CS-GY 6923 MACHINE LEARNING

Professor: Dr. Raman Kannan

Project 1: Classification Techniques

Author: Dharu Piraba Muguntharaman

INTRODUCTION:

From the Exploratory Data Analysis, we have a total of 23 features. But when we look at the heat map, a lot of features are highly negatively corelated. Hence I decided to explore further on the feature importance.

PRINCIPAL COMPONENT ANALYSIS:

Principal Component Analysis is commonly used for dimensionality reduction by using each data point onto only the first few principal components (most cases first and second dimensions) to obtain lower-dimensional data while keeping as much of the data's variation as possible.

PCA is performed on the music data and the summary is displayed. It could be seen that, the first five features contribute more compared to rest of the features. Hence, we could remove the less contributing features.

```
music_data.pca=prcomp(music_data[,c(1:23)], center = TRUE,scale. = TRUE)
summary(music_data.pca)
```

```
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6     PC7
Standard deviation     1.9058 1.47283 1.13843 1.07071 1.06279 1.05734 1.05165
Proportion of Variance 0.1579 0.09431 0.05635 0.04984 0.04911 0.04861 0.04809
Cumulative Proportion  0.1579 0.25223 0.30858 0.35843 0.40754 0.45614 0.50423
                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
Standard deviation     1.04564 1.04179 1.0375 1.03370 1.02785 1.01515 0.99545
Proportion of Variance 0.04754 0.04719 0.0468 0.04646 0.04593 0.04481 0.04308
Cumulative Proportion  0.55177 0.59895 0.6458 0.69222 0.73815 0.78296 0.82604
                          PC15    PC16    PC17    PC18   PC19    PC20    PC21
Standard deviation     0.97391 0.94901 0.84462 0.77613 0.6850 0.50620 0.33274
Proportion of Variance 0.04124 0.03916 0.03102 0.02619 0.0204 0.01114 0.00481
Cumulative Proportion  0.86728 0.90644 0.93745 0.96364 0.9840 0.99519 1.00000
                            PC22      PC23
Standard deviation     3.299e-16 2.671e-16
Proportion of Variance 0.000e+00 0.000e+00
Cumulative Proportion  1.000e+00 1.000e+00
```

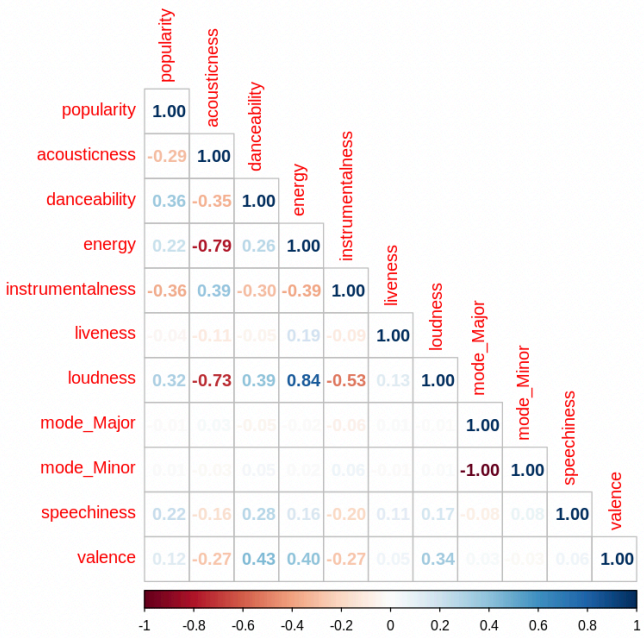
REMOVING UNHELPFUL FEATURES:

From PCA and the heat map, it could be seen that we had a lot of unhelpful features. Hence, these features are removed. The final dataset contains only 13 features and one target variable. The heat map of the feature variables are plotted.

```
names(music_data)
```

'popularity' · 'acousticness' · 'danceability' · 'energy' · 'instrumentalness' · 'liveness' · 'loudness' · 'mode_Major' · 'mode_Minor' · 'speechiness' · 'valence' · 'music_genre'

```
M = cor(numeric_cols)
corrplot(M, method = 'number',type = "lower")
```



PREPARING DATA FOR CLASSIFICATION:

In order to perform classification, the dataset has to be split into training and testing datasets. We set the seed to 5596. The whole dataset is split into two datasets such that the training dataset has 80% of the original dataset and the testing dataset has 20% of the original dataset.

```
[ ]  set.seed(5596)
     idx=sample(1:nrow(music_data),0.8*nrow(music_data))
     train_set=music_data[idx,]
     test_set=music_data[-idx,]
```

```
     dim(train_set)
     dim(test_set)
```

```
40000 · 12
10000 · 12
```

CLASSIFICATION MODELS:

1.LOGISTIC REGRESSION:

TRAIN DATA:

Building the model:

Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables. Since, our dataset contains more than two categorical values in the target column, a simple logistic regression model would be insufficient. Hence, we use the multinominal logistic regression model to classify our target variable.

```
require(nnet)
library(tidyverse)
formstr="music_genre~."
lr_model=multinom(formstr,train_set,maxit=1000) # Train the model
lr_pred_train=predict(lr_model,train_set,type="class")
lr_train_table=confusionMatrix(table(train_set$music_genre, lr_pred_train))
```

We first import the require packages like nnet and tidyverse. We then put the features in a single variable called formstr. The multinom logistic regression model is built using the multinom() function. The train data and the features are given as input to this model. Lr_pred_train stores the predicted values of the model. We then construct the confusion matrix of the train data and store it in the lr_train_table.

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

✔ tibble  3.1.8     ✔ purrr   0.3.5
✔ tidyr   1.2.1     ✔ stringr 1.4.1
✔ readr   2.1.3     ✔ forcats 0.5.2

── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ data.table::between() masks dplyr::between()
✖ dplyr::filter()       masks stats::filter()
✖ data.table::first()   masks dplyr::first()
✖ dplyr::lag()          masks stats::lag()
✖ data.table::last()    masks dplyr::last()
✖ purrr::lift()         masks caret::lift()
✖ tidyr::replace_na()   masks mltools::replace_na()
✖ purrr::transpose()    masks data.table::transpose()

# weights:  130 (108 variable)
initial  value 92103.403720
iter  10 value 81663.220347
iter  20 value 70612.897729
iter  30 value 61321.609198
iter  40 value 58767.788933
iter  50 value 54131.223827
iter  60 value 52632.777058
iter  70 value 51765.359956
iter  80 value 51264.403383
iter  90 value 51042.334484
iter 100 value 50980.300064
iter 110 value 50959.753593
iter 120 value 50952.503883
iter 130 value 50952.062727
iter 130 value 50952.062571
iter 130 value 50952.062567
final  value 50952.062567
converged
```

Confusion Matrix:

```
lr_train_table
```

```
Confusion Matrix and Statistics

             lr_pred_train
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative        1280    21   100        12     933         286     317  243
  Anime               102  2454   401       484     220         220       0  109
  Blues               139   727  1846        74     442         198       4  411
  Classical            73   249   104      3206      41          98       0  207
  Country             232    53   458        11    2283         154      55  120
  Electronic          263   329   245        45     188        2323     112  336
  Hip-Hop             301     0     8         1     134          49    1860   33
  Jazz                 92   189   558       281     418         630     107 1562
  Rap                 308     2     0         0     120          18    1507   39
  Rock                447     7    12         8     524          30      71   88
             lr_pred_train
              Rap Rock
  Alternative  152  683
  Anime          3   13
  Blues          9  162
  Classical      0   19
  Country       29  621
  Electronic    66  100
  Hip-Hop     1338  214
  Jazz          13  152
  Rap         1566  440
  Rock         228 2580

Overall Statistics

               Accuracy : 0.524
                 95% CI : (0.5191, 0.5289)
    No Information Rate : 0.1326
    P-Value [Acc > NIR] : < 2.2e-16
```

The above figure displays the confusion matrix of the train data. It can be seen that the accuracy of the model is around 52%. This accuracy is lower.

Performance Metrics:

The other performance metrics of the model are evaluated for a better understanding of the working of the model. The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the lr_train_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics.

