CS-GY 6923 Machine Learning
Fall 2022

Professor: Dr. Raman Kannan

Classification and Performance Analysis Report

Music Genre Classification

Author: Atmaja Raman
Net ID: ar6871

\

Table of Contents

Introduction

Multiclass classification problem classifies the instances into three or more classes. Each instance has exactly one label and there can be three or more labels overall. There are several algorithms that can be used to perform multiclass classification, but this report explores the following methods:

- Multinomial logistic regression
- Decision Trees
- K Nearest Neighbors
- Support Vector Machines

These classification models are executed for the Music genre classification dataset which has 10 classes in the target variable - music_genre. The model performances are analyzed and compared using following metrics:

- Accuracy
- Confusion matrix
- Sensitivity/recall
- Specificity
- Precision
- F1 score
- ROC curve
- Area under Curve (AUC)
- Bias
- Variance
- Kappa

Dataset link: https://www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre

Review

This section gives a recap of the exploratory data analysis task and summarises its results.

Number of attributes: **18**
Number of independent attributes: **17**
Number of dependent attributes: **1**
Target variable: **music_genre**
Number of classes in target: **10**

The dataset was cleaned to remove null values and faulty data types. Data imputation was performed to impute missing values in the tempo column. The data was checked for class imbalance and outliers were removed for the duration column. The categorical variables were encoded either using one-hot encoding or label encoding. Histograms and bar plots were plotted to understand the data distribution and correlation matrices were plotted to find highly correlated features. The data also underwent standard scaling to ensure that each feature contributes equally to the model and does not cause bias. Lastly, feature selection was performed to find the most important features that would influence the model.

Encoding for target variable

Feature	Label assigned
Alternative	1
Anime	2
Blues	3
Classical	4
Country	5
Electronic	6
Hip-Hop	7
Jazz	8
Rap	9
Rock	10

Loading the dataset

The cleaned dataset from the EDA task has been loaded and used for the classification task.

```
11] data = read.csv("/content/clean_music_genre.csv")
```

```
head(data)  #scaled and clean data after EDA
```

A data.frame: 6 × 30

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key_A	⋯	key_G.	liveness	loudness	mode_Major	mode_Minor	speechiness
	<int>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	⋯	<int>	<dbl>	<dbl>	<int>	<int>	<dbl>
1	46652	Thievery Corporation	The Shining Path	-0.8620880	-0.8545075	0.34696392	-0.2575814	1.09107118	2.3928336	0	⋯	0	-0.4333897	0.3287244	0	1	-0.6285765
2	30097	Dillon Francis	Hurricane	-1.0556701	-0.8829633	0.33565847	-0.2901090	0.57717063	-0.5137700	0	⋯	1	2.1164068	0.7267707	1	0	-0.5843629
3	62177	Dubloadz	Nitro	-0.6685058	-0.8170190	1.20617823	-0.8816497	0.36780374	-0.5424890	0	⋯	0	-0.2281622	0.7462956	1	0	1.4249000
4	24907	What So Not	Divide & Conquer	-0.7975606	-0.8782699	0.43740753	-0.2081104	-0.06235005	2.2658130	0	⋯	0	-0.2281622	0.4562108	1	0	-0.5175512
5	89064	Axel Boman	Hello	0.1703501	-0.8765578	1.09877644	3.3978266	0.48581053	2.0954195	0	⋯	0	0.1387598	-0.2412726	0	1	-0.5185337
6	43760	Jordan Comolli	Clash	0.1058227	-0.8066875	0.06432763	-0.3047343	0.75989083	-0.5503031	0	⋯	0	-0.5453320	0.7797669	1	0	2.5253277

This dataset is then split into train and test by using random sampling. Random sampling would be sufficient in this case as each target label has similar proportions of data and is equally represented in the dataset, so stratified sampling was not used. The train dataset has 31268 instances and the test has 13400 instances and both have 30 features in total (after one hot encoding). The proportion of data for each class under the train and test dataset is also similar as shown below.

```
#test train split
set.seed(6871)

sample = sample.split(data$music, SplitRatio = 0.7)
train = subset(data, sample == TRUE)
test = subset(data, sample == FALSE)
```

```
[14] dim(train)
     dim(test)
```

```
31268 · 30
13400 · 30
```

```
[15]  table(train$music_genre) %>% prop.table()
```

```
         1          2          3          4          5          6          7 
0.10090188 0.10112575 0.10019829 0.09428169 0.10093386 0.10064603 0.10093386 
         8          9         10 
0.10003838 0.10048612 0.10045414 
```

```
[16]  table(test$music_genre) %>% prop.table()
```

```
         1          2          3          4          5          6          7 
0.10089552 0.10111940 0.10022388 0.09432836 0.10089552 0.10067164 0.10097015 
         8          9         10 
0.10000000 0.10044776 0.10044776 
```

Performance Metrics

This section gives a brief introduction to the metrics used to analyze the model performance and how to interpret the results. For multi-class classification where the target variable classes go from class 1 to n:

- True positive of class 1 is, all class 1 instances that are classified as class 1.
- True negative of class 1 is all non class 1 instances that are not classified as class 1.
- False positive of class 1 is all non class 1 instances that are classified as class 1.
- False negative of class 1 is all class 1 instances that are not classified as class 1.

The following performance metrics are used:

a. ROC - A ROC curve is a graph showing the performance of a classification model at all classification thresholds.

b. AUC - Area under the ROC curve. AUC ranges in value from 0 to 1. A model whose predictions are completely wrong has an AUC of 0, one whose predictions are fully correct has an AUC of 1.

c. Confusion Matrix - It is a matrix with two dimensions actual and predicted where each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. From this matrix the true positive, false positive, true negative and false negative values can be obtained.

d. Accuracy - It's the ratio of the correctly labeled instances to the entire set of instances. The sum of true positive and false negative is divided by the total number of events.

e. Specificity - Specificity measures the rate of actual negatives identified correctly. It is the number of true negatives divided by the sum of true positives and false positives.

f. Precision - Precision identifies how accurately the model predicted the positive classes. The number of true positive events is divided by the sum of positive true and false events.

g. Recall/Sensitivity - Recall/sensitivity measures the ratio of predicted positive classes. The number of true positive events is divided by the sum of true positive and false negative events.

h. F1-score - The F1 score is the weighted average score of recall and precision. The value at 1 is the best performance and at 0 is the worst.

i. Prevalence - Prevalence represents how often positive events occurred. The sum of true positive and false negative events is divided by the total number of events.

j. Balanced accuracy - Balanced accuracy is the average of both sensitivity and specificity. The balanced accuracy is in the range of 0 to 1 where a value of 0 indicates the worst possible classifier and 1 indicates the best-possible classifier.

k. Variance - Variance is the variability of model prediction for a data point which tells us the spread of our data. A model with high variance focuses a lot on the training data and does not generalize on unseen data. Such models perform very well on training data but have high error rates on test data.

l. Bias - Bias is the difference between the average prediction of the model and the correct value. A model with high bias pays less heed to training data and oversimplifies the model. It always leads to high errors in training and test data.

m. Kappa - Kappa is a measure of agreement between the predictions and the actual labels. It can be considered as the comparison of overall accuracy to the expected random chance accuracy.

Principal Component Analysis

Principal component analysis or PCA is a dimensionality reduction method that transforms a dataset with a large number of variables into a small number of components that hold most of the information. The principal components represent the directions of data that has a maximum variance. The new components formed are uncorrelated and the first few components contain most of the information.

Performing PCA involves:

- Selecting numerical variables and scaling them.
- Computing covariance matrix - The covariance matrix shows the correlation between the features of the dataset.
- Computing Eigen Vectors and Values - These represent the amount of variance carried by each principal component.
- Recasting the data along the principal component.

```
summary(pca)      #PC1 explains only 32% of variability, PC2 12%, PC3 and PC4
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7
Standard deviation     1.9002 1.1779 1.02449 0.99724 0.97296 0.90123 0.85985
Proportion of Variance 0.3289 0.1264 0.09561 0.09059 0.08623 0.07399 0.06735
Cumulative Proportion  0.3289 0.4553 0.55092 0.64151 0.72774 0.80173 0.86908
                           PC8     PC9    PC10    PC11
Standard deviation     0.77505 0.68177 0.50983 0.33437
Proportion of Variance 0.05472 0.04234 0.02368 0.01018
Cumulative Proportion  0.92380 0.96614 0.98982 1.00000
```

Component	Variability
PC1	32%
PC2	12%
PC3	9%
PC4	9%
PC5	8%
PC6	7%
PC7	7%

From the above snippet we can see that PC1 holds only 32% of variability, PC2 hold 12% of variability, PC3 and PC4 holds 9% variability each, PC5 holds 8%, PC6, and PC7 approximately 7% each. Together PC1 to PC7 capture about 84% of variability or information in total.

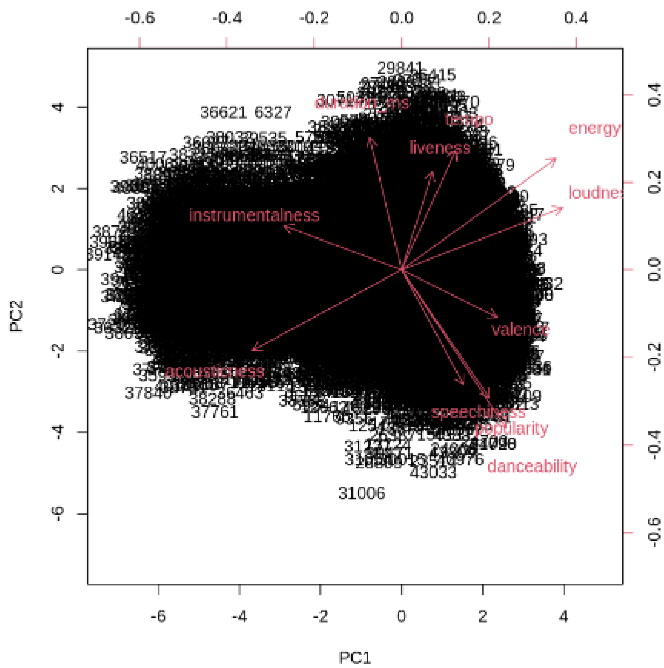
	PC1	PC2	PC3	PC4	PC5
popularity	0.25067964	-0.3649218	0.14472027	0.454255720	-0.159176946
acousticness	-0.42665785	-0.2304438	0.04630158	-0.148953938	-0.008110264
danceability	0.29850064	-0.4517801	-0.13476091	0.006424318	0.225238136
duration_ms	-0.09107364	0.3770738	0.09174213	0.637439588	0.319945649
energy	0.44170113	0.3175934	-0.06499701	-0.004046115	0.065320788
instrumentalness	-0.33666187	0.1256773	-0.09680272	-0.091045771	0.076963042
liveness	0.08867115	0.2798165	0.62126377	-0.378562865	0.420714513
loudness	0.46012072	0.1772571	-0.06584409	0.078734848	0.006929444
speechiness	0.17670321	-0.3268579	0.63110605	-0.064336863	-0.171073828
tempo	0.15494600	0.3403975	-0.02097039	-0.221804000	-0.698520930
valence	0.27290790	-0.1354757	-0.38431885	-0.392355700	0.342144573

