Music Genre Classification via Machine Learning

Group 29: Nianlin Chen, Jiaqi Zhu, Yan Xu

April 22, 2022

Abstract

With the faster-evolving music industry, the density of different music genres is increasing and the classification of music genres is complicated and labor-intensive, which requires reducing labor costs and improving efficiency. This project aims to classify multiple music genres based on different audio features via machine learning methods. Within the project, after performing data prepossessing and exploratory data analysis, three machine learning methods were applied and the overall testing accuracy rate for classifying rock, instrumental, and hip hop music genres is higher than 93%. Moreover, PCA was also performed for understanding the similarity between rock and hip hop music and the dissimilarity of instrumental music. This project also shows that simple methods, such as KNN, outperforms complicated ones, such as AdaBoost. Besides class characteristics, the impact of imbalanced data on classification is shown in this project, where the low volume class got overrode by the high volume class and resulted in a low accuracy rate for smaller class. Bootstrap sampling can be a potential resolution to deal with imbalanced classes. For further studies, models can be trained and performed on similar music genres, and thus contribute more to the music classification.

Keywords: Classification, KNN, Random forest, AdaBoost, PCA

1 Motivation and Problem

Nowadays, with the widely using of portable devices and the blooming of social medial and promptly raise in density of music genres, music industry is evolving rapidly and music genres is developing and fusing with each other. Music genres classification is required for better comprehension of the music industry and future applications. Automation of this process can reduce time and labor cost and improve efficiency.

Thus, within this project, our main goal is to utilize machine learning methods to classify music genres based on multiple audio features. Considering the purpose of automating the classify process, the main goal will be classification rather than interpretation, and high accuracy rate will be pursuing.

2 Data

2.1 Data Description

The data set used in this project is from Kaggle and it was acquired from one of the Machine Hack Hackathons. The data set is separated into training and testing data in 8 to 2 ratio, with 17996 observations in total. There are 17 variables in the training data set, which includes the target variable: class, 11 types of music genres (Acoustic Folk Genre, Alt Music Genre, Blues Genre, Bollywood Music Genre, Country Music Genre, Hip Hop Music Genre, Indie Music Genre, Instrumental Music Genre, Metal Music Genre, Pop Music Genre, Rock Music Genre) denoted from 0 to 10; Artist name; track name; popularity - numerical variables scored from 0 to 100; danceability, energy, speechiness, acousticness, instrumentalness, liveness, valence - numerical variables scored from 0 to 1; tempo; duration in milliseconds; time signature - categorical variables with 4 levels; Key; loudness; mode - a categorical variable with two levels 0 and 1. The testing data have identical variables except the target predicted variable class is removed.

2.2 Exploratory Data Analysis

In the preliminary analysis, the data set has 4377 missing values in variable instrumentalness, 2014 in key, and 428 in popularity. Considering the large size of the data set and the difficulty of reasonably filling missing values in certain variables, observations with missing values are dropped. One potential issue for the data set is highly imbalanced given the proportions of various music classes, with more than 28% of rock music genres with 3374 observations, 17% of indie music genre with 2039 observations, 12% of metal music genres with 1523 observations, and 8 other music genres with less than 1500 observation in each class.

To understand more about characteristics of each class, the box plot for numerical variables is drawn (shown in Figure 1). Based on the boxplots, class 5, 7, 10 have obvious difference for multiple attributes. Therefore, within this project, we will concentrate on these three classes, which are Hip Hop, Instrumental, and Rock music, with 464, 517, and 3374 obs respectively. The training set we eventually use in the project with 4355 observations in total.