```
# Logistic Regression Performance Parameters
require(pROC)
# Training Data
lr_train_pm=lr_train_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Logistic-Regression Training data Performance Parameters:")
lr_train_pm
lr_train_macavg=round(apply(lr_train_pm,2,mean),4)
print("Macro Averages:")
lr_train_macavg
```

[1] "Logistic-Regression Training data Performance Parameters:"

A matrix: 10 x 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6603530	0.3178545	0.3954279	0.9252781	0.3954279
Class: Anime	0.7828168	0.6125811	0.6087819	0.9568517	0.6087819
Class: Blues	0.7174594	0.4601196	0.4946409	0.9402779	0.4946409
Class: Classical	0.8778654	0.8021016	0.7777778	0.9779531	0.7777778
Class: Country	0.6902822	0.5684761	0.4305110	0.9500533	0.4305110
Class: Electronic	0.7665473	0.5797355	0.5798802	0.9532144	0.5798802
Class: Hip-Hop	0.7017100	0.4723210	0.4611951	0.9422248	0.4611951
Class: Jazz	0.7149886	0.3903048	0.4961881	0.9337892	0.4961881
Class: Rap	0.6967685	0.3915000	0.4600470	0.9334900	0.4600470
Class: Rock	0.7386232	0.6458073	0.5176565	0.9595899	0.5176565

[1] "Macro Averages:"

Balanced Accuracy: 0.7347 Precision: 0.5241 Sensitivity: 0.5222 Specificity: 0.9473 Recall: 0.5222

Area Under the Curve:

The AUC of the train data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed.

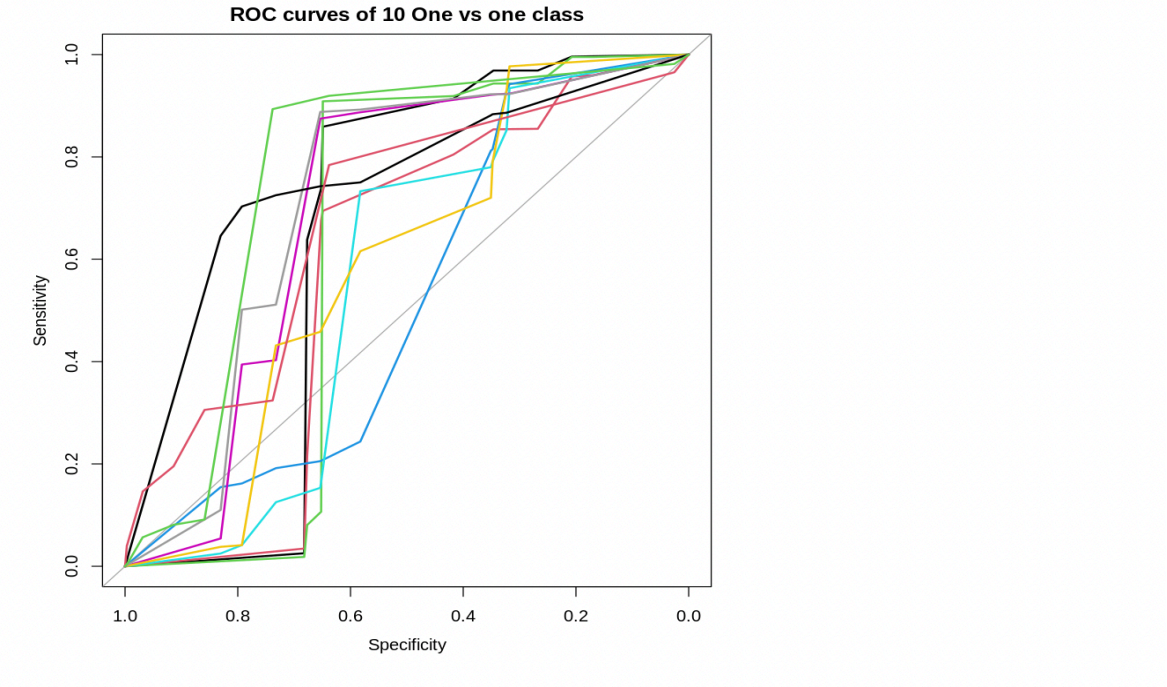
```
# finding AUC
lr_train_=predict(lr_model, train_set)
lr_train_AUC=multiclass.roc(train_set$music_genre,as.numeric(lr_train_))
print(paste("Logistic-Regression Training AUC:",round(lr_train_AUC$auc,4)))
```

- [1] "Logistic-Regression Training AUC: 0.7352"

Receiver Operating Characteristic Curve:

ROC of the train data is plotted using plot.roc function. Since, we have more than two classes in the target variable, a for loop is used to plot each of the classes with specificity in X axis and sensitivity in Y axis.

```
# finding ROC
lr_train_ROC=lr_train_AUC$rocs
plot.roc(lr_train_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(lr_train_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias of an estimator is the difference between this estimator's expected value and the true value of the parameter being estimated. The variance is a numerical measure of how the data values is dispersed around the mean. The bias and variance of the trained model is calculated using the bias() and variance() function. From the below figure, it could be seen that the variance of the model is 4 and bias is 0.09.

```
#bias and variance
print(var(as.numeric(lr_train_), as.numeric(as.factor(train_set$music_genre))))
bias(as.numeric(pred1), as.numeric(as.factor(train_set$music_genre)))
```

```
[1] 4.004507
0.09275
```

TEST DATA:

The test data and the features are given as input to this model. Lr_pred_test stores the predicted values of the model. We then construct the confusion matrix of the train data and store it in the lr_test_table.

```
lr_pred_test=predict(lr_model,test_set,type="class")
lr_test_table=confusionMatrix(table(test_set$music_genre, lr_pred_test))
lr_test_table
```

Confusion Matrix:

```
Confusion Matrix and Statistics

             lr_pred_test
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         314     8    18         3     208         66      90   66
  Anime                25   608    88       129      51         63       0   24
  Blues                47   172   434        23     107         61       1  105
  Classical            25    70    27       787       8         17       0   65
  Country              62    24    99         1     565         35      22   37
  Electronic           72    85    49        17      47        559      21   96
  Hip-Hop              84     0     1         0      28         18     495   18
  Jazz                 27    64   128        67     100        141      32  395
  Rap                  60     0     1         0      25          9     356    6
  Rock                125     4     3         1     123          7      12   22
             lr_pred_test
              Rap Rock
  Alternative  34  166
  Anime         2    4
  Blues         1   37
  Classical     0    4
  Country      11  128
  Electronic   20   27
  Hip-Hop     360   58
  Jazz          2   42
  Rap         417  126
  Rock         52  656

Overall Statistics

               Accuracy : 0.523
                 95% CI : (0.5132, 0.5328)
    No Information Rate : 0.1262
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.47
```

The above figure displays the confusion matrix of the test data. It can be seen that the accuracy of the model is around 52%. This accuracy is similar to the train accuracy.

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the lr_test_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics. The performance metrics of the test data do not vary much in comparison to the train data.

```
# Logistic Regression Performance Parameters
require(pROC)
# Testing Data
lr_test_pm=lr_test_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Logistic-Regression Testing data Performance Parameters:")
lr_test_pm
lr_test_macavg=round(apply(lr_test_pm,2,mean),4)
print("Macro Averages:")
lr_test_macavg
```

```
[1] "Logistic-Regression Training data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6507070	0.3227133	0.3733650	0.9280489	0.3733650
Class: Anime	0.7721916	0.6116700	0.5874396	0.9569437	0.5874396
Class: Blues	0.7256296	0.4392713	0.5117925	0.9394668	0.5117925
Class: Classical	0.8707447	0.7846461	0.7655642	0.9759251	0.7655642
Class: Country	0.6998753	0.5741870	0.4477021	0.9520485	0.4477021
Class: Electronic	0.7623260	0.5629406	0.5727459	0.9519060	0.5727459
Class: Hip-Hop	0.7089230	0.4661017	0.4810496	0.9367963	0.4810496
Class: Jazz	0.7039173	0.3957916	0.4736211	0.9342134	0.4736211
Class: Rap	0.6998949	0.4170000	0.4638487	0.9359411	0.4638487
Class: Rock	0.7428822	0.6527363	0.5256410	0.9601234	0.5256410

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.7337 Precision: 0.5227 Sensitivity: 0.5203 Specificity: 0.9471 Recall: 0.5203

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the test data is similar to the train data.