	PC6	PC7	PC8	PC9	PC10
popularity	-0.19180963	-0.348606601	0.53019965	-0.332555151	0.028999925
acousticness	0.03413950	-0.323647851	-0.13946263	-0.069433825	0.738537855
danceability	0.34295320	-0.005046169	0.18001949	0.674575346	0.064777571
duration_ms	0.48529691	-0.258556222	-0.17052455	-0.004162226	0.027694833
energy	-0.03247424	0.259159635	0.03409937	-0.211862173	0.206834109
instrumentalness	0.37591925	0.421360401	0.67969034	-0.164557052	0.155786571
liveness	-0.15303429	-0.289335527	0.27948823	0.145494171	-0.014947006
loudness	-0.13265003	0.157062493	-0.05505726	0.066737350	0.608854538
speechiness	0.43847506	0.319713471	-0.26430705	-0.251503135	0.009407261
tempo	0.36730260	-0.378423316	0.13312775	0.163159676	-0.001016430
valence	0.31361856	-0.329144094	-0.07800609	-0.491990706	-0.103397817

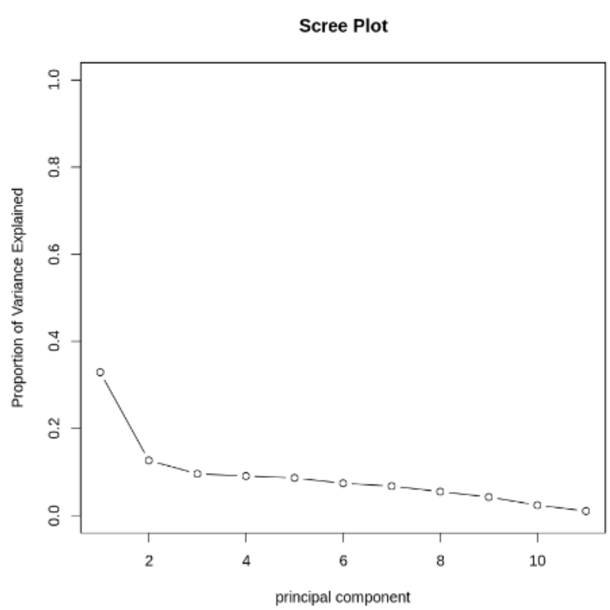
	PC11
popularity	-0.022343005
acousticness	-0.254635737
danceability	-0.168999235
duration_ms	0.013510387
energy	-0.733811817
instrumentalness	0.122854696
liveness	0.030343669
loudness	0.570817421
speechiness	0.050615798
tempo	-0.009617737
valence	0.151450897

The output above shows the correlation between the features and the principal components. We can see that popularity, danceability, energy, loudness, etc. are positively correlated with PC1 because of the positive value whereas, acosuticness, duration, and instrumentalness are negatively correlated to PC1 because of the negative sign.

The biplot shown below represents the multidimensional dataset in 2D. If the angle between two variables is 90 degrees, then there is no correlation between the two variables. If it's less than 90 degrees, there is a positive correlation between the two variables and if it's more than 90 degrees, there is a negative correlation between the two variables. We can see energy and loudness have a small angle which means they are highly positively correlated whereas instrumentalness and danceability have an angle greater than 90 which shows a negative correlation. The longer the vector line, the greater the variance, and the shorter the line, the lesser the variance. Valence has less variance as compared to energy or loudness.



A scree plot is a line plot that shows the eigenvalues for each principal component and is always a downward curve. From the plot below, we can see that the first component explains the highest variability, the next few components explain a moderate amount, and the latter explains a small fraction of the overall variability.



Multinominal Logistic Regression

Multinominal logistic regression is a method that applies classification to multiclass problems. It is used when the dependent variable or target variable is nominal i.e it does not have any order associated with it. The data needs to have certain properties in order to get accurate results when using the model, this is called assumptions of the model. The assumptions for multinomial logistic regression are no outliers, independence, and no multicollinearity.

The model was run on two different datasets:

- Principal components that were recast on the original dataset
- Selecting important features from the original dataset

Taking Principal components

Initially, I hypothesized that the principal component model would give better accuracy as compared to the model using important features as the principal components are uncorrelated and multinominal logistic regression assumes that there is no multicollinearity in the data. But on executing both the models, we see that they have similar performance and in fact, the model run on selected features has slightly better performance. This might be because dimensionality was not reduced significantly using PCA. We still need to consider at least 7-8 principal components to capture a good amount of information from this dataset.

Taking Principal Components

```
train_pca = predict(pca, train)
train_pca = data.frame(train_pca, train[30])
test_pca = predict(pca, test)
test_pca = data.frame(test_pca, test[30])
```

```
pca_model = multinom(music_genre~., data = train_pca)
```

```
# weights:  130 (108 variable)
initial  value 71997.230688
iter  10 value 42682.538645
iter  20 value 41901.892314
iter  30 value 41699.012692
iter  40 value 41260.041917
iter  50 value 40955.225691
iter  60 value 40630.993482
iter  70 value 40520.326122
iter  80 value 40419.120930
iter  90 value 40369.493342
iter 100 value 40288.292858
final  value 40288.292858
stopped after 100 iterations
```

```
mean(pred_pca == test_pca$music_genre)
```

```
0.519850746268657
```

```
confusionMatrix(pred_pca, as.factor(test_pca$music_genre))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   404   40   41   21   70   75   83   26   98  151
        2    10  831  245   78   21  114    0   58    0    3
        3    30  157  609   26  155   84    0  186    0    8
        4     4  160   35 1035    3   17    0   96    0    1
        5   332   54  150    9  758   70   58  146   33  154
        6    97   78   61   26   43  767   22  205    3   12
        7   122    0    4    0   14   35  621   40  541   19
        8    70   29  145   64   57  125   19  525    0   28
        9    33    1    3    0   14   22  453    9  520   74
        10  250    5   50    5  217   40   97   49  151  896

Overall Statistics

               Accuracy : 0.5199
                 95% CI : (0.5114, 0.5283)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4665

 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.29882  0.61328  0.45346  0.81883  0.56065  0.56857
Specificity           0.94978  0.95608  0.94642  0.97396  0.91650  0.95461
Pos Pred Value        0.40040  0.61103  0.48526  0.76610  0.42971  0.58371
Neg Pred Value        0.92349  0.95648  0.93956  0.98099  0.94895  0.95185
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03015  0.06201  0.04545  0.07724  0.05657  0.05724
Detection Prevalence  0.07530  0.10149  0.09366  0.10082  0.13164  0.09806
Balanced Accuracy     0.62430  0.78468  0.69994  0.89640  0.73858  0.76159
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.45898  0.39179  0.38633   0.66568
Specificity           0.93567  0.95547  0.94948   0.92832
Pos Pred Value        0.44484  0.49435  0.46058   0.50909
Neg Pred Value        0.93902  0.93394  0.93269   0.96134
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.04634  0.03918  0.03881   0.06687
Detection Prevalence  0.10418  0.07925  0.08425   0.13134
Balanced Accuracy     0.69732  0.67363  0.66790   0.79700
```

```
genrepca = multiclass.roc(as.numeric(test_pca$music_genre), pred_testpca)
auc(genrepca)
```

```
0.903532305838732
```

```
var(as.numeric(pred_pca), as.numeric(test_pca$music_genre))

bias(as.numeric(pred_pca), as.numeric(test_pca$music_genre))
```

```
4.06186947566816
0.139179104477612
```

Taking Features based on feature importance

In the EDA task, feature importance was calculated by running random forest feature importance and we got the following order of important features: popularity > danceability > speechiness > acousticness > instrumentalness > track name > energy > loudness > tempo > duration > tempo > artist > valence > liveness > mode > key. From the correlation matrix, we know that loudness and energy are very highly correlated. So loudness was dropped and only energy was considered in the feature list. In the code below, I have taken multiple train data frames with different sets of features. The logistic regression model was trained for each train data frame and their accuracies were compared to find the optimum number of features that gives the best-performing model.

Taking features based on feature importance

```
#creating df with varying number of top important features, take either energy or loudness(high corr)

train0 = subset(train, select = c(4,5,6,7,8,9,10:21,22,24,25,26,27,29,30))  #24 features
train1 = subset(train, select = c(4,5,6,7,8,9,22,24,25,26,27,29,30))  #12 features
train2 = subset(train, select = c(4,5,6,7,8,9,22,26,27,29,30))  #10 features
train3 = subset(train, select = c(4,5,6,8,9,26,27,30))  #7 features
train4 = subset(train, select = c(4,5,6,8,9,26,30))  #6 features
```

```
model0 = multinom(music_genre~., data = train0)
model1 = multinom(music_genre~., data = train1)
model2 = multinom(music_genre~., data = train2)
model3 = multinom(music_genre~., data = train3)
model4 = multinom(music_genre~., data = train4)
```

```
# weights:  260 (225 variable)
initial  value 71997.230688
iter  10 value 42507.128406
iter  20 value 41995.456415
iter  30 value 41783.933158
iter  40 value 41106.894329
iter  50 value 40433.833760
iter  60 value 40346.757868
iter  70 value 40185.701697
iter  80 value 40090.592677
iter  90 value 40079.102759
iter 100 value 40070.970566
final  value 40070.970566
stopped after 100 iterations
```

```
[ ] pred0 = model0 %>% predict(test)
    pred1 = model1 %>% predict(test)
    pred2 = model2 %>% predict(test)
    pred3 = model3 %>% predict(test)
    pred4 = model4 %>% predict(test)
```

```
[ ] mean(pred0 == test$music_genre)

    0.519402985074627
```

```
[ ] mean(pred1 == test$music_genre)

    0.51910447761194
```

```
[ ] mean(pred2 == test$music_genre)

    0.513880597014925
```

```
[ ] mean(pred3 == test$music_genre)

    0.488507462686567
```

```
[ ] mean(pred4 == test$music_genre)

    0.486268656716418
```

From the snippet above, we can see that the test accuracy for model0 with 24 features and the model1 with 12 features are similar. The other models have lesser accuracy. Since model0 and model1 have similar performance, I considered model1 with 12 features for further analysis as it takes less computation time and resources. The screenshots below show the train and test statistics such as confusion matrix, accuracy, specificity, sensitivity, precision, f1 score of each target class.

Summary of insights from training

- The training accuracy is 0.51
- Sensitivity/ recall is highest for target class 4 - classical with a value of 0.78, which implies the model could predict the classical genre well.
- The specificity is high for all classes which implies that instances not belonging to a certain class were identified as not belonging to that class
- Precision is highest for classical (class 4) with a value of 0.71 and lowest for class alternative (class 1) with a value of 0.39 which implies that classical genre is predicted well by the model and alternative genre is not predicted that well.
- F1 score is highest for the classical genre and lowest for the alternative genre for the train data.
- Balanced accuracy is highest for classical (class 4) with a value of 0.87 and lowest for class alternative (class 1) with a value of 0.62 which implies that the classical is classified well and alternative genre is classified poorly.

```
pred1_train = model1 %>% predict(train)
mean(pred1_train == train$music_genre)
```