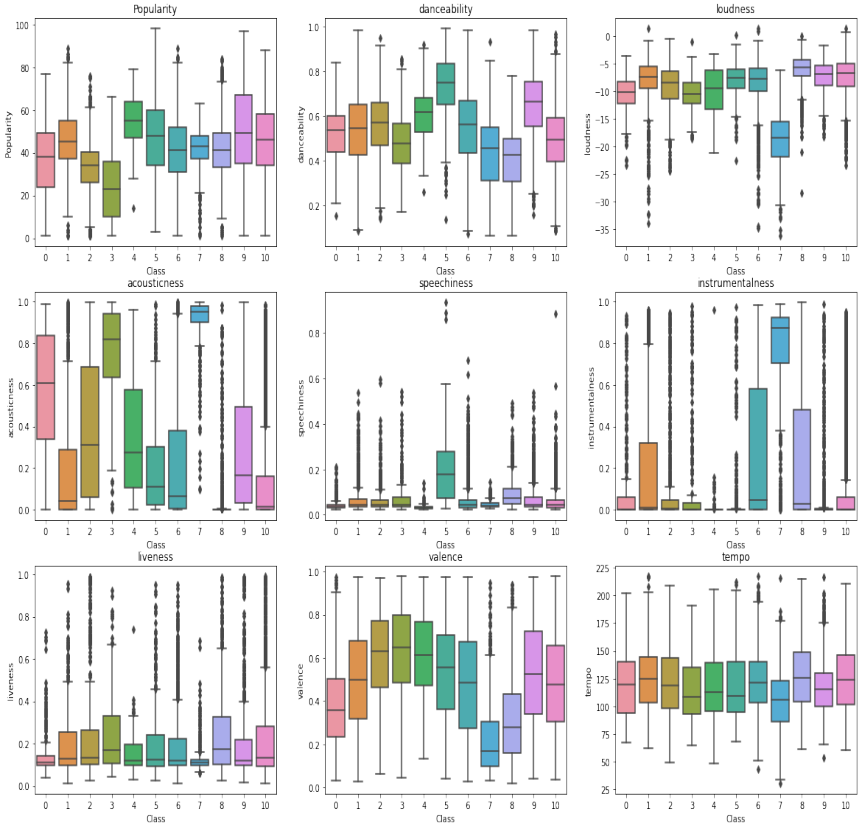


Figure 1: Boxplot for some numerical variables of all classes

To check potential ouliers, Figure 2 below displayed a more clear comparison of the three music genres regarding selected continuous features. Instrumental music genre is the most distinctive one among the three music group, and it shows low rating in loudness and valence, while high rating in acousticness and instrumentalness. However, Hip hop and Rock music genres both have similar ratings in loudness, valence, acousticness, but varied in danceability, and speechiness scores. All these presented in the boxplots are consistent with facts of these music genres and proves that the data is trustworthy. Besides, for numerical variables shown in the plot, most of them are ranging from 0 to 1 to describe one feature of the music, however, popularity is ranging from 1 to 100, loudness is ranging from -40 to 1.35, and tempo is ranging from 30to 220. For further research, we normalize 11 numerical variables.

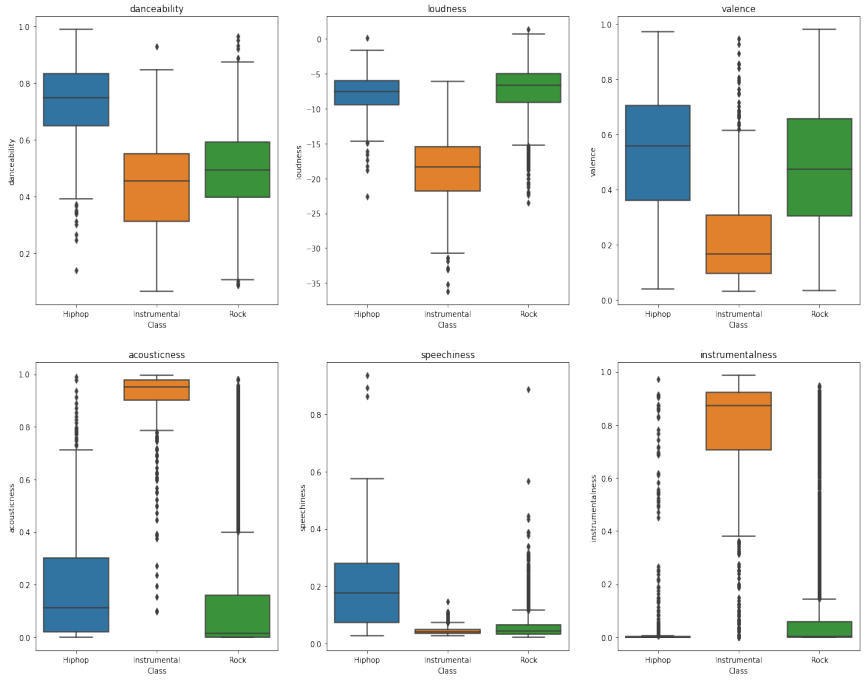


Figure 2: Boxplot for some numerical variables of three classes

To have some motivations regarding the structure of our model, figure 3 below provides the distribution of several continuous variables. Popularity and danceability are approximately normally distributed, while energy and loudness are left skewed, and acousticness and instrumental are extremely right skewed. There is no much improvement after doing log-transformation. Thus, some parametric classifiers such as LDA and QDA that require normal assumptions might not suit this data set.

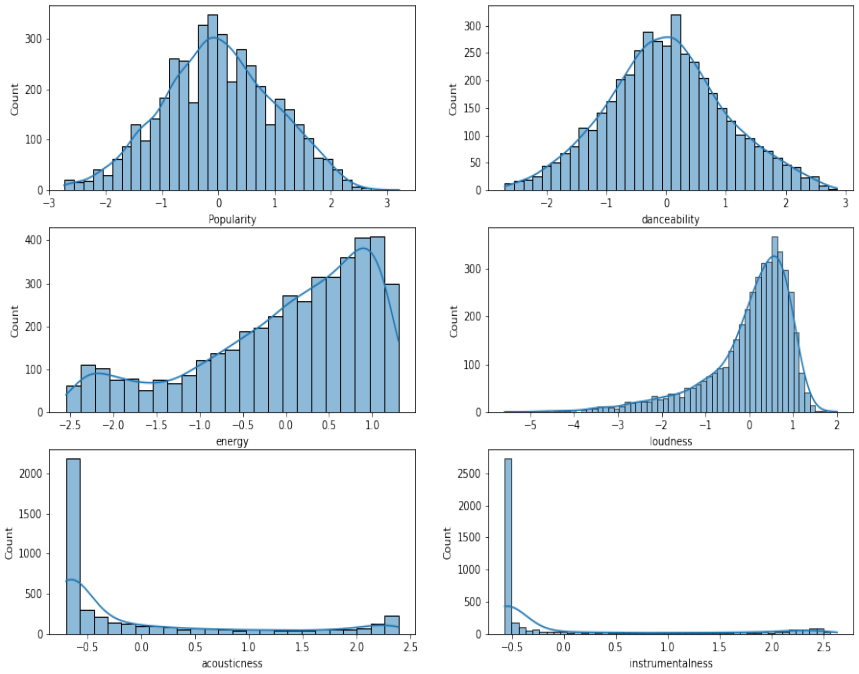


Figure 3: Distribution for some numerical variables

2.3 Correlation Analysis

For the numerical variables, we first use heat map (shown in Figure 4) to investigate the correlation. It is observed that there is a high correlation between energy and loudness. Energy and acousticness are negative correlated, loudness and acousticness are negative correlated as well, which makes sense a lot.