```
# finding AUC
lr_test_=predict(lr_model, test_set)
lr_test_AUC=multiclass.roc(test_set$music_genre,as.nume
print(paste("Logistic-Regression Testing AUC:",round(lr
```

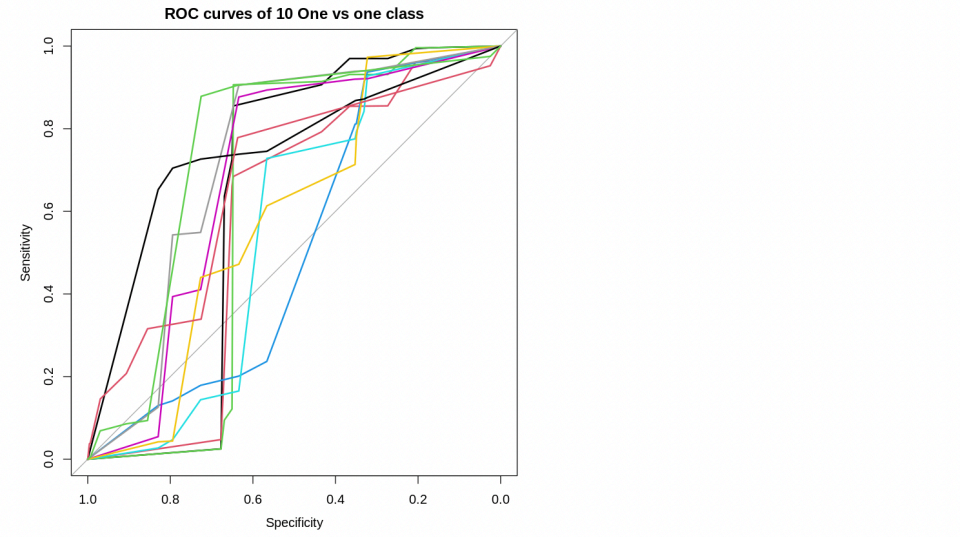
```
Setting direction: controls < cases

[1] "Logistic-Regression Testing AUC: 0.7338"
```

Receiver Operating Characteristic Curve:

ROC of the test data is plotted using plot.roc function. Since, we have more than two classes in the target variable, a for loop is used to plot each of the classes with specificity in X axis and sensitivity in Y axis.

```
# finding ROC
require(pROC)
lr_test_ROC=lr_test_AUC$rocs
plot.roc(lr_test_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(lr_test_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the test data is found using the bias() and variance() function. It could be inferred that the variance is similar to train data but variance is slightly lower compared to training data.

```
#Bias and variance
print(var(as.numeric(lr_test_), as.numeric(as.factor(test_set$music_genre))))
bias(as.numeric(lr_test_), as.numeric(as.factor(test_set$music_genre)))
```

```
[1] 4.036767
0.0896
```

INFERENCE :

When comparing the training and testing models under logistic regression it could be seen that the accuracies of both models are similar and the respective performance metrics too. From this, we could infer that the model fails to learn from the given data to classify the music genre.

2. NAIVE BAYES CLASSIFIER:

TRAIN DATA:

Building the model:

Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. The e1071 package is used to call the multiclass naiveBayes() classifier model. The model is built by passing the training data and the features. The built model is stored under the nb_model variable.

```
require(e1071)
formstr="music_genre~."
formstr_nb=formula(formstr)
nb_model=naiveBayes(formstr_nb,data=train_set) # train model
```

```
Loading required package: e1071


Attaching package: 'e1071'


The following object is masked from 'package:mltools':

    skewness
```

The nb_model is used to predict the genre from training data. This output is stored in the nb_pred_train variable. The corresponding confusion matrix is calculated using the confusionMatrix() function.

```
nb_pred_train=predict(nb_model,train_set,type="class")
nb_train_table=confusionMatrix(table(train_set$music_genre,nb_pred_train))
print(nb_train_table)
```

Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 42%. This accuracy is less than 50%, so it can be seen that the model doesn't perform so well.

```
Confusion Matrix and Statistics

             nb_pred_train
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         862    10    65        44    1633        189     379  202
  Anime               134  1195   138       792    1197        447       1   94
  Blues               186   260   768       143    1750        373      32  414
  Classical            78    70    30      3474     119         90       1  133
  Country             206    26   111        57    3125         60     121  107
  Electronic          284   179   165        64     614       1929     185  426
  Hip-Hop              73     0     4         2     508         34    1365   50
  Jazz                130    92   262       655     690        631     225 1244
  Rap                  99     1     9         3     601         14    1008   57
  Rock                247     1    25        94    1948         95      73  130
             nb_pred_train
              Rap Rock
  Alternative  349  294
  Anime          6    2
  Blues         25   61
  Classical      0    2
  Country      105   98
  Electronic   105   56
  Hip-Hop     1843   59
  Jazz          28   45
  Rap         2037  171
  Rock         503  879

Overall Statistics

               Accuracy : 0.422
                 95% CI : (0.4171, 0.4268)
    No Information Rate : 0.3046
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3577
```

Performance Metrics:

The other performance metrics of the model are evaluated for a better understanding of the working of the model. The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the nb_train_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics.

```
# Naive Bayes Performance Parameters
require(pROC)
# Training Data
nb_train_pm=nb_train_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Naive Bayes Training data Performance Parameters:")
nb_train_pm
nb_train_macavg=round(apply(lr_train_pm,2,mean),4)
print("Macro Averages:")
nb_train_macavg
```

```
[1] "Naive Bayes Training data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6454978	0.2140551	0.3749456	0.9160500	0.3749456
Class: Anime	0.7889647	0.2983025	0.6515812	0.9263481	0.6515812
Class: Blues	0.7012860	0.1914257	0.4870006	0.9155714	0.4870006
Class: Classical	0.8184714	0.8691519	0.6520270	0.9849158	0.6520270
Class: Country	0.6122149	0.7781375	0.2564629	0.9679669	0.2564629
Class: Electronic	0.7209902	0.4814075	0.4994821	0.9424982	0.4994821
Class: Hip-Hop	0.6661868	0.3466227	0.4026549	0.9297187	0.4026549
Class: Jazz	0.6805841	0.3108446	0.4354218	0.9257464	0.4354218
Class: Rap	0.6756156	0.5092500	0.4073185	0.9439127	0.4073185
Class: Rock	0.7230034	0.2200250	0.5272945	0.9187123	0.5272945

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.7347 Precision: 0.5241 Sensitivity: 0.5222 Specificity: 0.9473 Recall: 0.5222

Area Under the Curve:

The AUC of the train data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the train data is found to be 0.70.

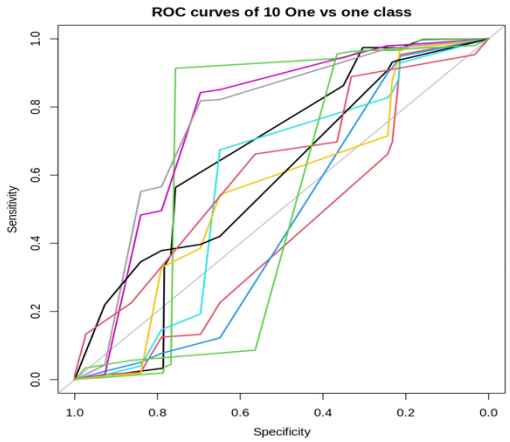
```
# finding AUC
nb_train_=predict(nb_model, train_set)
nb_train_AUC=multiclass.roc(train_set$music_genre,as.numeric(nb_train_))
print(paste("Naive Bayes Training AUC:",round(nb_train_AUC$auc,4)))
nb_train_ROC=nb_train_AUC$rocs
```

```
[1] "Naive Bayes Training AUC: 0.7038"
```

Receiver Operating Characteristic Curve:

ROC of the train data is plotted using plot.roc function. Since, we have more than two classes in the target variable, a for loop is used to plot each of the classes with specificity in X axis and sensitivity in Y axis.