0.519476781373929

```
cmtrain = confusionMatrix(pred1_train, as.factor(train$music_genre))  #train
cmtrain
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   980   69  116   54  186  199  236   83  213  366
        2    20 1846  509  194   48  248    0  124    3    8
        3    71  392 1443   83  310  175    5  402    0    8
        4     9  467   86 2302   16   39    0  292    0    9
        5   719  122  365   33 1821  165  106  285  107  384
        6   196  192  155   95  101 1835   35  497   20   28
        7   264    0    6    0   61   84 1559   77 1193   60
        8   218   60  329  172  111  280   40 1257   30   79
        9   122    2    3    0   28   46  995    9 1189  188
        10  556   12  121   15  474   76  180  102  387 2011

Overall Statistics

               Accuracy : 0.5195
                 95% CI : (0.5139, 0.525)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4661

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.31062  0.58381  0.46058  0.78087  0.57700  0.58310
Specificity           0.94586  0.95894  0.94860  0.96758  0.91868  0.95310
Pos Pred Value        0.39169  0.61533  0.49948  0.71491  0.44339  0.58180
Neg Pred Value        0.92439  0.95345  0.94045  0.97697  0.95085  0.95333
Prevalence            0.10090  0.10113  0.10020  0.09428  0.10093  0.10065
Detection Rate        0.03134  0.05904  0.04615  0.07362  0.05824  0.05869
Detection Prevalence  0.08002  0.09594  0.09239  0.10298  0.13135  0.10087
Balanced Accuracy     0.62824  0.77137  0.70459  0.87423  0.74784  0.76810
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.49398  0.40185  0.37842   0.64024
Specificity           0.93793  0.95313  0.95047   0.93163
Pos Pred Value        0.47185  0.48797  0.46050   0.51118
Neg Pred Value        0.94289  0.93479  0.93192   0.95866
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.04986  0.04020  0.03803   0.06431
Detection Prevalence  0.10567  0.08238  0.08258   0.12582
```

```
cmtrain$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3106181	0.9458613	0.3916867	0.9243899	0.3916867	0.3106181	0.3464734	0.10090188	0.03134195	0.08001791	0.6282397
Class: 2	0.5838077	0.9589412	0.6153333	0.9534456	0.6153333	0.5838077	0.5991561	0.10112575	0.05903799	0.09594474	0.7713744
Class: 3	0.4605809	0.9486049	0.4994808	0.9404489	0.4994808	0.4605809	0.4792428	0.10019829	0.04614942	0.09239478	0.7045929
Class: 4	0.7808684	0.9675847	0.7149068	0.9769681	0.7149068	0.7808684	0.7464332	0.09428169	0.07362159	0.10298068	0.8742266
Class: 5	0.5769962	0.9186824	0.4433893	0.9508486	0.4433893	0.5769962	0.5014457	0.10093386	0.05823845	0.13134834	0.7478393
Class: 6	0.5830950	0.9530956	0.5818009	0.9533329	0.5818009	0.5830950	0.5824472	0.10064603	0.05868620	0.10086990	0.7680953
Class: 7	0.4939797	0.9379269	0.4718523	0.9428909	0.4718523	0.4939797	0.4826625	0.10093386	0.04985928	0.10566714	0.7159533
Class: 8	0.4018542	0.9531272	0.4879658	0.9347902	0.4879658	0.4018542	0.4407433	0.10003838	0.04020084	0.08238455	0.6774907
Class: 9	0.3784214	0.9504729	0.4604957	0.9319180	0.4604957	0.3784214	0.4154437	0.10048612	0.03802610	0.08257644	0.6644471
Class: 10	0.6402420	0.9316315	0.5111845	0.9586595	0.5111845	0.6402420	0.5684806	0.10045414	0.06431495	0.12581553	0.7859367

```
[ ] cmtest =confusionMatrix(pred1, as.factor(test$music_genre))        #test
    cmtest
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   422   35   50   23   74   74   84   35  105  150
        2    12  767  246   93   17  129    0   50    0    3
        3    42  188  600   22  132   77    2  151    0    6
        4     4  208   43 1013   11   21    0  119    0    1
        5   306   50  152    8  796   51   57  145   31  172
        6   100   73   53   25   45  772   20  204    4   13
        7   112    0    4    0   16   38  651   40  546   22
        8    79   26  141   68   48  125   18  544    6   27
        9    45    1    3    0    8   26  428    8  507   68
        10  230    7   51   12  205   36   93   44  147  884

Overall Statistics

               Accuracy : 0.5191
                 95% CI : (0.5106, 0.5276)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4657

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.31213  0.56605  0.44676  0.80142   0.5888  0.57228
Specificity           0.94771  0.95434  0.94858  0.96646   0.9193  0.95544
Pos Pred Value        0.40114  0.58238  0.49180  0.71338   0.4502  0.58976
Neg Pred Value        0.92468  0.95134  0.93900  0.97905   0.9522  0.95228
Prevalence            0.10090  0.10112  0.10022  0.09433   0.1009  0.10067
Detection Rate        0.03149  0.05724  0.04478  0.07560   0.0594  0.05761
Detection Prevalence  0.07851  0.09828  0.09104  0.10597   0.1319  0.09769
Balanced Accuracy     0.62992  0.76019  0.69767  0.88394   0.7540  0.76386
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.48115  0.40597  0.37667   0.65676
Specificity           0.93542  0.95539  0.95130   0.93156
Pos Pred Value        0.45556  0.50277  0.46344   0.51726
Neg Pred Value        0.94136  0.93538  0.93182   0.96048
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.04858  0.04060  0.03784   0.06597
Detection Prevalence  0.10664  0.08075  0.08164   0.12754
Balanced Accuracy     0.70829  0.68068  0.66399   0.79416
```

```
[ ] cmtest$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3121302	0.9477092	0.4011407	0.9246842	0.4011407	0.3121302	0.3510815	0.10089552	0.03149254	0.07850746	0.6299197
Class: 2	0.5660517	0.9543379	0.5823842	0.9513366	0.5823842	0.5660517	0.5741018	0.10111940	0.05723881	0.09828358	0.7601948
Class: 3	0.4467610	0.9485776	0.4918033	0.9389984	0.4918033	0.4467610	0.4682013	0.10022388	0.04477612	0.09104478	0.6976693
Class: 4	0.8014241	0.9664634	0.7133803	0.9790484	0.7133803	0.8014241	0.7548435	0.09432836	0.07559701	0.10597015	0.8839437
Class: 5	0.5887574	0.9193227	0.4502262	0.9522008	0.4502262	0.5887574	0.5102564	0.10089552	0.05940299	0.13194030	0.7540401
Class: 6	0.5722758	0.9554394	0.5897632	0.9522786	0.5897632	0.5722758	0.5808879	0.10067164	0.05761194	0.09768657	0.7638576
Class: 7	0.4811530	0.9354196	0.4555633	0.9413583	0.4555633	0.4811530	0.4680086	0.10097015	0.04858209	0.10664179	0.7082863
Class: 8	0.4059701	0.9553897	0.5027726	0.9353791	0.5027726	0.4059701	0.4492155	0.10000000	0.04059701	0.08074627	0.6806799
Class: 9	0.3766716	0.9513025	0.4634369	0.9318219	0.4634369	0.3766716	0.4155738	0.10044776	0.03783582	0.08164179	0.6639870
Class: 10	0.6567608	0.9315580	0.5172616	0.9604824	0.5172616	0.6567608	0.5787234	0.10044776	0.06597015	0.12753731	0.7941594

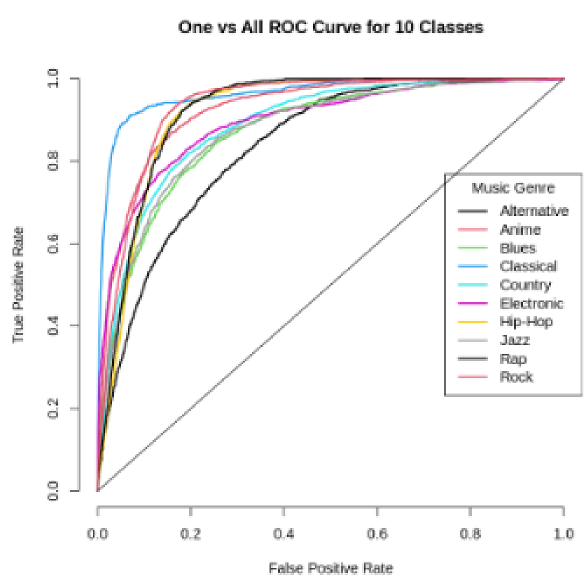
Summary of insights from testing

- The test accuracy is 0.51, similar to the train.
- Sensitivity/ recall is highest for the classical genre (label 4) with a value of 0.80, which implies the model could predict the classical genre well.
- The specificity is high ~0.93 to 0.95 for all classes which implies that instances not belonging to a certain class were identified as not belonging to that class correctly.
- Precision is highest for classical with a value of 0.71 and lowest for class alternative with a value of 0.40 which implies that the classical genre is predicted well by the model and the alternative genre is not predicted that well.
- F1 score is highest for the classical genre and lowest for the alternative genre for the train data.
- Balanced accuracy is highest for classical (class 4) with a value of 0.88 and lowest for class alternative (class 1) with a value of 0.62 which implies that the classical is classified well and alternative genre is classified poorly.

```
genre_roc = multiclass.roc(as.numeric(test$music_genre), pred_test)
auc(genre_roc)
```

0.903220505624288

```
[ ] multiclass_roc_plot(test, pred_test)
```




```
[ ] var(as.numeric(pred1), as.numeric(test$music_genre))

    bias(as.numeric(pred1), as.numeric(test$music_genre))
```

4.05280584539056
0.115597014925373

Summary of insights

- The area under the curve is 0.90 (close to 1.0, the baseline is 0.5) which implies the model's predictions are good
- From the multiclass roc plot, we can see that the model predicts classical, hip hop, and anime better than alternative or blues.
- The model has a slightly higher bias and a low variance value which might indicate some overfitting. Ideally, a low-bias low variance model is best but in reality, it is hard to achieve because of variance and bias trade-off.
- Kappa statistic has a value of 0.46 which means it's a decent model compared to random chance.
- The predictions are not very accurate i.e accuracy is not great and from the confusion matrix, we can see that there is a lot of misclassification. This might be because of the outliers in the numerical values. Although the outliers were in the accepted range of values of the parameter i.e 0 to 1, they seem to be impacting the performance of the model.

Decision Trees

Decision Trees perform multiclass classification by using tree-like structures. Each node corresponds to an attribute and each branch node corresponds to a decision-making node. The algorithm considers all of the predictor variables and selects the one that does the best job of discriminating the classes. It starts at the root and at each branch finds the next feature that will best discriminate the classes using entropy or the Gini index. The size of the tree changes the accuracy of the prediction. In general, if a tree is too big it overfits the data and gives poor accuracy. To decrease the size of the tree, one can prune the tree or remove certain branches. There are other parameters that help tune and improve the performance of the decision tree model, such as minsplit, minbucket, maxdepth, and complexity parameter (cp).

Taking Features based on feature importance

I have taken multiple train data frames with different sets of features. The decision tree model was trained for each train data frame and their accuracies were compared to find the optimum number of features that gives the best-performing model.

Decision Tree

```
[ ] traindata = subset(train, select = c(4,5,6,7,8,9,22,24,25,26,27,29,30))  #12 features
    traindata1 = subset(train, select = c(4,5,6,7,8,9,22,26,27,29,30))  #10 features
    traindata2 = subset(train, select = c(4,5,6,8,9,26,27,30))  #7 features
```

```
[ ] tree = rpart(music_genre~., data = traindata, method = "class", cp=0.001)    #ran model for different maxdepth, minsplit values - not much difference
                                                                                 #cp=0.1,0.01 - underfitting model, poor prediction cp=0.0001 - overfitting
```

```
[ ] best = tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
    best
```

```
0.00101401835906924
```

The above snippet shows the number of splits, relative errors, and cross-validation errors for different complexity parameter values. The summary also shows the variable importance and the node at each split, the branches, class counts, and their probabilities at each split. After trying out multiple values for the tuning parameters: minsplit, maxdepth, minbuckets and CP it was found that changes made to maxdepth, minsplit had minimal changes to model performance whereas changes to CP value impacted the model significantly. CP value of 0.001 gave the best performance. The prune function prunes or trims the tree at the node where xerror is the least, which is at 0.001.

