Random Forest

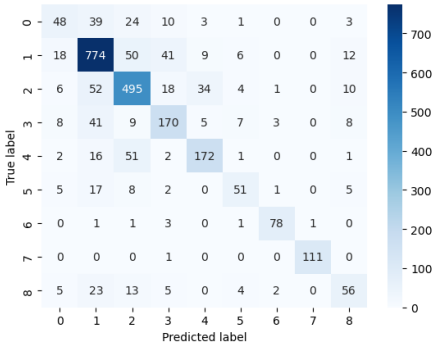

Fig.21 Confusion matrix of Multiple Layer perceptron

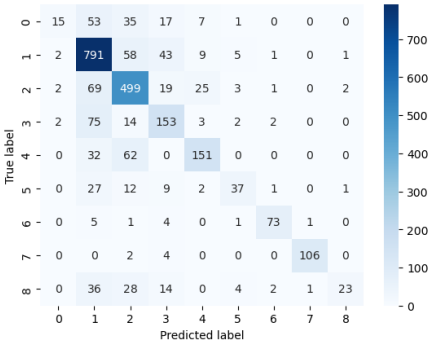

Fig.22 Confusion matrix of CatBoost

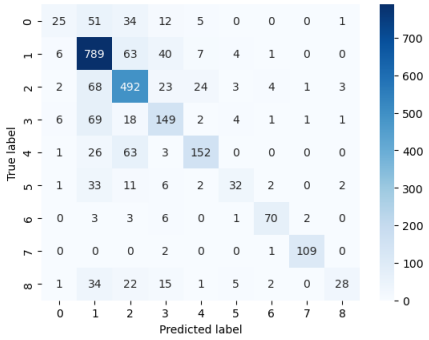

Fig.23 Confusion matrix of XGBoost

From the results, we can see that SVM achieved the highest F1 score of 0.7452 and the highest accuracy of 77.55%, followed closely by Multiple Layer Perceptron with an F1 score of 0.7261 and an accuracy of 76.73%. On the other hand, the performance of Naïve Bayes and Random Forest seems to be unacceptable, with F1 scores of merely 0.5749 and 0.4929 respectively. The rest 5 models also achieved a relatively high F1 score around 0.63-0.64.

Overall, the results suggest that SVM and Multiple Layer Perceptron are the most promising models for our music genre classification problem, as they achieved the highest accuracy and F1 scores. In contrast, Naïve Bayes and Random Forest with the current hyperparameter settings may not be ideal choices.

5. Insights and Reflection

5.1 Importance of feature selection and dimensionality reduction

To verify our team conducted effective feature reduction, we further compared the performance of the best models we obtained (SVM and Multiple Layer Perceptron) trained by features with Chi-square Testing and PCA (totally 121 features), and without them (totally 874 features).

Model	Number of features	Training Time	Testing Time	Accuracy	Precision	Recall	F1-Score (macro)
SVM(with feature selection)	121	5.7801	2.2243	77.55%	0.7848	0.7179	0.7452
SVM(without feature selection)	874	12.3244	4.0154	74.05%	0.7076	0.6751	0.6888
MLP(with feature selection)	121	39.6285	0.0159	76.73%	0.7455	0.7115	0.7261
MLP(without feature selection)	874	132.4131	0.0179	73.46%	0.6884	0.7006	0.6894

From the above table, for both SVM and the Multiple Layer Perceptron, ignoring feature selection and dimensionality reduction will not only lead to the degradation of the model performance, but also greatly increase the training time and testing time, which verifies the feature selection and dimensionality process in our project is essential.

5.2 Explanation of the poor performance of Naïve Bayes and Random Forest

For Naïve Bayes, the most likely reason for its poor performance is that there is high dependence in our features which is against the assumption of Naïve Bayes that all the features should be independent from each other. It makes sense because most of the features were extracted by the same music analyzation algorithms which may be highly dependent on each other. The result of PCA also verifies this since we only needed 121 principal components to explain 90% variance of our data (originally 774 features before PCA).

For Random Forest, we were surprised at its poor performance since it is a widely accepted fact that it should perform very well in multi-class classification problem. Hence, we made detailed testing and found it suffered from the overfitting issue since there was a big gap between the training error and the validation error.

```
Hyperparameters:  {'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.4313223441179603
Mean training score (macro f1):  0.7891674459696352
Hyperparameters:  {'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.43179360326267036
Mean training score (macro f1):  0.7897358215171197
Hyperparameters:  {'max_depth': 10, 'max_features': 'log2', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.3522367213532726
Mean training score (macro f1):  0.709317425952343
Hyperparameters:  {'max_depth': 10, 'max_features': 'log2', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.3411578285134075
Mean training score (macro f1):  0.709661512158477
Hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.4814380419392846
Mean training score (macro f1):  0.9995878538185895
Hyperparameters:  {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.4851124624572657
Mean training score (macro f1):  0.9997194507635061
Hyperparameters:  {'max_depth': 20, 'max_features': 'log2', 'n_estimators': 200}
Mean cross-validation score (macro f1):  0.4355610056586614
Mean training score (macro f1):  0.999849858193494
Hyperparameters:  {'max_depth': 20, 'max_features': 'log2', 'n_estimators': 500}
Mean cross-validation score (macro f1):  0.44191232410661146
Mean training score (macro f1):  0.9998144536017733
```

Fig.24 Identifying overfitting in Random Forest

Finally, we added some more restrictions such as “max_depth”, “min_samples_leaf”, and “min_samples_split” to decrease the model complexity of the Random Forest. The table below demonstrates that restricting the model complexity of the Random Forest can significantly upgrade the model performance.