```
# finding ROC
require(pROC)
nb_train_ROC=nb_train_AUC$rocs
plot.roc(nb_train_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(nb_train_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the train data is found using the bias() and variance() function. It could be inferred that the variance is 2.7 and the bias is 0.11 of the model.

```
#Bias and variance
print(var(as.numeric(nb_train_), as.numeric(as.factor(train_set$music_genre))))
bias(as.numeric(nb_train_), as.numeric(as.factor(train_set$music_genre)))
```

```
[1] 2.723254
0.116375
```

TEST DATA:

The test data and the features are given as input to this model. Lr_pred_test stores the predicted values of the model. We then construct the confusion matrix of the train data and store it in the lr_test_table.

```
nb_pred_test=predict(nb_model,test_set,type="class")
nb_test_table=confusionMatrix(table(test_set$music_genre,nb_pred_test))
print(nb_test_table)
```

Confusion Matrix:

```
Confusion Matrix and Statistics

             nb_pred_test
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         195     3     9        11     398          49      93   60
  Anime                23   319    36       193     277         118       2   21
  Blues                56    66   199        46     392          91      11  115
  Classical            14    15    14       872      31          15       0   39
  Country              44    14    24        14     747          15      44   33
  Electronic           76    44    26        22     144         474      58  100
  Hip-Hop              24     0     5         0     127           8     359   23
  Jazz                 34    31    55       168     158         156      68  307
  Rap                  22     1     2         0     138           8     255   18
  Rock                 77     2     7        13     480          20      13   33
             nb_pred_test
              Rap Rock
  Alternative  97   58
  Anime         2    3
  Blues         2   10
  Classical     0    3
  Country      27   22
  Electronic   31   18
  Hip-Hop     498   18
  Jazz          9   12
  Rap         510   46
  Rock        135  225

Overall Statistics

               Accuracy : 0.4207
                 95% CI : (0.411, 0.4304)
    No Information Rate : 0.2892
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3564
```

The above figure displays the confusion matrix of the test data. It can be seen that the accuracy of the model is around 42%. This accuracy is similar to the train accuracy.

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the nb_test_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics. The performance metrics of the test data do not vary much in comparison to the train data.

```r
# Naive Bayes Performance Parameters
require(pROC)
# Training Data
nb_test_pm=nb_test_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Naive Bayes Testing data Performance Parameters:")
nb_test_pm
nb_test_macavg=round(apply(lr_test_pm,2,mean),4)
print("Macro Averages:")
nb_test_macavg
```

```
[1] "Naive Bayes Testing data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6313369	0.2004111	0.3451327	0.9175411	0.3451327
Class: Anime	0.7867146	0.3209256	0.6444444	0.9289847	0.6444444
Class: Blues	0.7229302	0.2014170	0.5278515	0.9180089	0.5278515
Class: Classical	0.8180535	0.8693918	0.6512323	0.9848747	0.6512323
Class: Country	0.6124780	0.7591463	0.2582988	0.9666573	0.2582988
Class: Electronic	0.7197410	0.4773414	0.4968553	0.9426266	0.4968553
Class: Hip-Hop	0.6601427	0.3380414	0.3975637	0.9227218	0.3975637
Class: Jazz	0.6675926	0.3076152	0.4098798	0.9253054	0.4098798
Class: Rap	0.6663114	0.5100000	0.3890160	0.9436069	0.3890160
Class: Rock	0.7303958	0.2238806	0.5421687	0.9186228	0.5421687

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.7337 Precision: 0.5227 Sensitivity: 0.5203 Specificity: 0.9471 Recall: 0.5203

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the test data is found to be 0.70. This is similar to train data AUC.

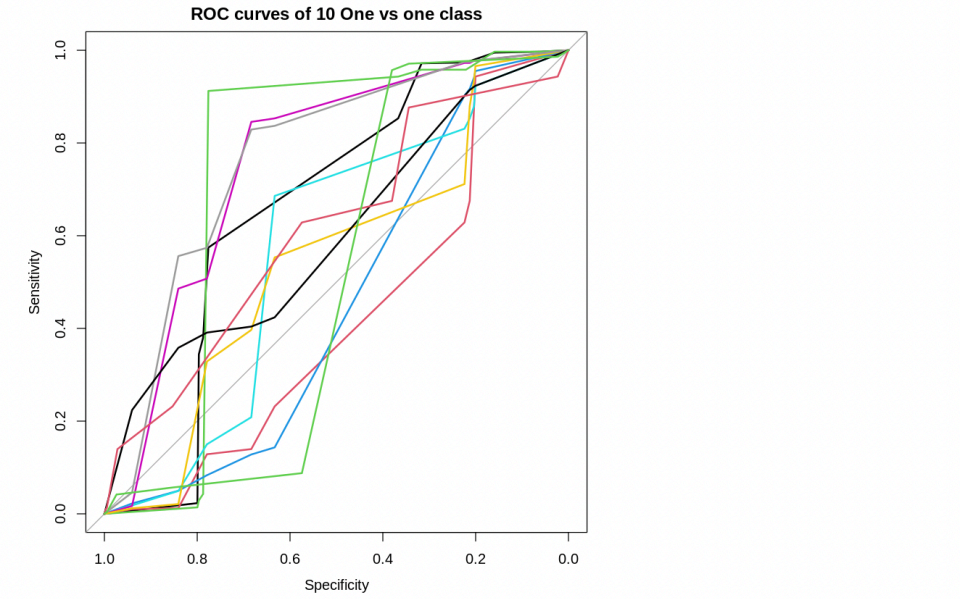
```r
# finding AUC
nb_test_=predict(nb_model, test_set)
nb_test_AUC=multiclass.roc(test_set$music_genre,as.numeric(nb_test_))
print(paste("Naive Bayes Testing AUC:",round(nb_test_AUC$auc,4)))
nb_test_ROC=nb_test_AUC$rocs
```

```
[1] "Naive Bayes Testing AUC: 0.7028"
```

Receiver Operating Characteristic Curve:

ROC of the test data is plotted using plot.roc function. Since, we have more than two classes in the target variable, a for loop is used to plot each of the classes with specificity in X axis and sensitivity in Y axis.

```
# finding ROC
require(pROC)
nb_test_ROC=nb_test_AUC$rocs
plot.roc(nb_test_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(nb_test_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the test data is found using the bias() and variance() function. It could be inferred that the variance is 2.7 and the bias is 0.12 of the model. This is somewhat similar to the train data.

```
#Bias and variance
print(var(as.numeric(nb_test_), as.numeric(as.factor(test_set$music_genre))))
bias(as.numeric(nb_test_), as.numeric(as.factor(test_set$music_genre)))
```

```
[1] 2.722211
0.1205
```

INFERENCE :

When comparing the training and testing models under naïve bayes it could be seen that the accuracies of both models are similar and the respective performance metrics too. From this, we could infer that the model fails to learn from the given data to classify the music genre properly.

3. SUPPORT VECTOR MACHINE:

TRAIN DATA:

Building the model:

An SVM classifies data by determining the optimal hyperplane that separates observations according to their class labels. The central concept is to accommodate classes separable by linear and non-linear class boundaries. The e1071 package is used to call the multiclass svm() classifier model. The model is built by passing the training data and the features. The built model is stored under the song_svm variable.