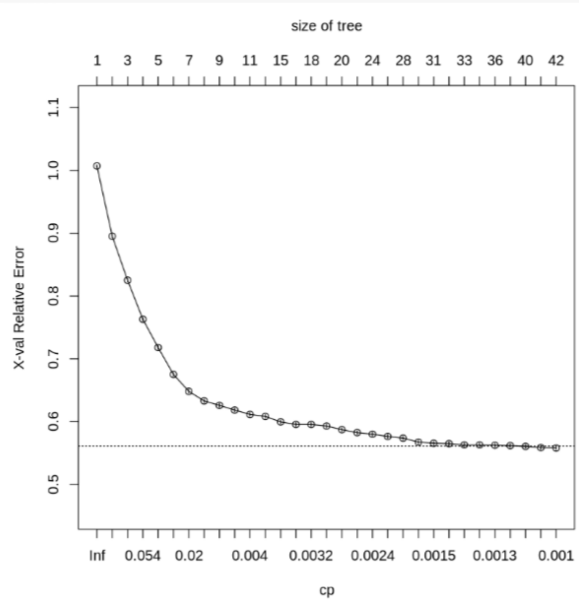
```
printcp(tree)
```

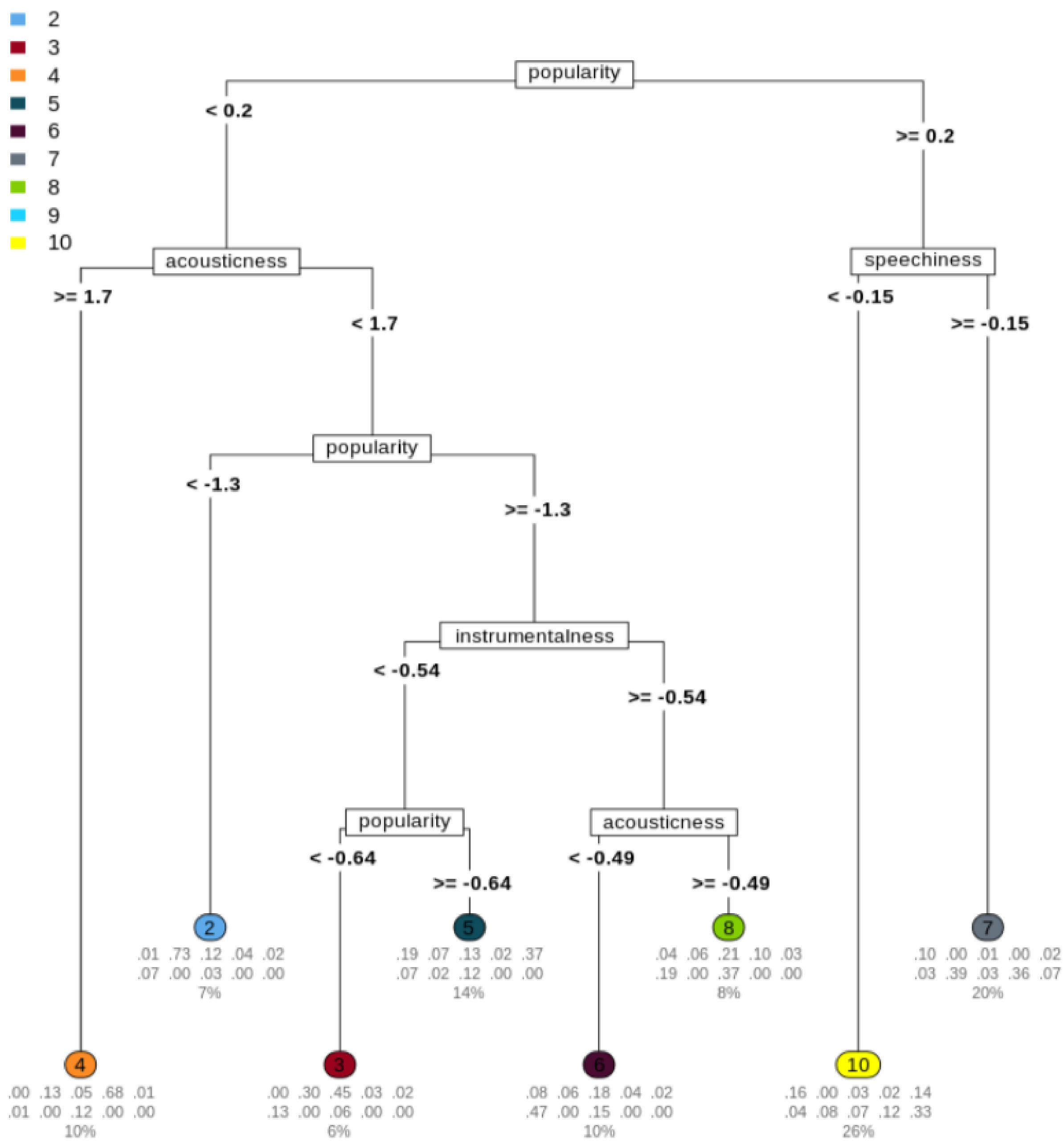
```
Classification tree:
rpart(formula = music_genre ~ ., data = traindata, method = "class",
    cp = 0.001)

Variables actually used in tree construction:
[1] acousticness     danceability     duration_ms      energy
[5] instrumentalness popularity       speechiness      tempo
[9] valence

Root node error: 28106/31268 = 0.89887

n= 31268

          CP nsplit rel error  xerror      xstd
1  0.1088380      0   1.00000 1.00801 0.0018354
2  0.0681349      1   0.89116 0.89337 0.0025022
3  0.0634740      2   0.82303 0.83167 0.0027331
4  0.0454351      3   0.75955 0.76475 0.0029164
5  0.0413435      4   0.71412 0.71874 0.0030085
6  0.0268982      5   0.67277 0.67509 0.0030731
7  0.0154415      6   0.64588 0.64819 0.0031025
8  0.0082545      7   0.63043 0.63332 0.0031154
9  0.0062976      8   0.62218 0.62716 0.0031201
10 0.0041272      9   0.61588 0.61791 0.0031263
11 0.0039315     10   0.61176 0.61129 0.0031303
12 0.0037003     12   0.60389 0.60873 0.0031317
13 0.0033445     14   0.59649 0.60005 0.0031360
14 0.0032377     16   0.58980 0.59624 0.0031376
15 0.0031310     17   0.58657 0.59617 0.0031376
16 0.0026685     18   0.58343 0.59404 0.0031385
17 0.0024550     19   0.58077 0.58813 0.0031406
18 0.0024016     21   0.57586 0.58472 0.0031416
19 0.0023127     23   0.57105 0.58276 0.0031422
20 0.0022652     24   0.56874 0.58013 0.0031428
21 0.0018146     27   0.56194 0.57297 0.0031443
22 0.0015833     28   0.56013 0.56860 0.0031450
23 0.0014232     30   0.55696 0.56607 0.0031452
24 0.0013520     31   0.55554 0.56479 0.0031454
25 0.0013164     32   0.55419 0.56322 0.0031455
26 0.0012987     33   0.55287 0.56322 0.0031455
27 0.0012809     35   0.55027 0.56333 0.0031455
28 0.0011563     37   0.54771 0.56258 0.0031455
29 0.0010674     39   0.54540 0.56109 0.0031456
30 0.0010318     40   0.54433 0.55871 0.0031457
31 0.0010140     41   0.54330 0.55839 0.0031457
32 0.0010000     43   0.54127 0.55824 0.0031457
```

```
summary(tree)
```

```
Variable importance
      popularity     acousticness instrumentalness      speechiness
              29               15               12               12
          energy     danceability          valence            tempo
              12               11                3                2
     duration_ms
               1

Node number 1: 31268 observations,    complexity param=0.108838
  predicted class=2   expected loss=0.8988742  P(node) =1
    class counts:  3155  3162  3133  2948  3156  3147  3156  3128  3142  3141
   probabilities: 0.101 0.101 0.100 0.094 0.101 0.101 0.101 0.100 0.100 0.100
  left son=2 (17147 obs) right son=3 (14121 obs)
  Primary splits:
      popularity       < 0.2026138   to the left,  improve=1865.3880, (0 missing)
      acousticness     < 1.718914    to the right, improve=1432.8180, (0 missing)
      energy           < -1.522208   to the left,  improve=1361.9300, (0 missing)
      instrumentalness < -0.1986969  to the right, improve= 964.0547, (0 missing)
      speechiness      < 0.2507831   to the left,  improve= 844.2832, (0 missing)
  Surrogate splits:
      instrumentalness < -0.5502145  to the right, agree=0.661, adj=0.250, (0 split)
      speechiness      < -0.1299452  to the left,  agree=0.633, adj=0.187, (0 split)
      danceability     < 0.4571921   to the left,  agree=0.627, adj=0.174, (0 split)
      acousticness     < 0.1662419   to the right, agree=0.585, adj=0.080, (0 split)
      energy           < -0.6200273  to the left,  agree=0.573, adj=0.055, (0 split)

Node number 2: 17147 observations,    complexity param=0.06347399
  predicted class=2   expected loss=0.8188021  P(node) =0.5483881
    class counts:  1223  3107  2839  2786  1928  2630   120  2444    28    42
   probabilities: 0.071 0.181 0.166 0.162 0.112 0.153 0.007 0.143 0.002 0.002
  left son=4 (3258 obs) right son=5 (13889 obs)
  Primary splits:
      acousticness     < 1.718914    to the right, improve=1274.5560, (0 missing)
      energy           < -1.522208   to the left,  improve=1210.5820, (0 missing)
      popularity       < -1.087934   to the left,  improve= 821.0962, (0 missing)
      danceability     < -0.8711985  to the left,  improve= 678.7281, (0 missing)
      instrumentalness < -0.1026569  to the left,  improve= 578.4684, (0 missing)
  Surrogate splits:
      energy           < -1.358521   to the left,  agree=0.931, adj=0.634, (0 split)
      danceability     < -1.696496   to the left,  agree=0.839, adj=0.152, (0 split)
      valence          < -1.460607   to the left,  agree=0.832, adj=0.118, (0 split)
      instrumentalness < 2.171322    to the right, agree=0.830, adj=0.104, (0 split)
      tempo            < -1.581863   to the left,  agree=0.826, adj=0.085, (0 split)
```

```
plotcp(tree)
```