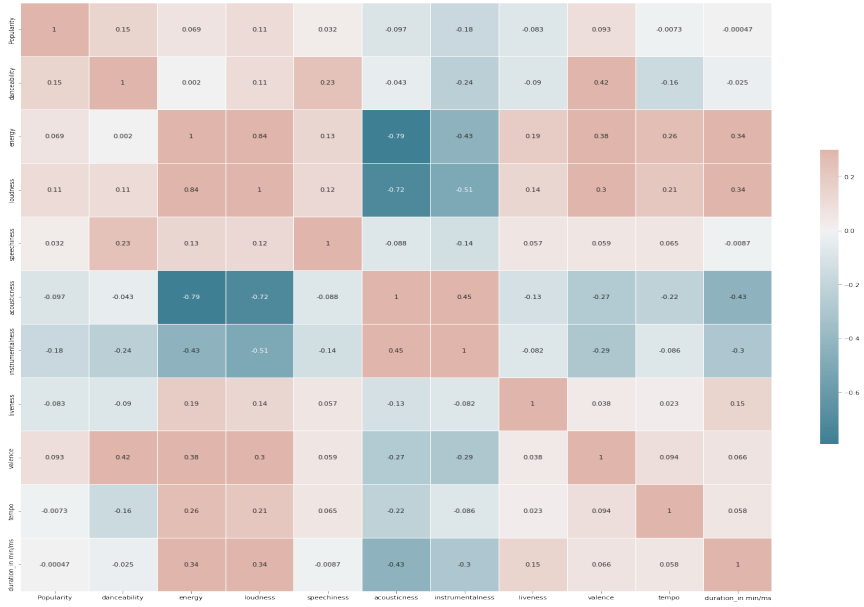


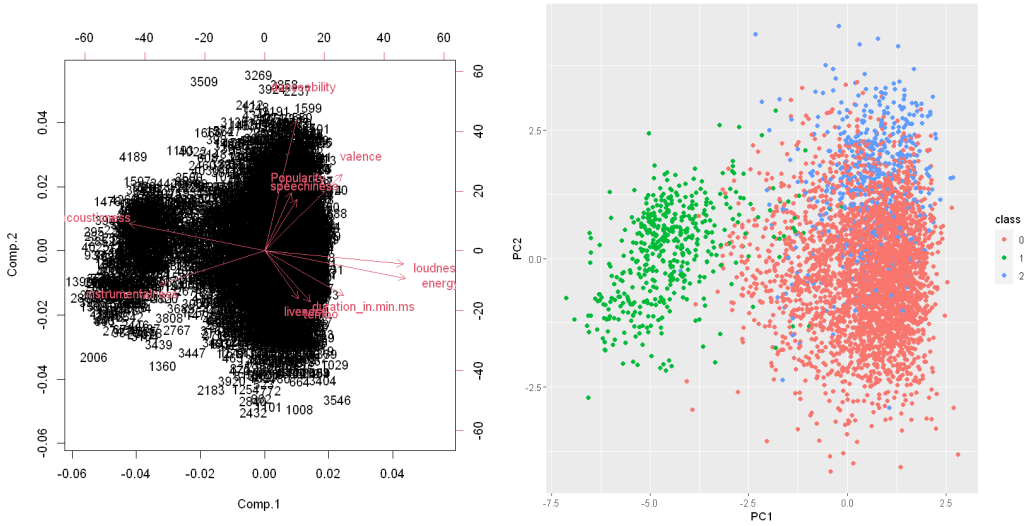
Figure 4: Heat map for numerical variables

2.4 PCA

As there are 11 numerical variables contained in the data set, we perform dimension reduction method via Principal Component Analysis (PCA) and get insights into variables and three classes' attributions. From the biplot of PCA shown in Figure 5a, variables of loudness and energy have a strong high positive correlation, as well as variables of danceability, popularity, speechiness, and valence have positive correlations. Also, variables of loudness and acoustiouss have a high negative correlation, which is also verified in previous correlation analysis section discussed above. The first principal component direction can be interpreted as the difference between the sum of energy, loudness, liveness, tempo, and the sum of acousticness and instrumentalness. The second principal component direction is the sum of

danceability, popularity, speechiness, and valence. Therefore, we can interpret PC1 as music with high energy but low instrumental components, and PC2 as music with high danceability and valence.

Figure 5b provides the visualization of the training set projected into the first two principal component directions. In this plot, classes 0, 1, and 2 stand for rock, instrumental, and hip hop music genres respectively. From the PC1 direction, Instrumental music can be clearly separated from Rock music and Hip Hop music. One applaudable reason for that is Instrumental music tends to be less intensive than the others. However, for Rock music and Hip Hop music genres, they are sometimes similar in intensity but different in rhythm, so they are similar distributed from PC1 direction but distinctive from PC2, thus the boundary between these two classes is fuzzy. Moreover, due to much larger data volume of Rock music genre, the most of the data points from Hip Hop music genre are overrode from the plot.