The song_svm is used to predict the genre from training data. This output is stored in the pred_train_svm variable. The corresponding confusion matrix is calculated using the confusionMatrix() function.

```
[ ]
    song_svm=svm(as.factor(music_genre)~., data = train_set, cost = 1)
```

```
pred_train_svm = predict(song_svm)
table_svm_train=table(train_set$music_genre, pred_train_svm)
svm_train_stat=caret::confusionMatrix(table_svm_train)
print(svm_train_stat)
```

Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 58%. This accuracy is greater than 50%, so it can be seen that the model performs well compared to the previous models.

```
Confusion Matrix and Statistics

             pred_train_svm
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative        1571    24    58         6     626        188     421  208
  Anime               144  2798   289       329     162        192       1   67
  Blues               176   382  2176        55     417        206      11  396
  Classical           103   119    83      3400      22         96       0  161
  Country             274    70   273         4    2210        115      81  200
  Electronic          281   208   234        25     167       2443     129  368
  Hip-Hop             144     0     6         0      38         36    2146   26
  Jazz                150    76   439       273     275        528     108 1980
  Rap                 148     0     2         0      43         16    1546   26
  Rock                349     9    11         9     252         31     111   97
             pred_train_svm
               Rap Rock
  Alternative  134  791
  Anime          4   20
  Blues          6  187
  Classical      0   13
  Country       30  759
  Electronic    50  102
  Hip-Hop     1282  260
  Jazz          16  157
  Rap         1683  536
  Rock         172 2954

Overall Statistics

               Accuracy : 0.584
                 95% CI : (0.5792, 0.5889)
    No Information Rate : 0.1445
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5378
```

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the svm_train_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics.

```
# SVM train Performance Parameters
require(pROC)
# Training Data
svm_train_pm=svm_train_stat$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Support Vector Machine Training Data Performance Parameters:")
svm_train_pm
svm_train_macavg=round(apply(svm_train_pm,2,mean),4)
print("Macro Averages:")
svm_train_macavg
```

```
[1] "Support Vector Machine Training Data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.7016826	0.3901167	0.4703593	0.9330060	0.4703593
Class: Anime	0.8629115	0.6984523	0.7590884	0.9667346	0.7590884
Class: Blues	0.7794769	0.5423729	0.6093531	0.9496006	0.6093531
Class: Classical	0.9062180	0.8506380	0.8290661	0.9833700	0.8290661
Class: Country	0.7371138	0.5502988	0.5246914	0.9495362	0.5246914
Class: Electronic	0.7955577	0.6096831	0.6343807	0.9567346	0.6343807
Class: Hip-Hop	0.7103392	0.5449467	0.4712341	0.9494442	0.4712341
Class: Jazz	0.7528121	0.4947526	0.5610655	0.9445587	0.5610655
Class: Rap	0.7175525	0.4207500	0.4983713	0.9367337	0.4983713
Class: Rock	0.7403706	0.7394243	0.5111611	0.9695801	0.5111611

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.7704 Precision: 0.5841 Sensitivity: 0.5869 Specificity: 0.9539 Recall: 0.5869

Area Under the Curve:

The AUC of the train data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the train data is found to be 0.75.

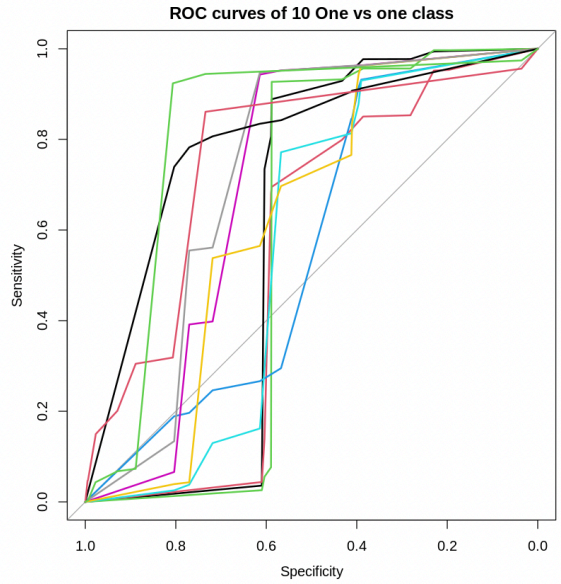
```
# finding AUC
svm_train_=predict(song_svm, train_set, type="prob")
svm_train_AUC=multiclass.roc(train_set$music_genre,as.numeric(svm_train_))
print(paste("Support Vector Machine Training AUC:",round(svm_train_AUC$auc,4)))
svm_train_ROC=svm_train_AUC$rocs
```

```
[1] "Support Vector Machine Training AUC: 0.755"
```

Receiver Operating Characteristic Curve:

The ROC curve of the train data is plotted for each of the 10 classes.

```
#Plotting ROC
svm_train_ROC=svm_train_AUC$rocs
plot.roc(svm_train_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(svm_train_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the train data is found using the bias() and variance() function. It could be inferred that the variance is 4.6 and the bias is 0.26 of the model.

```
#Bias and variance
print(var(as.numeric(svm_train_), as.numeric(as.factor(train_set$music_genre))))
bias(as.numeric(svm_train_), as.numeric(as.factor(train_set$music_genre)))
```

```
[1] 4.653399
0.264275
```

TEST DATA:

Building the model:

The test data and the features are given as input to this model. pred_test_svm stores the predicted values of the model. We then construct the confusion matrix of the train data and store it in the svm_test_stat.

```
pred_test_svm =predict(song_svm, test_set)
table_svm_test=table(test_set$music_genre, pred_test_svm)
svm_test_stat=caret::confusionMatrix(table_svm_test)
print(svm_test_stat)
```

Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 56%. This accuracy is greater than 50%, but it is less than the training model accuracy.

```
Confusion Matrix and Statistics

             pred_test_svm
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         356     8    18         1     145          41     108   58
  Anime                24   684    66        88      49          55       1   19
  Blues                58    88   504        16      94          56       4  121
  Classical            31    35    28       831       4          16       0   55
  Country              73    28    65         1     518          29      34   53
  Electronic           71    50    48        11      40         588      32  108
  Hip-Hop              44     0     1         0      10          11     525   15
  Jazz                 43    22   113        74      68         128      36  474
  Rap                  35     0     2         0       9          10     406    8
  Rock                 88     4     2         1      61           9      25   21
             pred_test_svm
              Rap Rock
  Alternative  35  203
  Anime         1    7
  Blues         1   46
  Classical     0    3
  Country      11  172
  Electronic   14   31
  Hip-Hop     387   69
  Jazz          4   36
  Rap         388  142
  Rock         44  750

Overall Statistics

               Accuracy : 0.5618
                 95% CI : (0.552, 0.5716)
    No Information Rate : 0.1459
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.513
```

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the svm_test_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics. We could notice that there are some minor differences in the performance metrics of the test data and train data.

```
# SVM test Performance Parameters
require(pROC)
# Testing Data
svm_test_pm=svm_test_stat$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Support Vector Machine Testing Data Performance Parameters:")
svm_test_pm
svm_test_macavg=round(apply(svm_test_pm,2,mean),4)
print("Macro Averages:")
svm_test_macavg
```

```
[1] "Support Vector Machine Testing Data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6826652	0.3658787	0.4325638	0.9327667	0.4325638
Class: Anime	0.8550750	0.6881288	0.7442873	0.9658628	0.7442873
Class: Blues	0.7710812	0.5101215	0.5950413	0.9471212	0.5950413
Class: Classical	0.8965783	0.8285145	0.8123167	0.9808399	0.8123167
Class: Country	0.7336359	0.5264228	0.5190381	0.9482337	0.5190381
Class: Electronic	0.7894125	0.5921450	0.6235419	0.9552832	0.6235419
Class: Hip-Hop	0.6937562	0.4943503	0.4483348	0.9391777	0.4483348
Class: Jazz	0.7253990	0.4749499	0.5085837	0.9422144	0.5085837
Class: Rap	0.6856380	0.3880000	0.4384181	0.9328579	0.4384181
Class: Rock	0.7420974	0.7462687	0.5140507	0.9701440	0.5140507

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.7575 Precision: 0.5615 Sensitivity: 0.5636 Specificity: 0.9515 Recall: 0.5636

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the test data is found to be 0.74. This is slightly less than train data AUC.

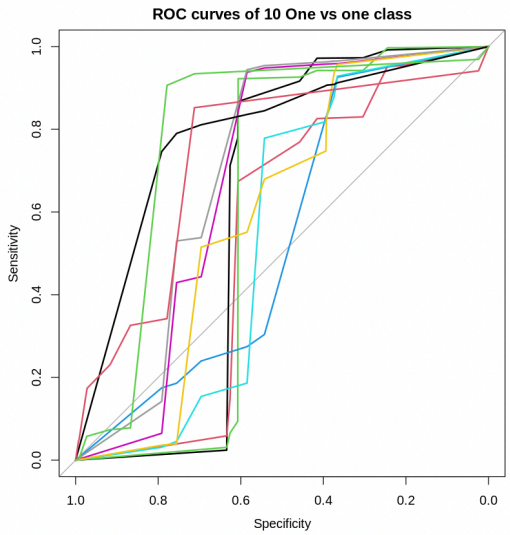
```
# finding AUC
svm_test_=predict(song_svm, test_set, type="prob")
svm_test_AUC=multiclass.roc(test_set$music_genre,as.numeric(svm_test_))
print(paste("Support Vector Machine Testing AUC:",round(svm_test_AUC$auc,4)))
svm_test_ROC=svm_test_AUC$rocs
```