The first figure shows the relative error and CP values with an increase in the size of the tree. The next figure shows the decision tree. The most important feature of the decision tree model is popularity. Then speechiness and acousticness help in splitting the tree further. Tracks with speechiness <= 0.15 are rock and the ones >=0.15 are hip-hop. Similarly tracks with acousticness >=1.7 are classical. Further, popularity, instrumentalness are used for splitting, and so on.

```r
cart_train = predict(tree, data = traindata, type = "class")
```

```r
cart_test = predict(object = tree, newdata = test, type = "class")
```

```r
carttrn = confusionMatrix(cart_train, as.factor(traindata$music_genre))
carttrn
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1111   93  218   60  475  269  167  316   64  277
        2    55 1971  377  184   72  211    6   81    6    8
        3    53  317 1229  101   57  270    1  272    2    7
        4    11  359   73 2269   20   30    0  270    0    2
        5   416  160  419   49 1555  183   22  286   13   82
        6   170  194  319  124   39 1645   12  591    3   14
        7   335    2   25    1  100  107 1646  116 1241  193
        8   102   43  321  138  136  243   12  977    8   23
        9   141    2   18    0   51   62 1011   56 1245  185
        10  761   21  134   22  651  127  279  163  560 2350

Overall Statistics

               Accuracy : 0.5116
                 95% CI : (0.5061, 0.5172)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4574

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.35214  0.62334  0.39228  0.76967  0.49271  0.52272
Specificity           0.93103  0.96442  0.96161  0.97299  0.94202  0.94787
Pos Pred Value        0.36426  0.66341  0.53227  0.74786  0.48823  0.52877
Neg Pred Value        0.92756  0.95791  0.93425  0.97595  0.94299  0.94666
Prevalence            0.10090  0.10113  0.10020  0.09428  0.10093  0.10065
Detection Rate        0.03553  0.06304  0.03931  0.07257  0.04973  0.05261
Detection Prevalence  0.09754  0.09502  0.07385  0.09703  0.10186  0.09949
Balanced Accuracy     0.64158  0.79388  0.67694  0.87133  0.71736  0.73529
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.52155  0.31234  0.39624   0.74817
Specificity           0.92459  0.96354  0.94574   0.90337
Pos Pred Value        0.43707  0.48777  0.44930   0.46369
Neg Pred Value        0.94509  0.92650  0.93343   0.96981
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.05264  0.03125  0.03982   0.07516
Detection Prevalence  0.12044  0.06406  0.08862   0.16208
```

```r
carttrn$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3521395	0.9310283	0.3642623	0.9275640	0.3642623	0.3521395	0.3580983	0.10090188	0.03553153	0.09754381	0.6415839
Class: 2	0.6233397	0.9644204	0.6634130	0.9579107	0.6634130	0.6233397	0.6427523	0.10112575	0.06303569	0.09501727	0.7938800
Class: 3	0.3922758	0.9616136	0.5322650	0.9342519	0.5322650	0.3922758	0.4516722	0.10019829	0.03930536	0.07384547	0.6769447
Class: 4	0.7696744	0.9729873	0.7478576	0.9759510	0.7478576	0.7696744	0.7586092	0.09428169	0.07256620	0.09703211	0.8713308
Class: 5	0.4927123	0.9420176	0.4882261	0.9429904	0.4882261	0.4927123	0.4904589	0.10093386	0.04973135	0.10186133	0.7173650
Class: 6	0.5227201	0.9478681	0.5287689	0.9466562	0.5287689	0.5227201	0.5257271	0.10064603	0.05260970	0.09949469	0.7352941
Class: 7	0.5215463	0.9245874	0.4370685	0.9450949	0.4370685	0.5215463	0.4755851	0.10093386	0.05264168	0.12044263	0.7230668
Class: 8	0.3123402	0.9635394	0.4877683	0.9264992	0.4877683	0.3123402	0.3808225	0.10003838	0.03124600	0.06405910	0.6379398
Class: 9	0.3962444	0.9457442	0.4492963	0.9334316	0.4492963	0.3962444	0.4211060	0.10048612	0.03981707	0.08862095	0.6709943
Class: 10	0.7481694	0.9033669	0.4636938	0.9698092	0.4636938	0.7481694	0.5725423	0.10045414	0.07515671	0.16208264	0.8257681

```
carttst = confusionMatrix(cart_test, as.factor(test$music_genre))
carttst
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   480   40   89   21  230  113   73  146   36  111
        2    22  836  192   71   31  103    0   32    1    4
        3    28  165  453   36   18  133    0  109    0    4
        4     7  150   40 1012   12   20    1  121    0    4
        5   168   64  175   17  645   61   11  136    4   35
        6    73   75  155   42   22  683    0  263    0    5
        7   147    0   17    1   33   38  677   35  549   66
        8    56   13  153   55   72  109   14  408    3   10
        9    53    1   10    0   30   23  443   24  519   90
        10  318   11   59    9  259   66  134   66  234 1017

Overall Statistics

               Accuracy : 0.5022
                 95% CI : (0.4937, 0.5107)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4469

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.35503  0.61697  0.33730  0.80063  0.47707  0.50630
Specificity           0.92870  0.96214  0.95911  0.97075  0.94431  0.94731
Pos Pred Value        0.35848  0.64706  0.47886  0.74031  0.49012  0.51821
Neg Pred Value        0.92770  0.95714  0.92854  0.97906  0.94149  0.94488
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03582  0.06239  0.03381  0.07552  0.04813  0.05097
Detection Prevalence  0.09993  0.09642  0.07060  0.10201  0.09821  0.09836
Balanced Accuracy     0.64187  0.78956  0.64821  0.88569  0.71069  0.72680
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.50037  0.30448  0.38559    0.7556
Specificity           0.92645  0.95978  0.94408    0.9041
Pos Pred Value        0.43314  0.45689  0.43504    0.4680
Neg Pred Value        0.94289  0.92548  0.93225    0.9707
Prevalence            0.10097  0.10000  0.10045    0.1004
Detection Rate        0.05052  0.03045  0.03873    0.0759
Detection Prevalence  0.11664  0.06664  0.08903    0.1622
Balanced Accuracy     0.71341  0.63213  0.66484    0.8298
```

```
carttst$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3550296	0.9287019	0.3584765	0.9277009	0.3584765	0.3550296	0.3567447	0.10089552	0.03582090	0.09992537	0.6418657
Class: 2	0.6169742	0.9621420	0.6470588	0.9571358	0.6470588	0.6169742	0.6316585	0.10111940	0.06238806	0.09641791	0.7895581
Class: 3	0.3373045	0.9591109	0.4788584	0.9285370	0.4788584	0.3373045	0.3958060	0.10022388	0.03380597	0.07059701	0.6482077
Class: 4	0.8006329	0.9707482	0.7403072	0.9790576	0.7403072	0.8006329	0.7692892	0.09432836	0.07552239	0.10201493	0.8856905
Class: 5	0.4770710	0.9443061	0.4901216	0.9414929	0.4901216	0.4770710	0.4835082	0.10089552	0.04813433	0.09820896	0.7106886
Class: 6	0.5063010	0.9473073	0.5182094	0.9448767	0.5182094	0.5063010	0.5121860	0.10067164	0.05097015	0.09835821	0.7268041
Class: 7	0.5003695	0.9264547	0.4331414	0.9428909	0.4331414	0.5003695	0.4643347	0.10097015	0.05052239	0.11664179	0.7134121
Class: 8	0.3044776	0.9597844	0.4568869	0.9254817	0.4568869	0.3044776	0.3654277	0.10000000	0.03044776	0.06664179	0.6321310
Class: 9	0.3855869	0.9440850	0.4350377	0.9322520	0.4350377	0.3855869	0.4088224	0.10044776	0.03873134	0.08902985	0.6648359
Class: 10	0.7555721	0.9040982	0.4680166	0.9706956	0.4680166	0.7555721	0.5780051	0.10044776	0.07589552	0.16216418	0.8298351

Summary of insights from training and testing

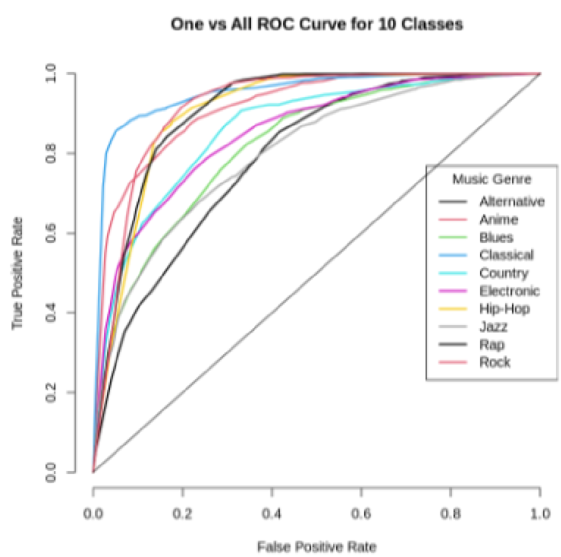
- The training accuracy is 0.51, and the testing accuracy is 0.50.
- Sensitivity/ recall is highest for the classical class (label 4) with a value 0.80 followed by the rock class (label 10) with a value of 0.75 for test, and 0.76 for classical and 0.74 for rock for training which implies the model could predict the classical and rock genre well.
- The specificity is high ~0.93 to 0.97 for all classes which implies that instances not belonging to a certain class were identified as not belonging to that class correctly.
- Precision is highest for classical class with a value of 0.74 and lowest for class alternative with a value of 0.35 for testing which implies that the classical genre is predicted well by the model and the alternative genre is not predicted that well.
- F1 score (harmonic mean of precision and recall) is highest for the classical genre and lowest for the alternative genre for the train and test which means the classifier works well for classical genre.

```
[ ] cart_prob_test = predict(object = tree, newdata = test, type = "prob")
```

```
[ ] roc_cart = multiclass.roc(test$music_genre, cart_prob_test)
    auc(roc_cart)
```

0.872593031612828

```
multiclass_roc_plot(test, cart_prob_test)
```




```
[ ] var(as.numeric(cart_test), as.numeric(test$music_genre))
    bias(as.numeric(cart_test), as.numeric(test$music_genre))
```

4.36539453267286
0.261417910447761

Summary of insights

- The area under the curve is 0.87 (close to 1.0, the baseline is 0.5) which implies the model's predictions are pretty decent.
- From the multiclass roc plot, we can see that the model predicts classical, anime and hip hop better than alternative, blues and jazz.
- The model has a slightly higher variance and a low bias value. Ideally, a low-bias low variance model is best but in reality, it is hard to achieve because of variance and bias trade-off.
- The model has a kappa statistic of 0.44 which means its a decent model compared to random chance.
- The decision tree models tend to overfit easily, after tuning the hyperparameters this was found to be the model that is least overfit. For other values of maxdepth, CP etc. the train accuracy was high and test accuracy was comparatively low.
- The predictions are not very accurate and from the confusion matrix, we can see that there is some misclassification.
- Decision trees are quite insatiable and small changes in data can give drastically different results. They are computationally bit more expensive.
- The decision tree model trained on train dataset with 12 features gave the best accuracy (refer code for implementation).

Taking Principal Components

The model was also trained with the principal components just to check if it has any difference in performance. The performance of decision tree trained on PC performed more poorly as compared to the one trained on feature dataset. The test accuracy for the model trained on PCs was 0.43 and the misclassification error is also higher as we can see from the confusion matrix.

From both the logistic regression model and the decision tree model its evident that the model run of selected important features outperforms the model based on the principal components. This might be because the principal component requires 11 components to capture most of the data and just considering the first few PCs do not suffice. So the performance is similar to the model where you consider certain number of features.

```
[ ] #checking perfomance of principal components as well

    tree_pca = rpart(music_genre~., data = train_pca, method = "class",cp=0.001)
```

```
[ ] cart_test_pca = predict(object = tree_pca, newdata = test_pca, type = "class")
```

```
confusionMatrix(cart_test_pca, as.factor(test_pca$music_genre))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4   5   6   7   8   9  10
        1  267  38  52  21  81  88  27  36  43 124
        2   35 692 156  55  32 115   2  34   0   6
        3   84 240 530  90 164 109  32 196  15  24
        4    0 169  15 994   1   5   0 118   0   5
        5  231  61 249   9 545 121  57 208  42 116
        6   74  85  74  23  41 580  14 199  20  31
        7  190   4  29   0  64 100 778  89 748 104
        8   40  41 118  58  56 109  23 335  11  49
        9   17   0   1   0  18  15 208   9 208  15
        10 414  25 119  14 350 107 212 116 259 872

Overall Statistics

               Accuracy : 0.4329
                 95% CI : (0.4245, 0.4414)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3699

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.19749  0.51070  0.39464  0.78639  0.40311  0.42995
Specificity           0.95767  0.96389  0.92088  0.97421  0.90920  0.95345
Pos Pred Value        0.34363  0.61402  0.35714  0.76052  0.33252  0.50833
Neg Pred Value        0.91405  0.94598  0.93177  0.97767  0.93138  0.93727
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.01993  0.05164  0.03955  0.07418  0.04067  0.04328
Detection Prevalence  0.05799  0.08410  0.11075  0.09754  0.12231  0.08515
Balanced Accuracy     0.57758  0.73729  0.65776  0.88030  0.65615  0.69170
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.57502  0.25000  0.15453   0.64785
Specificity           0.88977  0.95813  0.97652   0.86594
Pos Pred Value        0.36942  0.39881  0.42363   0.35048
Neg Pred Value        0.94909  0.91998  0.91184   0.95656
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05806  0.02500  0.01552   0.06507
Detection Prevalence  0.15716  0.06269  0.03664   0.18567
Balanced Accuracy     0.73239  0.60406  0.56553   0.75689
```

```
[39] roc_cart_pca = multiclass.roc(test_pca$music_genre, cart_prob_pca)
     auc(roc_cart_pca)
```

0.840111745835516

```
[40] var(as.numeric(cart_test_pca), as.numeric(test_pca$music_genre))
     bias(as.numeric(cart_test_pca), as.numeric(test_pca$music_genre))
```

3.36620599888831
0.354253731343284

K Nearest Neighbors

K nearest neighbours is a classification algorithm for estimating if a data point will become part of one class or another based on what group the data points nearest to it belong to. It is a lazy learning algorithm as it does not perform any training on the data and does the computation only during test. It does not make any assumptions about the parameters or features of the dataset and it classifies a data point by looking at its nearest neighbours.