(a) Biplot for PCA (b) Clusters in PCA

Figure 5: Plot for PCA

3 Method

3.1 KNN

KNN is a simple and useful non-parametric classifier, which allows us to implement to data set without assumptions of distributions. It is robust to

multi-class classification. Also, it is not sensitive to the imbalanced data set. Therefore, we choose to carry out the KNN to train the data set first. One potential issue for KNN is that it is sensitive to outliers which might cause accurate problems. We tuned the parameters by using 10-fold cross-validation, and K = 5 is observed to provide the best performance on the testing set with the highest accuracy rate.

3.2 Random Forest

To classify 3 music genres accurately, the model of classification tree is a simple but often unstable method, since small changes in the data set can result in very different tree models. Random forest improves stability of decision trees by bagging them since it decorrelates the decision trees with the introduction of splitting on a random set of features. Taking the average of a large number of tree models can make tree models more steady, where each tree makes use of different simulated data sets generated by sampling with replacement from the original data set.

After tuning parameter using cross validation to minimize the testing error, it achieves a balance between goodness-of-fit and interpretation. The number of trees in our model is 25, and the criterion is entropy. The minimum number of samples required to split an internal node and to be at a leaf node is 2 and 1, respectively.

3.3 AdaBoost

Classification of Rock, Hip-Hop, and instrumental music genres is obviously not a binary classification problem, where a tree model of Adaboost is an appropriate choice. AdaBoost can boost performance of any machine learning algorithms. One may propose a small tree model using the whole data set as a grounded model, initially assigning equal weights to all the sample points. However, one potential issue about AdaBoost is that it is susceptible to over-fitting outliers due to the form of its loss function.

AdaBoost achieves the best performance on testing set by tuning parameters using cross-validation. We set weight applied to each classifier at each boosting iteration equal to 0.8. The maximum number of estimators at which boosting is terminated is 25.

4 Result Analysis

4.1 Accuracy Comparison

Figure 6 below displays the results of three machine learning methods. All of the three models have test accuracy rates higher than 90%. The best model based on accuracy score is Random forest, with an accuracy rate of 97.24%. KNN also has an accuracy rate higher than 95%, and the accuracy rate for AdaBoost is 91.93%.

As for each music genre, both rock and instrumental music genres have high accuracy over 80%. However, the test accuracy rate for the hip hop music genre is around 50%. One possible reason for this situation is that the data sets are highly imbalanced and each group has significantly various proportions. The instances belonging to the smaller class(hip hop 464) are typically misclassified more often than those belonging to the larger class (rock 3374), which explains the lowest accuracy score of the hip hop group. Besides that, from the data points projected on the first two principal component directions(shown in Figure 5b), it is observed that hip hop group and rock group overlap with each other from the first PC direction, which might also cause the relatively lower accuracy rate for hip hop group.

From the overall testing accuracy rate (shown in Figure 6), Random Forest gives the best performance on the testing set. Therefore, we decide to choose Random Forest for further research and analysis.

Methods	Overall Training	Overall Testing	Rock	Instrumental	Hip Hop
Random Forest	93.92%	97.24%	99.26%	97.85%	55.34%
KNN	93.23%	95.01%	98.52%	98.92%	53.40%
AdaBoost	91.16%	91.93%	98.81%	84.95%	46.60%

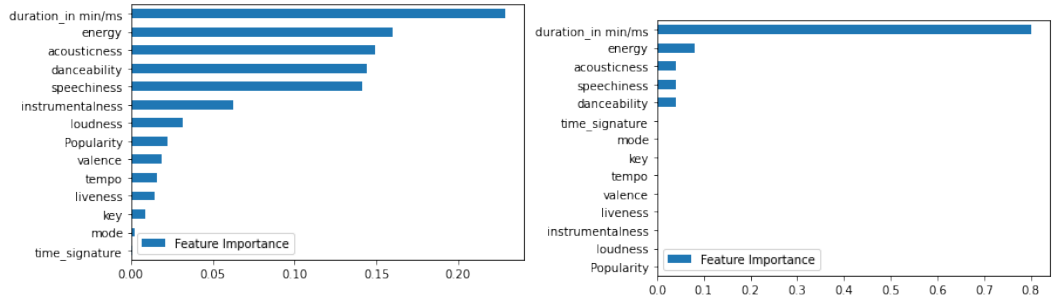
Figure 6: Model performance

4.2 Best Model Analysis

For the model random forest, Figure 7a presented that the variable of duration in ms is the most important for the model fitting. This might be because of the significant differences among the lengths of different genres of music. Instrumental music is typically short for the most of time, but for rock music, there are songs with multiple choruses and thus longer than other music genres. Besides the length of music, energy, acousticness, danceability, and speechiness are also important variables in classifying rock, instrumental, and hip hop music groups using Random Forest. This is consistent with

the fact that these three music genres are significantly different in those perspectives.

The relative influence plot for AdaBoost is shown in Figure 4b. The variable of duration in ms dominates is the most essential one, with about 80% influence toward the AdaBoost model. This is consistent with the plots of random forest(shown in Figure 7a) and suggests the importance of the length of music in classifying rock, hip hop, and instrumental music genres.