```
[1] "Support Vector Machine Testing AUC: 0.7456"
```

Receiver Operating Characteristic Curve:

The ROC curve of the test data is plotted for each of the 10 classes.

```
#Plotting ROC
svm_test_ROC=svm_test_AUC$rocs
plot.roc(svm_test_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(svm_test_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the test data is found using the bias() and variance() function. It could be inferred that the variance is 4.4 and the bias is 0.28 of the model. The variance is lesser than train data and bias is more than train data.

```
#Bias and variance
print(var(as.numeric(svm_test_), as.numeric(as.factor(test_set$music_genre))))
bias(as.numeric(svm_test_), as.numeric(as.factor(test_set$music_genre)))
```

```
[1] 4.495685
0.2867
```

INFERENCE :

When comparing the training and testing models under svm it could be seen that there is a variation in the accuracies of both models and the respective performance metrics too. From this, we could infer that the model learns from the given data to classify the music genre properly.

4. DECISION TREE CLASSIFIER:

TRAIN DATA:

Building the model:

Decision tree are powerful non-linear classifiers, which utilize a tree structure to model the relationships among the features and the potential outcomes. A decision tree classifier uses a structure of branching decisions, which channel examples into a final predicted class value.

The rpart package is used to call the decision tree classifier model. The model is built by passing the training data and the features. The built model is stored under the model_dt variable.

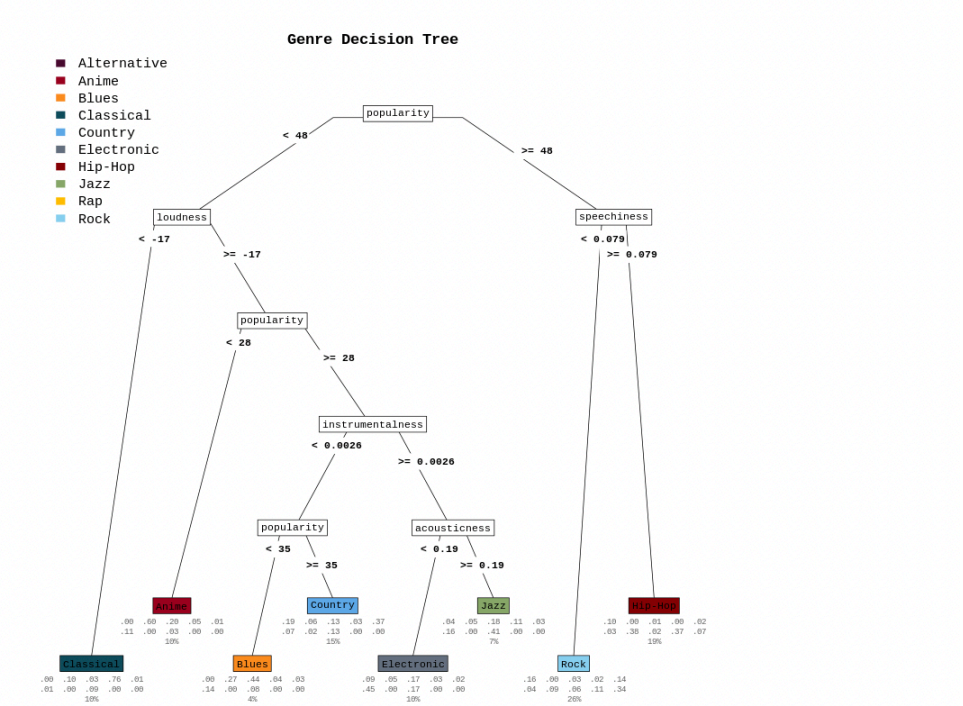
The decision tree is built using the training data. From the tree plot , it could be seen that it could classify only 8 of the 10 classes. It fails to classify the rap and alternative genre. It could be seen that popularity features plays a major role in determining the genre of music.

```
[ ]  install.packages("rpart")
     install.packages("rpart.plot")
     library("rpart")
     library("rpart.plot")
```

```
     Installing package into ‘/usr/local/lib/R/site-library’
     (as ‘lib’ is unspecified)

     Installing package into ‘/usr/local/lib/R/site-library’
     (as ‘lib’ is unspecified)
```

```
model_dt=rpart(music_genre~., data = train_set)
rpart.plot(model_dt,
           type = 5,
           extra = 104,
           box.palette = list(purple = "#490B32",
               red = "#9A031E",
               orange = '#FB8B24',
               dark_blue = "#0F4C5C",
               blue = "#5DA9E9",
               grey = '#66717E',
               brown="#880009",
               green="#87A96B",
               yellow="#FFBF00",
               light_blue="#89CFF0"),
           leaf.round = 0,
           fallen.leaves = TRUE,
           branch = 0.3,
           under = TRUE,
           under.col = 'grey40',
           family = 'Avenir',
           main = 'Genre Decision Tree',
           tweak = 1.2)
```



Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 43%. This accuracy is lesser than 50%.

```
predictions = predict(model_dt, train_set,type ="class")
dt_train_table=confusionMatrix(table(as.factor(train_set$music_genre),predictions))
print(dt_train_table)
```

```
Confusion Matrix and Statistics

             predictions
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative           0    16     2         8    1110        361     766  112
  Anime                 0  2354   472       396     373        181      30  149
  Blues                 0   780   778       117     759        667      53  518
  Classical             0   189    70      2900     151        132       9  322
  Country               0    34    52        22    2164         89     122   97
  Electronic            0   437   256        45     391       1769     211  459
  Hip-Hop               0     4     1         1     129         12    2910    0
  Jazz                  0   107   143       334     740        682     184 1162
  Rap                   0     6     0         0      27          1    2836    2
  Rock                  0     7     4         1      20          9     523    1
             predictions
               Rap Rock
  Alternative    0 1652
  Anime          0   51
  Blues          0  340
  Classical      0  224
  Country        0 1436
  Electronic     0  439
  Hip-Hop        0  881
  Jazz           0  650
  Rap            0 1128
  Rock           0 3430

Overall Statistics

               Accuracy : 0.4367
                 95% CI : (0.4318, 0.4416)
    No Information Rate : 0.2558
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3742
```

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the dt_train_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics. We could notice that because the model fails to classify two classes, the accuracy , sensitivity and recall values of the macro average has NA values.

```
# Decision Tree Performance Parameters
require(pROC)
# Training Data
dt_train_pm=dt_train_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Naive Bayes Training data Performance Parameters:")
dt_train_pm
dt_train_macavg=round(apply(dt_train_pm,2,mean),4)
print("Macro Averages:")
dt_train_macavg
```

```
[1] "Naive Bayes Training data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	NA	0.0000000	NA	0.8993250	NA
Class: Anime	0.7762841	0.5876186	0.5983732	0.9541951	0.5983732
Class: Blues	0.6764797	0.1939182	0.4375703	0.9153890	0.4375703
Class: Classical	0.8640221	0.7255442	0.7583682	0.9696760	0.7583682
Class: Country	0.6573889	0.5388446	0.3690314	0.9457464	0.3690314
Class: Electronic	0.6956207	0.4414774	0.4532411	0.9380004	0.4532411
Class: Hip-Hop	0.6744596	0.7389538	0.3806907	0.9682285	0.3806907
Class: Jazz	0.6676877	0.2903548	0.4117647	0.9236107	0.4117647
Class: Rap	NA	0.0000000	NA	0.9000000	NA
Class: Rock	0.6581381	0.8585732	0.3352556	0.9810205	0.3352556

```
[1] "Macro Averages:"
```

Balanced Accuracy: <NA> Precision: 0.4375 Sensitivity: <NA> Specificity: 0.9395 Recall: <NA>

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the train data is found to be 0.83.