Taking Features based on feature importance

The K value determines the number of neighbours it looks at to determine the class of data point. Lower values of k can have high variance, but low bias and can overfit the data, and larger values of k may lead to high bias and lower variance and can underfit the data. Euclidean distance metric is commonly used to find the distance between the neighbours, although there are other distance metrics such as hamming and manhattan. The KNN model has been executed multiple times for dataset of 12 features and the model test error is plotted. From this plot, we can see that the error is minimum for k value of 3.

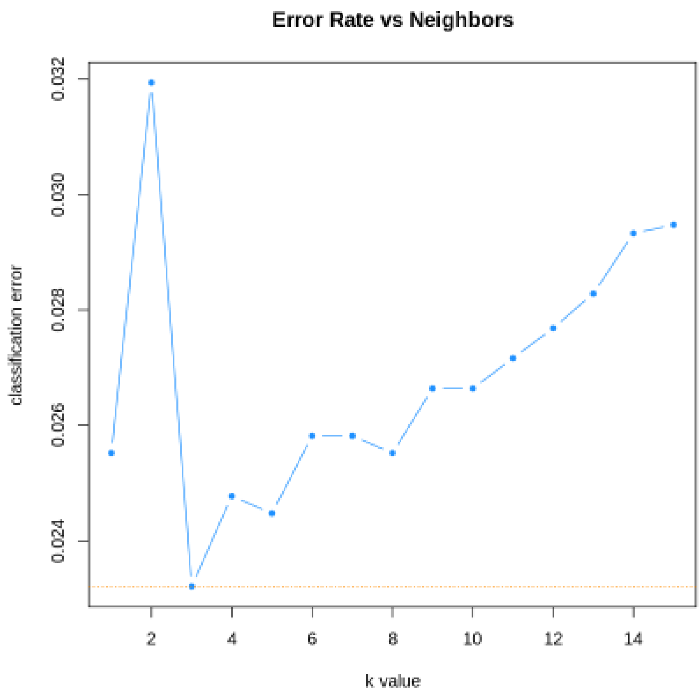
```
[ ] calc_class_err = function(actual, predicted) {
      mean(actual != predicted)
    }
```

```
[ ] set.seed(42)
    k_to_try = 1:15
    err_k = rep(x = 0, times = length(k_to_try))

    for (i in seq_along(k_to_try)) {
      pred = knn(train = train_data,
                 test  = test_data,
                 cl    = train_data$music_genre,
                 k     = k_to_try[i])
      err_k[i] = calc_class_err(test_data$music_genre, pred)
    }
```

```
[ ] err_k
```

0.0255223880597015 · 0.0319402985074627 · 0.0232089552238806 · 0.0247761194029851 · 0.0244776119402985 · 0.0282835820895522 · 0.0293283582089552 · 0.0294776119402985

```
# plot error vs choice of k

plot(err_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
xlab = "k value", ylab = "classification error",main = "Error Rate vs Neighbors")
abline(h = min(err_k), col = "darkorange", lty = 3)
```




```
[ ] which(err_k == min(err_k))
```

3

Summary of insights from training and testing

- The confusion matrix given below shows that most of the data points have been accurately classified. The accuracy of the model on test data is 0.97.
- Sensitivity/ recall is high for the alternative(label 1) with a value 0.99 followed by the country (label 5) and rock (label 10) with value 0.98 and all other classes follow closely which implies the model can predict the genres really well.
- The specificity is high ~0.99 for all classes which implies that instances not belonging to a certain class were identified as not belonging to that class correctly.
- Precision is highest for alternative class with a value of 0.99 and lowest for class blues with a value of 0.93. Since the gap is minimal one can say all genres are predicted accurately.
- F1 score (mean of precision and recall) is highest for the alternative genre.

- Balanced accuracy is high ~0.97 to 0.99 for all genres which implies all genres are classified well.

```
[ ]  knn_model = knn(train = train_data, test  = test_data, cl = train_data$music_genre, k = 3, prob=TRUE)
```

```
cmknn = confusionMatrix(knn_model, as.factor(test_data$music_genre))
cmknn
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1339   14    2    0    0    0    0    0    0    0
        2    12 1319   28    0    0    0    0    0    0    0
        3     1   21 1286   27    0    0    0    0    0    0
        4     0    1   27 1231    6    1    0    0    0    0
        5     0    0    0    6 1337   27    0    0    0    0
        6     0    0    0    0    9 1298    7    0    0    0
        7     0    0    0    0    0   21 1322   23    0    0
        8     0    0    0    0    0    2   22 1300    5    1
        9     0    0    0    0    0    0    2   17 1323   14
        10    0    0    0    0    0    0    0    0   18 1331

Overall Statistics

               Accuracy : 0.9766
                 95% CI : (0.9739, 0.9791)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.974

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.99038  0.97343  0.95756  0.97389  0.98891  0.96219
Specificity           0.99867  0.99668  0.99594  0.99712  0.99726  0.99867
Pos Pred Value        0.98819  0.97057  0.96330  0.97235  0.97591  0.98782
Neg Pred Value        0.99892  0.99701  0.99528  0.99728  0.99875  0.99578
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.09993  0.09843  0.09597  0.09187  0.09978  0.09687
Detection Prevalence  0.10112  0.10142  0.09963  0.09448  0.10224  0.09806
Balanced Accuracy     0.99453  0.98506  0.97675  0.98550  0.99308  0.98043
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.97709  0.97015  0.98291   0.98886
Specificity           0.99635  0.99751  0.99726   0.99851
Pos Pred Value        0.96779  0.97744  0.97566   0.98666
Neg Pred Value        0.99742  0.99669  0.99809   0.99876
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.09866  0.09701  0.09873   0.09933
Detection Prevalence  0.10194  0.09925  0.10119   0.10067
Balanced Accuracy     0.98672  0.98383  0.99009   0.99368
```

```
[ ] cmknn$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.9903846	0.9986720	0.9881919	0.9989207	0.9881919	0.9903846	0.9892870	0.10089552	0.09992537	0.10111940	0.9945283
Class: 2	0.9734317	0.9966791	0.9705666	0.9970102	0.9705666	0.9734317	0.9719971	0.10111940	0.09843284	0.10141791	0.9850554
Class: 3	0.9575577	0.9959360	0.9632959	0.9952756	0.9632959	0.9575577	0.9604182	0.10022388	0.09597015	0.09962687	0.9767468
Class: 4	0.9738924	0.9971160	0.9723539	0.9972804	0.9723539	0.9738924	0.9731225	0.09432836	0.09186567	0.09447761	0.9855042
Class: 5	0.9889053	0.9972610	0.9759124	0.9987531	0.9759124	0.9889053	0.9823659	0.10089552	0.09977612	0.10223881	0.9930831
Class: 6	0.9621942	0.9986723	0.9878234	0.9957802	0.9878234	0.9621942	0.9748404	0.10067164	0.09686567	0.09805970	0.9804333
Class: 7	0.9770880	0.9963476	0.9677892	0.9974240	0.9677892	0.9770880	0.9724163	0.10097015	0.09865672	0.10194030	0.9867178
Class: 8	0.9701493	0.9975124	0.9774436	0.9966860	0.9774436	0.9701493	0.9737828	0.10000000	0.09701493	0.09925373	0.9838308
Class: 9	0.9829123	0.9972623	0.9756637	0.9980903	0.9756637	0.9829123	0.9792746	0.10044776	0.09873134	0.10119403	0.9900873
Class: 10	0.9888559	0.9985067	0.9866568	0.9987553	0.9866568	0.9888559	0.9877551	0.10044776	0.09932836	0.10067164	0.9936813

```
prob = attr(knn_model, "prob")
```

```
[ ] var(as.numeric(knn_model), as.numeric(test_data$music_genre))
    bias(as.numeric(knn_model), as.numeric(test_data$music_genre))
```

8.2832036251313
0.000447761194029851

Summary of insights

- There is no AUC value for this model as in K-NN, the classification decision is usually taken according to the majority vote, and not according to some threshold like other algorithms. So there is no parameter to base a ROC curve on.
- The model has a higher variance and a low bias value, which was expected due to the lower K value. Ideally, a low-bias low variance model is best but in reality, it is hard to achieve because of variance and bias trade-off.
- The model has a kappa statistic of 0.97 which means its a really very good model compared to random chance.
- From the confusion matrix, we can see the misclassification is less and the model accuracy is also high.
- The K-NN model does not scale well with more data and performs poorly when there are more dimensions. In such situations, Principal component analysis or other dimensionality reduction techniques must be used.

Taking less number of features

One of the drawbacks of KNN was it does not perform well with too many features so the KNN model was retrained taking the top 7 important features with the hope to see maybe some improvement in performance.

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1347    0    1    0    0    0    0    0    0    0
        2     5 1352    4    0    0    0    0    0    0    0
        3     0    3 1333   10    0    0    0    0    0    0
        4     0    0    5 1253    1    0    0    0    0    0
        5     0    0    0    1 1345    4    0    0    0    0
        6     0    0    0    0    6 1338    1    0    0    0
        7     0    0    0    0    0    7 1344    6    0    0
        8     0    0    0    0    0    0    8 1330    1    1
        9     0    0    0    0    0    0    0    4 1344    5
        10    0    0    0    0    0    0    0    0    1 1340

Overall Statistics

               Accuracy : 0.9945
                 95% CI : (0.9931, 0.9957)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9939

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            0.9963   0.9978  0.99255  0.99130   0.9948  0.99185
Specificity            0.9999   0.9993  0.99892  0.99951   0.9996  0.99942
Pos Pred Value         0.9993   0.9934  0.99034  0.99523   0.9963  0.99480
Neg Pred Value         0.9996   0.9998  0.99917  0.99909   0.9994  0.99909
Prevalence             0.1009   0.1011  0.10022  0.09433   0.1009  0.10067
Detection Rate         0.1005   0.1009  0.09948  0.09351   0.1004  0.09985
Detection Prevalence   0.1006   0.1016  0.10045  0.09396   0.1007  0.10037
Balanced Accuracy      0.9981   0.9985  0.99574  0.99540   0.9972  0.99563
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity            0.9933  0.99254   0.9985    0.9955
Specificity            0.9989  0.99917   0.9993    0.9999
Pos Pred Value         0.9904  0.99254   0.9933    0.9993
Neg Pred Value         0.9993  0.99917   0.9998    0.9995
Prevalence             0.1010  0.10000   0.1004    0.1004
Detection Rate         0.1003  0.09925   0.1003    0.1000
Detection Prevalence   0.1013  0.10000   0.1010    0.1001
Balanced Accuracy      0.9961  0.99585   0.9989    0.9977
```

```
[40] var(as.numeric(knn_model2), as.numeric(testData$music_genre))
     bias(as.numeric(knn_model2), as.numeric(testData$music_genre))
```

```
8.28492090632738
0.000298507462686567
```