(a) Variable importance for Random Forest

(b) Relative influence for AdaBoost

Figure 7: Variable Importance

5 Conclusions and Discussion

The purpose of this project is to classify multiple music genres based on different audio features using several different methods including Random Forest, KNN, and AdaBoost. To the results presented above, The predictive performance of Random forest is quite nice with the highest overall testing accurate rate for classifying Rock, Instrumental, and Hip Hop music genres.

The data is imbalanced and the data volume of Rock music is significantly more than that of Hip Hop music(around 7 times), which poses a challenge to classification music groups, and especially, it would lead to poor performance to the minority class separation. Therefore, we consider this as the main reason for the lowest accuracy score of Hip Hop music genre compared with the Rock music genre while they are similar in several features, such as high energy and loudness.

Another concern of the models is that although overall accuracy rates are relatively high for all three models, there are chances of over-fitting. Possible solutions to resolve over-fitting are to train the model with more data and remove correlated features, and this will also stabilize the model.

In future research, more similar music genres should be trained on the model. Due to the time limitation, only classes with distinct features are included in this project, which helps to separate various music genres clearly. However, separating similar music genres is an essential aspect of music classification in real-world cases. Hence, the model can be improved by tuning hyper-parameters with a larger data set. Within this project, we did not use PCs after PCA as predictors for the model, still, variable selection and feature engineering can be done in the future to reduce over-fitting and stabilize the model.

6 Contribution

Throughout the project, the contribution of each group member is quite even. We discuss and analyze all problems together, process and clean the data, implementing models together.

Nianlin Chen: Write out Data Description, Correlation analysis, and Method part, and finalize and polish the report.
Jiaqi Zhu: Write out Abstract, Introduction, EDA, PCA, results, and conclusions parts.
Yan Xu: Prepared the verbal presentation, and help check with the written report.

References

[1]https://www.kaggle.com/datasets/purumalgi/music-genre-classification/code

A Appendix

Variable name	Data Type	Description
Popularity	continuous	score of popularity
danceability	continuous	score of popularity
energy	continuous	score of energy
loudness	continuous	score of loudness
speechness	continuous	score of speechness
acousticness	continuous	score of acousticness
instrumentalness	continuous	score of instrumentalness
liveness	continuous	score of liveness
valence	continuous	score of valence
tempo	continuous	tempo of the song
duration in ms	continuous	length of the song
Artist Name	categorical	artist name of the music
Track Name	categorical	track name of the music
key	categorical	key of the music
mode	categorical	mode of the music
time signature	categorical	time signature
Class	categorical	genre the music belongs to