```
# finding AUC
dt_train_=predict(model_dt, train_set)
dt_train_AUC=multiclass.roc(train_set$music_genre,dt_train_)
print(paste("Decision Tree Training AUC:",round(dt_train_AUC$auc,4)))
dt_train_ROC=dt_train_AUC$rocs
```

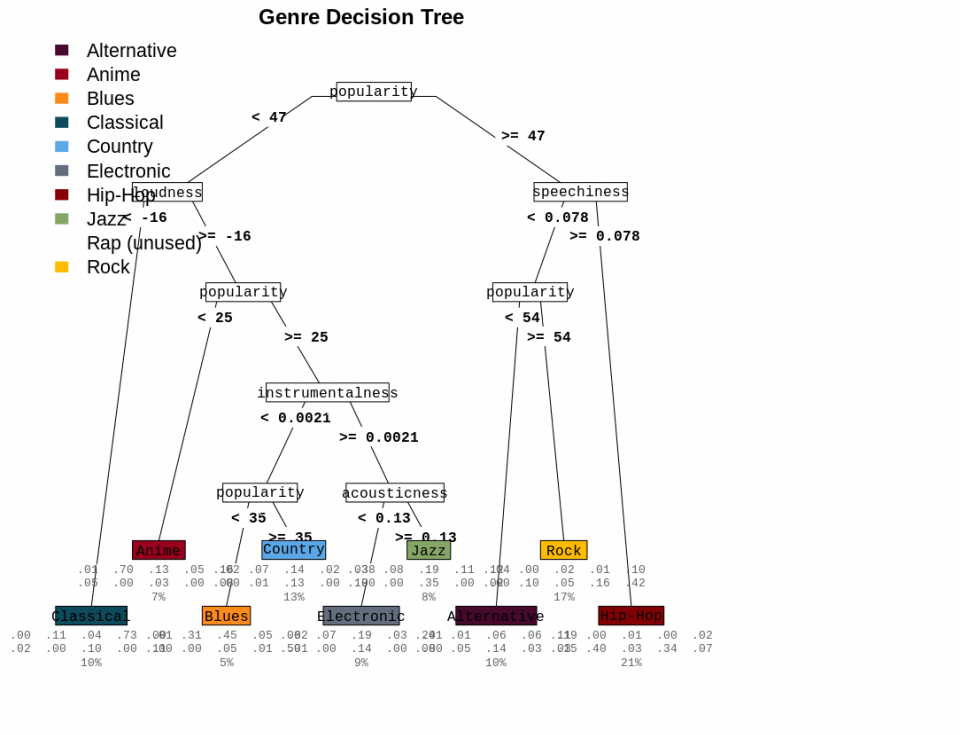
```
[1] "Decision Tree Training AUC: 0.8301"
```

TEST DATA:

Building the model:

The model is built using the rpart library package. Test data is fed as input to the model. The decision tree is built using the plot function. From the decision tree, it could be seen that it could classify only 9 of the 10 classes. It fails to classify the rap genre. It could be seen that popularity features plays a major role in determining the genre of music.

```
model_dtt=rpart(music_genre~., data = test_set)
rpart.plot(model_dtt,
           type = 5,
           extra = 104,
           box.palette = list(purple = "#490B32",
               red = "#9A031E",
               orange = '#FB8B24',
               dark_blue = "#0F4C5C",
               blue = "#5DA9E9",
               grey = '#66717E',
               brown="#880009",
               green="#87A96B",
               yellow="#FFBF00",
               light_blue="#89CFF0"),
           leaf.round = 0,
           fallen.leaves = TRUE,
           branch = 0.3,
           under = TRUE,
           under.col = 'grey40',
           family = 'Avenir',
           main = 'Genre Decision Tree',
           tweak = 1.2)
```




```
predictions_test = predict(model_dtt, test_set,type ="class")
dt_test_table=confusionMatrix(table(as.factor(test_set$music_genre),predictions_test))
print(dt_test_table)
```

Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 44%. This accuracy is lesser than 50%, but it is greater than the training model accuracy.

```
Confusion Matrix and Statistics

             predictions_test
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         249     4     1         0     215          56     220   27
  Anime                 7   495   156       108      93          60       5   64
  Blues                59    95   230        37     184         171      21  154
  Classical            57    36    24       728      32          27       4   86
  Country             199    17    12         6     502          12      36   29
  Electronic           84    38    55        17     102         456      57  150
  Hip-Hop              50     2     0         0      15           0     823    1
  Jazz                139    21    26       105     170         125      52  280
  Rap                  35     1     4         0       2           0     695    0
  Rock                150     3     3         0       1           1     152    0
             predictions_test
              Rap Rock
  Alternative   0  201
  Anime         0    6
  Blues         0   37
  Classical     0    9
  Country       0  171
  Electronic    0   34
  Hip-Hop       0  171
  Jazz          0   80
  Rap           0  263
  Rock          0  695

Overall Statistics

               Accuracy : 0.4458
                 95% CI : (0.436, 0.4556)
    No Information Rate : 0.2065
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3837
```

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the dt_test_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics. We could notice that because the model fails to classify two classes, the accuracy , sensitivity and recall values of the macro average has NA values.

```
# Decision Tree Performance Parameters
require(pROC)
# Testing Data
dt_test_pm=dt_test_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("Naive Bayes Training data Performance Parameters:")
dt_test_pm
dt_test_macavg=round(apply(dt_test_pm,2,mean),4)
print("Macro Averages:")
dt_test_macavg
```

```
[1] "Naive Bayes Training data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.5806390	0.2559096	0.2419825	0.9192955	0.2419825
Class: Anime	0.8207497	0.4979879	0.6952247	0.9462748	0.6952247
Class: Blues	0.6851079	0.2327935	0.4500978	0.9201180	0.4500978
Class: Classical	0.8483569	0.7258225	0.7272727	0.9694410	0.7272727
Class: Country	0.6629773	0.5101626	0.3814590	0.9444956	0.3814590
Class: Electronic	0.7215699	0.4592145	0.5022026	0.9409371	0.5022026
Class: Hip-Hop	0.6842137	0.7749529	0.3985472	0.9698803	0.3985472
Class: Jazz	0.6380075	0.2805611	0.3539823	0.9220328	0.3539823
Class: Rap	NA	0.0000000	NA	0.9000000	NA
Class: Rock	0.6898576	0.6915423	0.4169166	0.9627985	0.4169166

```
[1] "Macro Averages:"
```

Balanced Accuracy: <NA> Precision: 0.4429 Sensitivity: <NA> Specificity: 0.9395 Recall: <NA>

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the test data is found to be 0.82. This is slightly less than train data AUC.

```
# finding AUC
dt_test_=predict(model_dt, test_set, type="prob")
dt_test_AUC=multiclass.roc(test_set$music_genre,dt_test_)
print(paste("Decision Tree Training AUC:",round(dt_test_AUC$auc,4)))
dt_test_ROC=dt_test_AUC$rocs
```

```
[1] "Decision Tree Training AUC: 0.8256"
```

INFERENCE :

From the performance metrics of the train and test models of the decision tree it could be inferred that the model clearly fails to classify certain genres. This could be a reason for the drop in accuracy of the model.