The KNN model trained on the principal component data has an accuracy of 0.98 and a kappa value of 0.98 which is higher than the model trained on 12 selected features

Taking Principal Components

One of the drawbacks of KNN was it does not perform well with too many features and dimensionality reduction techniques may lead to better results. The principal component recast data was then used to train the KNN model again with the hope to see some improvement in performance.

```
[30] cmknn_pca= confusionMatrix(knn_model_pca, as.factor(test_pca$music_genre))
     cmknn_pca
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1342   12    1    0    0    0    0    0    0    0
        2     9 1318   17    0    0    0    0    0    0    0
        3     1   24 1310   20    0    0    0    0    0    0
        4     0    1   15 1235    4    2    0    0    0    0
        5     0    0    0    9 1340   21    0    0    0    0
        6     0    0    0    0    8 1302    4    0    0    0
        7     0    0    0    0    0   24 1333   19    0    0
        8     0    0    0    0    0    0   14 1312    2    0
        9     0    0    0    0    0    0    2    9 1335   11
        10    0    0    0    0    0    0    0    0    9 1335

Overall Statistics

               Accuracy : 0.9822
                 95% CI : (0.9799, 0.9844)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9803

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            0.9926  0.97269  0.97543  0.97706   0.9911  0.96516
Specificity            0.9989  0.99784  0.99627  0.99819   0.9975  0.99900
Pos Pred Value         0.9904  0.98065  0.96679  0.98250   0.9781  0.99087
Neg Pred Value         0.9992  0.99693  0.99726  0.99761   0.9990  0.99611
Prevalence             0.1009  0.10112  0.10022  0.09433   0.1009  0.10067
Detection Rate         0.1001  0.09836  0.09776  0.09216   0.1000  0.09716
Detection Prevalence   0.1011  0.10030  0.10112  0.09381   0.1022  0.09806
Balanced Accuracy      0.9958  0.98527  0.98585  0.98762   0.9943  0.98208
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.98522  0.97910  0.99183   0.99183
Specificity           0.99643  0.99867  0.99817   0.99925
Pos Pred Value        0.96875  0.98795  0.98379   0.99330
Neg Pred Value        0.99834  0.99768  0.99909   0.99909
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.09948  0.09791  0.09963   0.09963
Detection Prevalence  0.10269  0.09910  0.10127   0.10030
Balanced Accuracy     0.99082  0.98889  0.99500   0.99554
```

```
cmknn_pca$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.9926036	0.9989210	0.9904059	0.9991698	0.9904059	0.9926036	0.9915035	0.10089552	0.10014925	0.10111940	0.9957623
Class: 2	0.9726937	0.9978414	0.9806548	0.9969310	0.9806548	0.9726937	0.9766580	0.10111940	0.09835821	0.10029851	0.9852676
Class: 3	0.9754281	0.9962677	0.9667897	0.9972603	0.9667897	0.9754281	0.9710897	0.10022388	0.09776119	0.10111940	0.9858479
Class: 4	0.9770570	0.9981872	0.9824980	0.9976118	0.9824980	0.9770570	0.9797699	0.09432836	0.09216418	0.09380597	0.9876221
Class: 5	0.9911243	0.9975100	0.9781022	0.9990025	0.9781022	0.9911243	0.9845702	0.10089552	0.10000000	0.10223881	0.9943171
Class: 6	0.9651594	0.9990042	0.9908676	0.9961112	0.9908676	0.9651594	0.9778445	0.10067164	0.09716418	0.09805970	0.9820818
Class: 7	0.9852180	0.9964306	0.9687500	0.9983367	0.9687500	0.9852180	0.9769146	0.10097015	0.09947761	0.10268657	0.9908243
Class: 8	0.9791045	0.9986733	0.9879518	0.9976806	0.9879518	0.9791045	0.9835082	0.10000000	0.09791045	0.09910448	0.9888889
Class: 9	0.9918276	0.9981749	0.9837878	0.9990866	0.9837878	0.9918276	0.9877913	0.10044776	0.09962687	0.10126866	0.9950013
Class: 10	0.9918276	0.9992534	0.9933036	0.9990876	0.9933036	0.9918276	0.9925651	0.10044776	0.09962687	0.10029851	0.9955405

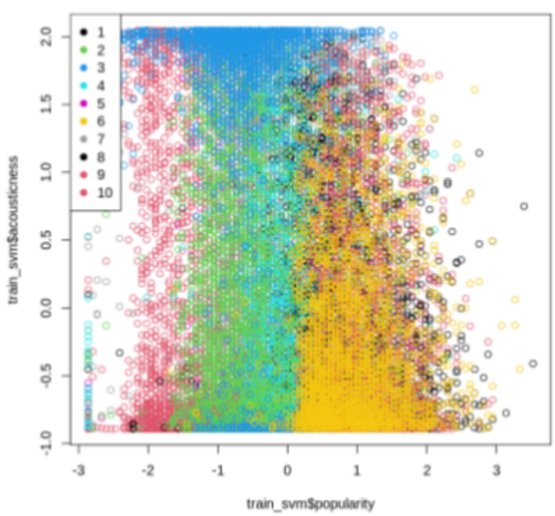
```
[32] var(as.numeric(knn_model_pca), as.numeric(test_pca$music_genre))
     bias(as.numeric(knn_model_pca), as.numeric(test_pca$music_genre))

8.2804769680963
0.000970149253731343
```

The KNN model trained on the principal component data has an accuracy of 0.98 and a kappa value of 0.98 which is higher than the model trained on 12 selected features but lesser than 7 features. The model has similar variance and bias values as the model trained on selected features.

Support Vector Machines

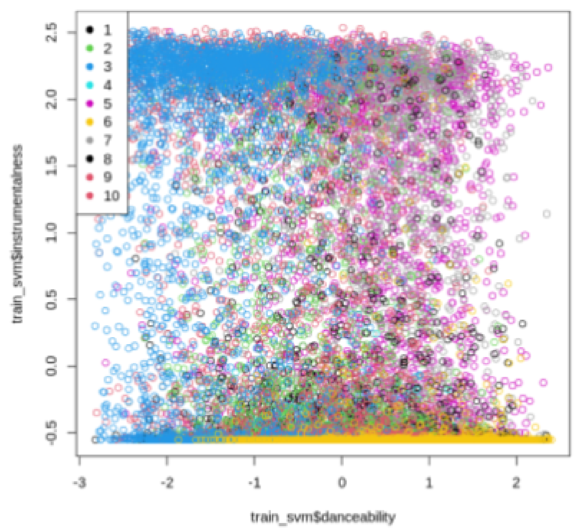
Support vector machine plots input data as points in an n-dimensional space. The algorithm then attempts to iteratively find a hyperplane that can act as a separator between the different target output classes and primarily used for binary classification tasks. The optimal hyperplane is plane with the largest margin between classes. The support vectors are the data points closest to the separating hyperplane. For multiclass classification, there are three approaches: one vs one approach or one vs all approach or directed acyclic graphs. The e1071 library in R uses one vs one approach, in which k(k-1)/2 binary classifiers are trained where k is the number of target classes.

```r
plot(train_svm$popularity, train_svm$acousticness,
     col = factor(train_svm$music_genre))

# Legend
legend("topleft",
       legend = levels(factor(train_svm$music_genre)),
       pch = 19,
       col = factor(levels(factor(train_svm$music_genre))))
```



The plots above and below just show a scatter plot which plots the data points between two numerical features of the train dataset. These plots were done just to see how much separation exists between the data points. We can see that in the first plot we can make out atleast five classes pretty well, where as in the plot below we can make out around three classes. These are just a 2D plots and its hard to distinguish the classes from them, but the algorithm casts it on a N dimensional space so it can distinguish between the classes better.

```
plot(train_svm$danceability, train_svm$instrumentalness,
     col = factor(train_svm$music_genre))

# Legend
legend("topleft",
       legend = levels(factor(train_svm$music_genre)),
       pch = 19,
       col = factor(levels(factor(train_svm$music_genre))))
```



Taking Features based on feature importance

Since the dataset with 12 important features worked the best for all the models so far the same 12 features have been used to train the SVM model.

```
svm_model = svm(music_genre~.,data=train_svm, type='C-classification',probability = TRUE)
```

```
summary(svm_model)
```

```
Call:
svm(formula = music_genre ~ ., data = train_svm, type = "C-classification",
    probability = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  25345

 ( 2255 1993 2723 3037 2694 3074 2707 2471 1384 3007 )


Number of Classes:  10

Levels:
 1 2 3 4 5 6 7 8 9 10
```

```
[ ] pred_svm_train = predict(svm_model, train_svm)
```

```
confusionMatrix(pred_svm_train,as.factor(train_svm$music_genre))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1188   92  144   75  199  206  106   96  121  264
        2    18 2257  304  111   59  154    0   56    0    9
        3    48  241 1719   74  184  150    4  306    2    9
        4     4  272   41 2478    4   19    0  211    0    8
        5   487   94  319   14 1824  124   45  159   45  197
        6   153  136  139   53   92 2011   37  374   24   27
        7   328    2    7    0   75   87 1910   84 1330   76
        8   185   52  306  135  162  278   23 1729   21   85
        9   115    1    5    0   24   44  859   12 1190  140
        10  629   15  149    8  533   74  172  101  409 2326

Overall Statistics

               Accuracy : 0.5959
                 95% CI : (0.5904, 0.6013)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.551

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.37655  0.71379  0.54868  0.84057  0.57795  0.63902
Specificity           0.95365  0.97470  0.96382  0.98026  0.94721  0.96319
Pos Pred Value        0.47692  0.76044  0.62806  0.81594  0.55139  0.66021
Neg Pred Value        0.93165  0.96802  0.95044  0.98335  0.95236  0.95975
Prevalence            0.10090  0.10113  0.10020  0.09428  0.10093  0.10065
Detection Rate        0.03799  0.07218  0.05498  0.07925  0.05833  0.06431
Detection Prevalence  0.07967  0.09492  0.08753  0.09713  0.10580  0.09742
Balanced Accuracy     0.66510  0.84425  0.75625  0.91042  0.76258  0.80111
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.60520  0.55275  0.37874   0.74053
Specificity           0.92925  0.95569  0.95733   0.92569
Pos Pred Value        0.48987  0.58098  0.49791   0.52672
Neg Pred Value        0.95447  0.95055  0.93241   0.96965
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.06108  0.05530  0.03806   0.07439
Detection Prevalence  0.12470  0.09518  0.07644   0.14123
Balanced Accuracy     0.76722  0.75422  0.66804   0.83311
```