Table 1: Variable List

```
library(ggplot2)
library(tidyverse)
library(GGally)
library(ggpubr)
library(kernlab)
music1_scale=read.csv("music_scale.csv")
head(music1_scale)
dim(music1_scale)
x=music1_scale[,c(2:12)]
x=as.matrix(x)
music1_scale$key=as.factor(music1_scale$key)
music1_scale$mode=as.factor(music1_scale$mode)
music1_scale$time_signature=as.factor(music1_scale$time_signature)
music1_scale$class=as.factor(music1_scale$class)
head(x)
x.pca_cov = prcomp(x,scale=FALSE)
summary(x.pca_cov)
screeplot(x.pca_cor,type = 'l')
x_pca_cov = princomp(x,cor=F)
x_pca_cov$loadings
x_pca_cov = princomp(x,cor=F)
plt1=biplot(x_pca_cov)
# Data Projection on the First Two PCs
pca_projections <- as.data.frame((x_pca_cov$scores)[, 1:2])
colnames(pca_projections) <- c("PC1", "PC2")
pca_projections <- cbind(pca_projections, class = music1_scale[, c("class")])
ggplot(data = pca_projections) +
  geom_point(aes(x = PC1, y = PC2, col = class))
pca_projections <- as.data.frame((x_pca_cov$scores)[, 1:2])
colnames(pca_projections) <- c("PC1", "PC2")
pca_projections <- cbind(pca_projections, mode = music1_scale[, c("mode")])
ggplot(data = pca_projections) +
  geom_point(aes(x = PC1, y = PC2, col = mode))

pca_projections <- as.data.frame((x_pca_cov$scores)[, 1:2])
colnames(pca_projections) <- c("PC1", "PC2")
pca_projections <- cbind(pca_projections, time_signature = music1_scale[,
c("time_signature")])
ggplot(data = pca_projections) +
  geom_point(aes(x = PC1, y = PC2, col = time_signature))

pca_projections <- as.data.frame((x_pca_cor$scores)[, 1:2])
colnames(pca_projections) <- c("PC1", "PC2")
pca_projections <- cbind(pca_projections, key = music1_scale[, c("key")])
ggplot(data = pca_projections) +
  geom_point(aes(x = PC1, y = PC2, col = key))
```

```
#!/usr/bin/env python
# coding: utf-8



import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

music = pd.read_csv('train.csv')

music.isnull().sum()


music = music.dropna()
music.shape

music.describe(include = 'all')


music.groupby('Class').size()


l = list(range(0,11))
labels = ['Hiphop', 'Instrumental', 'Rock']
music1 = music[music['Class'].isin(l)]

fig, ax = plt.subplots(ncols = 3,nrows = 3, figsize=(20, 15))
g = sns.boxplot(data = music1, x = 'Class', y='Popularity', ax = ax[0,0])
g.set_title('Popularity')
#g.set_xticklabels(labels)
f = sns.boxplot(data = music1, x = 'Class', y='danceability', ax = ax[0,1])
f.set_title('danceability')
#f.set_xticklabels(labels)
g = sns.boxplot(data = music1, x = 'Class', y='loudness', ax = ax[0,2])
g.set_title('loudness')
#g.set_xticklabels(labels)
f = sns.boxplot(data = music1, x = 'Class', y='acousticness', ax = ax[1,0])
f.set_title('acousticness')
#f.set_xticklabels(labels)
g = sns.boxplot(data = music1, x = 'Class', y='speechiness', ax = ax[1,1])
g.set_title('speechiness')
#g.set_xticklabels(labels)
f = sns.boxplot(data = music1, x = 'Class', y='instrumentalness', ax = ax[1,2])
f.set_title('instrumentalness')
#f.set_xticklabels(labels)
g = sns.boxplot(data = music1, x = 'Class', y='liveness', ax = ax[2,0])
g.set_title('liveness')
#g.set_xticklabels(labels)
f = sns.boxplot(data = music1, x = 'Class', y='valence', ax = ax[2,1])
f.set_title('valence')
#f.set_xticklabels(labels)
f = sns.boxplot(data = music1, x = 'Class', y='tempo', ax = ax[2,2])
f.set_title('tempo')
#f.set_xticklabels(labels)
plt.show()


l = [5, 7, 10]
labels = ['Hiphop', 'Instrumental', 'Rock']
music1 = music[music['Class'].isin(l)]

fig, ax = plt.subplots(ncols = 3,nrows = 2, figsize=(20, 15))
f = sns.boxplot(data = music1, x = 'Class', y='duration_in min/ms', ax = ax[0,0])
f.set_title('duration_in min/ms')
f.set_xticklabels(labels)

g = sns.boxplot(data = music1, x = 'Class', y='loudness', ax = ax[0,1])
g.set_title('loudness')
g.set_xticklabels(labels)

f = sns.boxplot(data = music1, x = 'Class', y='acousticness', ax = ax[1,0])
f.set_title('acousticness')
f.set_xticklabels(labels)

g = sns.boxplot(data = music1, x = 'Class', y='speechiness', ax = ax[1,1])
g.set_title('speechiness')
g.set_xticklabels(labels)

f = sns.boxplot(data = music1, x = 'Class', y='instrumentalness', ax = ax[1,2])
f.set_title('instrumentalness')
f.set_xticklabels(labels)

f = sns.boxplot(data = music1, x = 'Class', y='valence', ax = ax[0,2])
f.set_title('valence')
f.set_xticklabels(labels)
```

```
plt.show()


music.loc[music.Class == 10, 'class'] = 0 # Rock Music
music.loc[music.Class == 7, 'class'] = 1 # Instrumental Music Genre
music.loc[music.Class == 5, 'class'] = 2 # HipHop Music Genre
# music.loc[music.Class == 9, 'class'] = 1 # Pop
# music.loc[music.Class == 8, 'class'] = 2 # Metal Music Genre
# music1_scale.loc[music1_scale.Class == 9, 'class'] = 3
# music1_scale.loc[music1_scale.Class == 1, 'class'] = 4
# music1_scale['class'].fillna(3,inplace = True)


music = music.dropna()
music = music.drop('Class', axis=1)


music.describe()


music.groupby('class').size()


music_cate = music[['key', 'mode', 'time_signature', 'class']]
music_cate.head()

music1 = music.drop(columns=['Artist Name','Track Name','mode','class','time_signature','key'])
music1.head()

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
music1_scale = pd.DataFrame(sc.fit_transform(music1), columns=music1.columns)

music1_corr = music1_scale.corr()


f, ax = plt.subplots(figsize=(25,20))
cmap = sns.diverging_palette(220, 20, as_cmap=True)
sns.heatmap(music1_corr, cmap=cmap, vmax=.3, center=0, annot=True,square=True, linewidths=.5, cbar_kws={'shrink': .5})
plt.show()

music1_scale.head()


music1_scale['class'] = music['class'].tolist()
music1_scale['key'] = music['key'].tolist()
music1_scale['mode'] = music['mode'].tolist()
music1_scale['time_signature'] = music['time_signature'].tolist()
music1_scale

fig, ax = plt.subplots(ncols = 1,nrows = 3, figsize=(5, 15))
g = sns.boxplot(data = music1_scale, x = 'class', y='Popularity', ax = ax[0])
g.set_title('Popularity')
#f = sns.boxplot(data = music1_scale, x = 'class', y='danceability', ax = ax[0,1])
#f.set_title('danceability')
g = sns.boxplot(data = music1_scale, x = 'class', y='loudness', ax = ax[1])
g.set_title('loudness')
f = sns.boxplot(data = music1_scale, x = 'class', y='acousticness', ax = ax[2])
f.set_title('acousticness')
plt.show()


fig, ax = plt.subplots(ncols = 2,nrows = 3, figsize=(15, 10))
g = sns.histplot(data = music1_scale, x = 'Popularity',kde=True,ax = ax[0,0])
g.set(xlim=(-3,3.5))
sns.histplot(data = music1_scale, x = 'danceability',kde=True,ax = ax[0,1])
sns.histplot(data = music1_scale, x = 'energy',kde=True,ax = ax[1,0])
sns.histplot(data = music1_scale, x = 'loudness',kde=True,ax = ax[1,1])
sns.histplot(data = music1_scale, x = 'acousticness',kde=True,ax = ax[2,0])
sns.histplot(data = music1_scale, x = 'instrumentalness',kde=True,ax = ax[2,1])
plt.show()


# - logistic regression
# - KNN
# - decision tree
# - rf
# - NN
# - svm
# - pca
# - sdm
# - hierarchical clustering



from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(music1_scale, music1_scale["class"]):
    strat_train_set = music1_scale.iloc[train_index]
    strat_test_set = music1_scale.iloc[test_index]
```

```
strat_train_set = strat_train_set.reset_index(drop = True)
strat_test_set = strat_test_set.reset_index(drop = True)
```

```
train_X = strat_train_set.drop('class', axis = 1)
test_X = strat_test_set.drop('class', axis = 1)
train_y = strat_train_set['class']
test_y = strat_test_set['class']
```

```
train_y.unique()
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
# ### KNN
```