5. K NEAREST NEIGHBOURS CLASSIFIER:

Building the model:

The K-Nearest Neighbors Classification algorithm classifies observations by allowing the K observations in the training set that are nearest to the new observation to “vote” on the class of the new observation.

The class package is used to call the multiclass knn() classifier model. The model is built by passing the training data and the features. The built model is stored under the knn_op variable. The knn_op is used to predict the genre from testing data. This output is stored in the knn_table variable. The corresponding confusion matrix is calculated using the confusionMatrix() function.

```
#building model
require(class)
knn_op=knn(train_set[,-12],test_set[,-12],train_set$music_genre,k=15,prob = TRUE)
```

```
Loading required package: class
```

```
knn_table=confusionMatrix(table(test_set$music_genre,knn_op))
print(knn_table)
```

Confusion Matrix:

The confusion matrix for each of the classes is displayed. The accuracy of the model is found to be 43%.

```
Confusion Matrix and Statistics

             knn_op
              Alternative Anime Blues Classical Country Electronic Hip-Hop Jazz
  Alternative         335     5    14         1     164         24     102   45
  Anime                28   664    84        71      43         72       2   24
  Blues                51   122   390        40     151         76      10  101
  Classical            23    48    36       763      22         18       1   83
  Country             146    14    49         6     435         21      32   89
  Electronic          139   118   128        17     103        258      18  151
  Hip-Hop             103     1     5         0      41          1     289   26
  Jazz                 79    13   102       109     147        102      34  346
  Rap                  47     1     4         0      24          0     327    5
  Rock                 92     2     2         3      63          2      97   15
             knn_op
              Rap Rock
  Alternative  92  191
  Anime         1    5
  Blues        13   34
  Classical     1    8
  Country      53  139
  Electronic   19   42
  Hip-Hop     425  171
  Jazz         22   44
  Rap         382  210
  Rock        204  525

Overall Statistics

               Accuracy : 0.4387
                 95% CI : (0.4289, 0.4485)
    No Information Rate : 0.1369
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3763
```

Performance Metrics:

The pROC library is imported to enable ROC plotting. We store the balanced accuracy, precision, sensitivity, specificity and recall of each of the 10 classes in our target variable into the knn_test_pm variable. The performance metrics are then displayed in a table format. We then take the macro average of each of the performance metrics.

```
# KNN test Performance Parameters
require(pROC)
# Testing Data
knn_test_pm=knn_table$byClass[, c("Balanced Accuracy", "Precision", "Sensitivity", "Specificity", "Recall")]
print("KNN Testing Data Performance Parameters:")
knn_test_pm
knn_test_macavg=round(apply(knn_test_pm,2,mean),4)
print("Macro Averages:")
knn_test_macavg
```

```
[1] "KNN Testing Data Performance Parameters:"
```

A matrix: 10 × 5 of type dbl

	Balanced Accuracy	Precision	Sensitivity	Specificity	Recall
Class: Alternative	0.6249798	0.3442960	0.3211889	0.9287708	0.3211889
Class: Anime	0.8177235	0.6680080	0.6720648	0.9633822	0.6720648
Class: Blues	0.7070082	0.3947368	0.4791155	0.9349009	0.4791155
Class: Classical	0.8643746	0.7607178	0.7554455	0.9733037	0.7554455
Class: Country	0.6511451	0.4420732	0.3646270	0.9376632	0.3646270
Class: Electronic	0.6857508	0.2598187	0.4494774	0.9220242	0.4494774
Class: Hip-Hop	0.6159144	0.2721281	0.3168860	0.9149428	0.3168860
Class: Jazz	0.6597150	0.3466934	0.3909605	0.9284696	0.3909605
Class: Rap	0.6224292	0.3820000	0.3151815	0.9296768	0.3151815
Class: Rock	0.6639391	0.5223881	0.3834916	0.9443865	0.3834916

```
[1] "Macro Averages:"
```

Balanced Accuracy: 0.6913 Precision: 0.4393 Sensitivity: 0.4448 Specificity: 0.9378 Recall: 0.4448

Area Under the Curve:

The AUC of the test data is found by using the multiclass.roc() function. The AUC value is rounded to 4 decimal places and displayed. The AUC value of the test data is found to be 0.71.

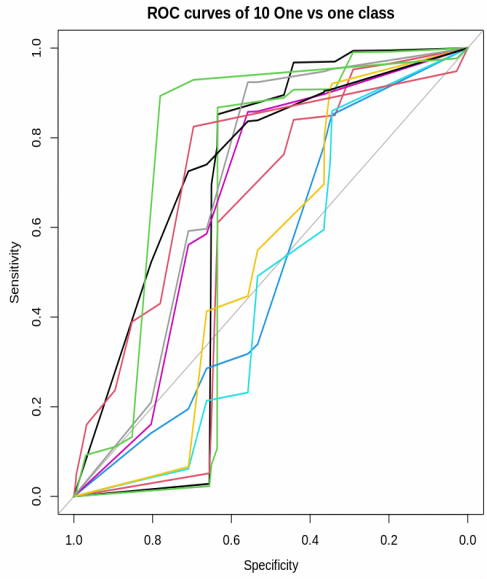
```
# finding AUC
#knn_test_=predict(knn_op,type="prob")
knn_test_AUC=multiclass.roc(test_set$music_genre,as.numeric(knn_op))
print(paste("KNN Testing AUC:",round(knn_test_AUC$auc,4)))
knn_test_ROC=knn_test_AUC$rocs
```

```
[1] "KNN Testing AUC: 0.7137"
```

Receiver Operating Characteristic Curve:

The ROC curve of the test data for each of the ten classes are plotted using the multiclass.roc function.

```
#Plotting ROC
knn_test_ROC=knn_test_AUC$rocs
plot.roc(knn_test_ROC[[1]],col=1,main="ROC curves of 10 One vs one class")
for(i in 2:11)
{num=paste("1/",as.character(i),sep="")
lines.roc(knn_test_ROC[[i]],col=i)
}
```



Bias and Variance:

The bias and variance of the test data are calculated using bias() and variance() functions. The variance is found to be 4.05 and bias 0.16.

```
#Bias and variance
print(var(as.numeric(knn_op), as.numeric(as.factor(test_set$music_genre))))
bias(as.numeric(knn_op), as.numeric(as.factor(test_set$music_genre)))
```

```
[1] 4.057175
0.1689
```

COMPARSION OF THE MODELS:

NAME	TEST ACCURACY	BIAS	VARIANCE	AUC
Logistic Regression	52	4.03	0.08	0.73
Naïve Bayes	42	2.7	0.12	0.70
Support Vector Machine	56	4.4	0.28	0.74
Decision Tree	44	-	-	0.82
K Nearest Neigbour	43	4.05	0.16	0.71

CONCLUSION:

From the table, it could be seen that SVM has higher accuracy compared to all the other classifiers. Decision tree has higher AUC compared to the other models. But when we look the bias of all the models, Naïve Bayes has a lower bias compared to all the other models. In terms of variance, Logistic regression has the least variance. In general, bias and variance has to be low for a good model, but in our case we have a higher bias for all the models and lower bias values. This could be due to variance bias tradeoff. To improve the accuracy of the model, ensemble techniques could be implemented in future.

REFERENCES:

https://www.r-bloggers.com/2021/05/principal-component-analysis-pca-in-r/

https://rpubs.com/jsl0516/spotify_genres

https://www.kaylinpavlik.com/classifying-songs-genres/

https://rstudio-pubs-static.s3.amazonaws.com/592175_dfd8ef3b109e479f9cd4f0509fa71dcb.html

https://odsc.medium.com/build-a-multi-class-support-vector-machine-in-r-abcdd4b7dab6