Model	Training Time	Testing Time	Accuracy	Precision	Recall	F1-Score (macro)
Random Forest (without restriction)	67.0917	0.4688	66.33%	0.7394	0.4719	0.4929
Random Forest (with restriction)	26.3539	0.3492	69.43%	0.6897	0.6125	0.6151

6 Conclusion/Future Work

In conclusion, our project demonstrated that machine learning techniques, combined with feature selection and dimensionality reduction, can effectively predict music genres. Among all the models tested, SVM and Multiple Layer Perceptron achieved the highest accuracy, while Naïve Bayes and Random Forest failed. We further verified the importance of feature selection and dimensionality reduction techniques and tried to explain the reasons for the poor performance of Naïve Bayes and Random Forest.

For future work, some technics like “SOMTE” could be used to make the dataset more balanced to improve the performance of the models. Other feature selection and dimensionality reduction methods could be explored to evaluate their effectiveness, and the dataset could be diversified to include more music genres and a broader range of audio features.

Overall, the project provided valuable insights into the potential of machine learning techniques for predicting music genres and highlighted the importance of algorithm selection, feature selection, and dimensionality reduction in improving the task fulfillment. Future work in this area could have significant practical applications in music recommendation, music retrieval, and related fields.

7 References

[1] G. Tzanetakis and P. R. Cook, "Musical genre classification of audio signals," in IEEE Transactions on Speech and Audio Processing, vol. 10, no. 5, pp. 293-302, July 2002, doi: 10.1109/TSA.2002.800560.

[2] R. Ghode, P. Navale, M. Jadhav, A. Chippa, and M. Bhandare, "Music Genre Classification and Recommendation," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 7, no. 6, pp. 298-301, Nov.-Dec. 2021, doi: 10.32628/CSEIT217684.

[3] A. ELBİR Et Al., "Music Genre Classification and Recommendation by Using Machine Learning Techniques," 2018 Innovations in Intelligent Systems and Applications Conference, ASYU 2018, Adana, Turkey, 2018, pp. 1-5, doi: 10.1109/ASYU.2018.8554016.

[4] A. Elbir and N. Aydin, "Music Genre Classification and Music Recommendation by Using Deep Learning," Electronics Letters, vol. 56, Mar. 2020, doi: 10.1049/el.2019.4202.

[5] K. S. Mounika, S. Deyaradevi, K. Swetha and V. Vanitha, "Music Genre Classification Using Deep Learning," 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA), Coimbatore, India, 2021, pp. 1-7, doi: 10.1109/ICAECA52838.2021.9675685.

[6] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, "FMA: A Dataset For Music Analysis," arXiv preprint arXiv:1612.01840, Dec. 2017, cs.SD. [Online]. Available: https://arxiv.org/abs/1612.01840, doi: https://doi.org/10.48550/arXiv.1612.01840

[7] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," arXiv preprint arXiv:1310.4546, Oct. 2013, cs.CL. [Online]. Available: https://arxiv.org/abs/1310.4546. doi: 10.48550/arXiv.1310.4546.

[8] P. F. Muhammad, R. Kusumaningrum, and A. Wibowo, "Sentiment Analysis Using Word2vec And Long Short-Term Memory (LSTM) For Indonesian Hotel Reviews," Procedia Computer Science, vol. 179, pp. 728-735, 2021. doi: https://doi.org/10.1016/j.procs.2021.01.061.

[9] Scikit-learn, "GridSearchCV," in Scikit-learn: Machine Learning in Python (1.2.2). [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed on: Apr. 17, 2023.

[10] S. Suthaharan, Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning, Integrated Series in Information Systems, 1st ed. New York, NY, USA: Springer, 2015, pp. XIX, 359. doi: https://doi.org/10.1007/978-1-4899-7641-3.

[11] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," WIREs Comput. Stat., vol. 4, no. 3, pp. 275-294, 2012. doi: 10.1002/wics.1198.

[12] S. Sperandei, "Understanding logistic regression analysis," Biochem. Med. (Zagreb), vol. 24, no. 1, pp. 12-18, Feb. 2014. doi: 10.11613/BM.2014.003. PMID: 24627710; PMCID: PMC3936971.

[13] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN model-based approach in classification," in On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, R. Meersman, Z. Tari, and D. C. Schmidt, Eds. Berlin, Heidelberg: Springer, 2003, vol. 2888, pp. 624-631. doi: 10.1007/978-3-540-39964-3_62.

[14] G. Webb, "Naïve Bayes," in Encyclopedia of Machine Learning and Data Mining, C. Sammut and G. I. Webb, Eds. MA: Springer US, 2017, pp. 1-2. doi: 10.1007/978-1-4899-7502-7_581-1.
[15] S. Srivastava, M. R. Gupta, and B. A. Frigyik, "Bayesian quadratic discriminant analysis," Journal of Machine Learning Research, vol. 8, pp. 1277-1305, Jun. 2007. [Online]. Available: http://www.jmlr.org/papers/volume8/srivastava07a/srivastava07a.pdf.

[16] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," Atmospheric Environment, vol. 32, no. 14, pp. 2627-2636, 1998. doi: 10.1016/S1352-2310(97)00447-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1352231097004470.

[17] J. T. Hancock and T. M. Khoshgoftaar, "CatBoost for big data: An interdisciplinary review," Journal of Big Data, vol. 7, no. 1, p. 94, Nov. 2020. doi: 10.1186/s40537-020-00369-8. [Online]. Available: https://doi.org/10.1186/s40537-020-00369-8.

[18] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 2016, pp. 785-794. doi: 10.1145/2939672.2939785.