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)

knn_model.fit(train_X, train_y)
pred_y = knn_model.predict(test_X)
train_knn = knn_model.score(train_X, train_y)
test_knn = knn_model.score(test_X, test_y)
```

```
from sklearn.model_selection import GridSearchCV

grid_clf_knn = GridSearchCV(cv=10, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'n_neighbors': [1, 3, 5, 10, 20]})
grid_clf_knn.fit(train_X, train_y)
```

```
grid_clf_knn.best_estimator_
```

```
y_pred = grid_clf_knn.predict(test_X)

train_knn = grid_clf_knn.score(train_X, train_y)
test_knn = grid_clf_knn.score(test_X, test_y)
```

```
grid_clf_knn.best_estimator_
```

```
m_knn = confusion_matrix(test_y, y_pred)
knn_byclass = m_knn.diagonal()/m_knn.sum(axis=1)
knn_byclass = knn_byclass.tolist()
knn_byclass
```

```
knn_byclass[1]
```

```
# ### Decision tree
```

```
from sklearn.tree import DecisionTreeClassifier

tr_model = DecisionTreeClassifier(
                random_state=42,
                criterion='entropy',
                splitter='best',
                max_depth=5,
                min_samples_split=2)

tr_model.fit(train_X, train_y)
y_pred = tr_model.predict(test_X)

train_tr = tr_model.score(train_X, train_y)
test_tr = tr_model.score(test_X, test_y)
```

```
train_tr
```

```
test_tr
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
                'min_samples_split': [2, 3, 4, 5, 6, 7, 8],
                'max_depth': [2, 5, 7, 9],
             }

grid_clf_tr = GridSearchCV(tr_model, param_grid, cv=10, n_jobs=-1, refit = True)
grid_clf_tr.fit(train_X, train_y)

y_pred = grid_clf_tr.predict(test_X)

train_tr = grid_clf_tr.score(train_X, train_y)
test_tr = grid_clf_tr.score(test_X, test_y)
```

```
grid_clf_tr.best_estimator_
```

```
train_tr
```

```
test_tr
```

```
m_tr = confusion_matrix(test_y, y_pred)
tr_byclass = m_tr.diagonal()/m_tr.sum(axis=1)
tr_byclass = tr_byclass.tolist()
tr_byclass
```

```
# ### Random forest
```

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    criterion='entropy',
    max_depth=8,
    min_samples_split=2)

rf_model.fit(train_X, train_y)
y_pred = rf_model.predict(test_X)

train_rf = rf_model.score(train_X, train_y)
test_rf = rf_model.score(test_X, test_y)
```

```
train_rf
```

```
test_rf
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
                'n_estimators': [5, 10, 15, 20, 25, 30, 35, 40, 50, 100],
                'max_depth': [2, 3, 4, 5, 7, 8],
```

```
            }

grid_clf_rf = GridSearchCV(rf_model, param_grid, cv=10, n_jobs=-1, refit = True)
grid_clf_rf.fit(train_X, train_y)

y_pred = grid_clf_rf.predict(test_X)

train_rf = grid_clf_rf.score(train_X, train_y)
test_rf = grid_clf_rf.score(test_X, test_y)
```

```
grid_clf_rf.best_estimator_
```

```
m_rf = confusion_matrix(test_y, y_pred)
rf_byclass = m_rf.diagonal()/m_rf.sum(axis=1)
rf_byclass = rf_byclass.tolist()
rf_byclass
```

```
model = RandomForestClassifier(
    criterion='entropy', max_depth=8, n_estimators=25,
    random_state=42)

model.fit(train_X,train_y)
y_pred = model.predict(test_X)

feat_importance = model.feature_importances_
pd.DataFrame({'Feature Importance':feat_importance},
             index=list(train_X)).sort_values('Feature Importance').plot(kind='barh')
```

```
# ### Logistic regression
```

```
from sklearn.linear_model import LogisticRegression

log_model = LogisticRegression(solver='liblinear',multi_class="auto")
log_model.fit(train_X, train_y)
y_pred = log_model.predict(test_X)

train_log = log_model.score(train_X, train_y)
test_log = log_model.score(test_X, test_y)
```

```
m_log = confusion_matrix(test_y, y_pred)
log_byclass = m_log.diagonal()/m_log.sum(axis=1)
log_byclass = log_byclass.tolist()
log_byclass
```

```
# ### AdaBoost
```

```
from sklearn.ensemble import AdaBoostClassifier

adb_model = AdaBoostClassifier(
    n_estimators=150,
    learning_rate=0.4,
    random_state=42)

adb_model.fit(train_X, train_y)
y_pred = adb_model.predict(test_X)

train_adb = adb_model.score(train_X, train_y)
test_adb = adb_model.score(test_X, test_y)
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {
                'n_estimators': [10, 15, 20, 25, 30, 35, 40, 50, 100, 150],
                'learning_rate': [0.2, 0.4, 0.5, 0.6, 0.8, 1],
             }

grid_clf_adb = GridSearchCV(adb_model, param_grid, cv=10, n_jobs=-1, refit = True)
grid_clf_adb.fit(train_X, train_y)
```

```
y_pred = grid_clf_adb.predict(test_X)

train_adb = grid_clf_adb.score(train_X, train_y)
test_adb = grid_clf_adb.score(test_X, test_y)
```

```
grid_clf_adb.best_estimator_
```

```
m_ada = confusion_matrix(test_y, y_pred)
ada_byclass = m_ada.diagonal()/m_ada.sum(axis=1)
ada_byclass = ada_byclass.tolist()
ada_byclass
```

```
model = AdaBoostClassifier(learning_rate=0.8, n_estimators=25, random_state=42)

model.fit(train_X,train_y)
y_pred = model.predict(test_X)

feat_importance = model.feature_importances_
pd.DataFrame({'Feature Importance':feat_importance},
             index=list(train_X)).sort_values('Feature Importance').plot(kind='barh')
```

```
from sklearn.ensemble import VotingClassifier


clf1 = DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=42)
clf2 = RandomForestClassifier(criterion='entropy', max_depth=8, n_estimators=25,
                        random_state=42)
clf3 = AdaBoostClassifier(learning_rate=0.8, n_estimators=25, random_state=42)


# ### SVM
```

```
from sklearn import svm
svm_model = svm.SVC(gamma="scale",kernel="rbf")
svm_model.fit(train_X, train_y)
y_pred = svm_model.predict(test_X)

train_svm = svm_model.score(train_X, train_y)
test_svm = svm_model.score(test_X, test_y)
```

```
m_svm = confusion_matrix(test_y, y_pred)
svm_byclass = m_svm.diagonal()/m_svm.sum(axis=1)
svm_byclass = svm_byclass.tolist()
svm_byclass
```

```
# ### MLP

from sklearn.neural_network import MLPClassifier
mlp_model = MLPClassifier(activation='tanh', hidden_layer_sizes=(20,), random_state=42)
mlp_model.fit(train_X, train_y)
y_pred = mlp_model.predict(test_X)

train_mlp = mlp_model.score(train_X, train_y)
test_mlp = mlp_model.score(test_X, test_y)


from sklearn.model_selection import GridSearchCV

param_grid = {
                'hidden_layer_sizes': [(10,30,10),(20,)],
                'activation': ['tanh', 'relu'],
                'solver': ['sgd', 'adam'],
                'alpha': [0.0001, 0.05],
                'learning_rate': ['constant','adaptive'],
             }

grid_clf_mlp = GridSearchCV(mlp_model, param_grid, cv=10, n_jobs=-1, refit = True)
grid_clf_mlp.fit(train_X, train_y)

y_pred = grid_clf_mlp.predict(test_X)
```

```
train_mlp = grid_clf_mlp.score(train_X, train_y)
test_mlp = grid_clf_mlp.score(test_X, test_y)
```

```
grid_clf_mlp.best_estimator_
```

```
m_mlp = confusion_matrix(test_y, y_pred)
mlp_byclass = m_mlp.diagonal()/m_mlp.sum(axis=1)
mlp_byclass = mlp_byclass.tolist()
mlp_byclass
```

```
methods = ['KNN', 'Decision Tree', 'Random Forest', 'Logistic Regression', 'AdaBoost', 'SVM', 'MLP']
train_accuracy = [train_knn, train_tr, train_rf, train_log, train_adb, train_svm, train_mlp]
test_accuracy = [test_knn, test_tr, test_rf, test_log, test_adb, test_svm, test_mlp]
rock_accuracy = [knn_byclass[0], tr_byclass[0], rf_byclass[0], log_byclass[0], ada_byclass[0], svm_byclass[0], mlp_byclass[0]]
instrumental_accuracy = [knn_byclass[1], tr_byclass[1], rf_byclass[1], log_byclass[1], ada_byclass[1], svm_byclass[1], mlp_byclass[1]]
hiphop_accuracy = [knn_byclass[2], tr_byclass[2], rf_byclass[2], log_byclass[2], ada_byclass[2], svm_byclass[2], mlp_byclass[2]]
```

```
zipped = list(zip(methods, test_accuracy, train_accuracy, rock_accuracy, instrumental_accuracy, hiphop_accuracy))
result = pd.DataFrame(zipped, columns=['Methods', 'Overall Training', 'Overall Testing', 'Rock', 'Instrumental', 'Hip Hop'])
```

```
result.round(4).sort_values('Overall Testing', ascending=False)
```