```
[33] cmtsvm$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3765452	0.9536513	0.4769169	0.9316468	0.4769169	0.3765452	0.4208289	0.10090188	0.03799412	0.07966611	0.6650983
Class: 2	0.7137887	0.9747029	0.7604447	0.9680212	0.7604447	0.7137887	0.7363785	0.10112575	0.07218242	0.09492133	0.8442458
Class: 3	0.5486754	0.9638173	0.6280599	0.9504399	0.6280599	0.5486754	0.5856899	0.10019829	0.05497633	0.08753358	0.7562464
Class: 4	0.8405699	0.9802613	0.8159368	0.9833516	0.8159368	0.8405699	0.8280702	0.09428169	0.07925035	0.09712805	0.9104156
Class: 5	0.5779468	0.9472112	0.5513906	0.9523605	0.5513906	0.5779468	0.5643564	0.10093386	0.05833440	0.10579506	0.7625790
Class: 6	0.6390213	0.9631948	0.6602101	0.9597477	0.6602101	0.6390213	0.6494429	0.10064603	0.06431495	0.09741589	0.8011080
Class: 7	0.6051965	0.9292473	0.4898692	0.9544740	0.4898692	0.6051965	0.5414600	0.10093386	0.06108482	0.12469618	0.7672219
Class: 8	0.5527494	0.9556859	0.5809812	0.9505514	0.5809812	0.5527494	0.5665138	0.10003838	0.05529615	0.09517718	0.7542176
Class: 9	0.3787397	0.9573349	0.4979079	0.9324053	0.4979079	0.3787397	0.4302242	0.10048612	0.03805808	0.07643597	0.6680373
Class: 10	0.7405285	0.9256942	0.5267210	0.9696484	0.5267210	0.7405285	0.6155882	0.10045414	0.07438915	0.14123065	0.8331113

```
[ ] pred_svm_test = predict(svm_model,test_svm,probability = TRUE)
```

```
confusionMatrix(pred_svm_test,as.factor(test_svm$music_genre))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   494   46   53   27   95   79   67   40   75  150
        2    13  933  150   49   19   88    0   28    0    4
        3    53  129  700   15   98   57    1  135    0    7
        4     2  126   25 1098    3   11    0  107    0    3
        5   201   27  120    5  778   45   23   88   17  136
        6    89   60   66   21   43  845   23  165    8   16
        7   148    0    7    0   23   30  692   31  592   36
        8    77   26  162   46   76  140   25  695    4   33
        9    40    1    3    0    8   16  432    7  497   58
        10  235    7   57    3  209   38   90   44  153  903

Overall Statistics

               Accuracy : 0.5698
                 95% CI : (0.5613, 0.5782)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.522

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.36538  0.68856  0.52122  0.86867  0.57544  0.62639
Specificity           0.94754  0.97086  0.95895  0.97718  0.94505  0.95926
Pos Pred Value        0.43872  0.72664  0.58577  0.79855  0.54028  0.63249
Neg Pred Value        0.93010  0.96517  0.94732  0.98620  0.95201  0.95822
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03687  0.06963  0.05224  0.08194  0.05806  0.06306
Detection Prevalence  0.08403  0.09582  0.08918  0.10261  0.10746  0.09970
Balanced Accuracy     0.65646  0.82971  0.74008  0.92292  0.76025  0.79282
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.51146  0.51866  0.36924   0.67088
Specificity           0.92803  0.95116  0.95313   0.93065
Pos Pred Value        0.44387  0.54128  0.46798   0.51926
Neg Pred Value        0.94418  0.94676  0.93119   0.96201
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05164  0.05187  0.03709   0.06739
Detection Prevalence  0.11634  0.09582  0.07925   0.12978
Balanced Accuracy     0.71974  0.73491  0.66118   0.80076
```

```
cmtstsvm$byClass
```

A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3676036	0.9473772	0.4394341	0.9303122	0.4394341	0.3676036	0.4003222	0.10089552	0.03708955	0.08440299	0.6574904
Class: 2	0.6892989	0.9709423	0.7274143	0.9652526	0.7274143	0.6892989	0.7078439	0.10111940	0.06970149	0.09582090	0.8301206
Class: 3	0.5197319	0.9595256	0.5885329	0.9471917	0.5885329	0.5197319	0.5519968	0.10022388	0.05208955	0.08850746	0.7396288
Class: 4	0.8686709	0.9770105	0.7973856	0.9861931	0.7973856	0.8686709	0.8315032	0.09432836	0.08194030	0.10276119	0.9228407
Class: 5	0.5798817	0.9447211	0.5406897	0.9524686	0.5406897	0.5798817	0.5596003	0.10089552	0.05850746	0.10820896	0.7623014
Class: 6	0.6256486	0.9591735	0.6317365	0.9581399	0.6317365	0.6256486	0.6286778	0.10067164	0.06298507	0.09970149	0.7924111
Class: 7	0.5121951	0.9276998	0.4430946	0.9442379	0.4430946	0.5121951	0.4751457	0.10097015	0.05171642	0.11671642	0.7199475
Class: 8	0.5164179	0.9514096	0.5414710	0.9465435	0.5414710	0.5164179	0.5286478	0.10000000	0.05164179	0.09537313	0.7339138
Class: 9	0.3670134	0.9529617	0.4655985	0.9309506	0.4655985	0.3670134	0.4104695	0.10044776	0.03686567	0.07917910	0.6599875
Class: 10	0.6693908	0.9309773	0.5199077	0.9618582	0.5199077	0.6693908	0.5852550	0.10044776	0.06723881	0.12932836	0.8001840

Summary of insights from training and testing

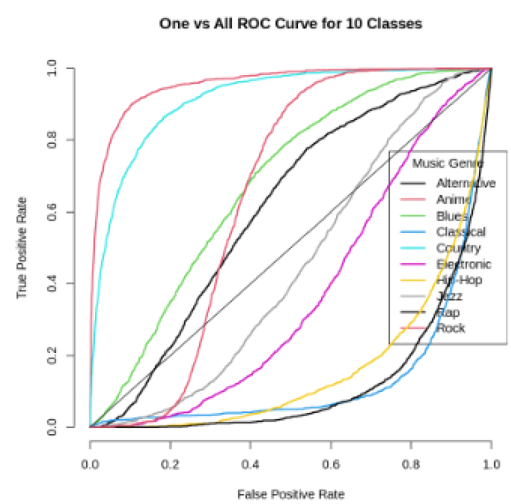
- The SVM model's train accuracy is 0.59 and test accuracy is 0.56.
- Sensitivity/ recall is high for the classical (label 4) with a value of 0.84 followed by the rock (label 10) with a value of 0.74 for train, which implies the model could predict the classical and rock genre well in train data.
- Sensitivity/ recall is high for the classical (label 4) with a value of 0.86 followed by the Anime (label 2) with a value of 0.68 for testing, which implies the model could predict the classical and Anime genre well in test.
- The specificity is high ~0.93 to 0.97 for all classes which implies that instances not belonging to a certain class were identified as not belonging to that class correctly.
- Precision is highest for classical class for both training and test.
- F1 score is highest for the classical genre in boh training and test with values of 0.82 and 0.83 respectively.
- Balanced accuracy is highest for classical (label 4) followed by Anime (label 2) in both train and test which implies that both genres are classified well by the classifier.
- Balanced accuracy is lowest for alternative genre which implies classifier can't classify it that well.

```
[34] test_prob = attr(pred_svm_test, "probabilities")
```

```
[35] roc_svm = multiclass.roc(test_svm$music_genre, test_prob)
     auc(roc_svm)
```

0.922247182844617

```
multiclass_roc_plot(test_svm, test_prob)
```




```
var(as.numeric(pred_svm_test), as.numeric(test_svm$music_genre))
bias(as.numeric(pred_svm_test), as.numeric(test_svm$music_genre))
```

4.41052135768653
0.175746268656716

Summary of insights

- The area under the curve is 0.92.
- From the ROC plot we can see that the false positive rate is high for classical, and hip-hop genres which implies that data points from other genres are getting classified as classical or hip-hop more.
- From the ROC plot we can see that the true positive rate is high for country and anime genres which implies that data points belonging to those were classified correctly.
- The model has a higher variance and a low bias value. Ideally, a low-bias low variance model is best but in reality, it is hard to achieve because of variance and bias trade-off.
- The model has a kappa statistic of 0.52 which means it's a decent model compared to random chance.
- The model was further tuned by changing the parameters such as epsilon, cost, and kernel type. The radial kernel gave the best results.
- The drawback with the one vs one strategy is that the model has to train a large number of classifiers.
- We can see that the accuracy of the model is not very great, this might be because SVM does not perform very well when the dataset has alot of sound and the target classes overlap. This makes it harder to find a good hyperplane.
- The SVM model was also trained with lesser number of features but the performance only degraded with decrease in features.

Taking Principal Components

One of the drawbacks of SVM model is that SVM does not perform very well when the dataset has alot of sound and the target classes overlap and it takes a long time on large datasets. Running SVM with the principal component dataset might improve performance and accuracy of the SVM model because of reduced dimensionality. The resulting accuracy was better than the SVM model trained with 12 features with an accuracy of 0.57, AUC of 0.92 and Kappa of 0.52. But as we can see from the confusion matrix and roc plot the change is minimal as compared to the SVM model trained with 12 features and the insights obtained for the other model still hold for this model as well.

```
[ ] pred_svm_test_pca = predict(svm_model_pca,test_pca,probability = TRUE)
```

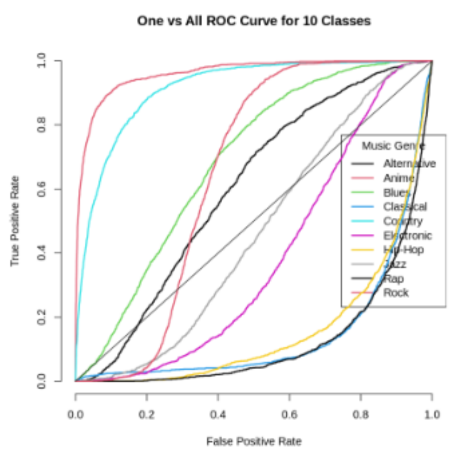
```
confusionMatrix(pred_svm_test_pca,as.factor(test_pca$music_genre))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   488   39   47   32  106   79   59   45   62  137
        2    13  972  153   48   31   80    0   28    0    4
        3    40  114  719   22  100   74    1  147    0   10
        4     4  111   25 1092    4   16    0  104    0    3
        5   203   36  116    5  735   53   21   71   17  129
        6    97   51   70   17   37  835   23  164    8   17
        7   140    0    7    0   27   35  696   33  608   29
        8    77   22  144   44   85  126   22  698    4   34
        9    44    1    3    0    6   15  441    6  493   65
        10  246    9   59    4  221   36   90   44  154  918

Overall Statistics

               Accuracy : 0.5706
                 95% CI : (0.5622, 0.579)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5229

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.36095  0.71734  0.53537  0.86392  0.54364  0.61898
Specificity           0.94970  0.97036  0.95787  0.97800  0.94597  0.95984
Pos Pred Value        0.44607  0.73138  0.58598  0.80353  0.53030  0.63306
Neg Pred Value        0.92979  0.96827  0.94874  0.98572  0.94864  0.95745
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03642  0.07254  0.05366  0.08149  0.05485  0.06231
Detection Prevalence  0.08164  0.09918  0.09157  0.10142  0.10343  0.09843
Balanced Accuracy     0.65532  0.84385  0.74662  0.92096  0.74480  0.78941
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.51441  0.52090  0.36627   0.68202
Specificity           0.92704  0.95373  0.95180   0.92841
Pos Pred Value        0.44190  0.55573  0.45903   0.51544
Neg Pred Value        0.94444  0.94713  0.93080   0.96316
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05194  0.05209  0.03679   0.06851
Detection Prevalence  0.11754  0.09373  0.08015   0.13291
Balanced Accuracy     0.72072  0.73731  0.65904   0.80521
```

```
[ ] roc_svm_pca = multiclass.roc(test_pca$music_genre, test_prob_pca)
    auc(roc_svm_pca)
```

0.923487837613461

```
multiclass_roc_plot(test_pca, test_prob_pca)
```




```
48] var(as.numeric(pred_svm_test_pca), as.numeric(test_pca$music_genre))
    bias(as.numeric(pred_svm_test_pca), as.numeric(test_pca$music_genre))
```

4.45976425061795
0.186268656716418

Comparison of Models

This section compares the model performances for all the models in this report and summarises the results.

Model	Accuracy	Kappa	AUC	Variance	Bias
Logistic Regression - 12 features	0.51	0.46	0.90	4.05	0.11
Logistic Regression - PC	0.51	0.46	0.90	4.06	0.13
Decision Tree - 12 features	0.50	0.44	0.87	4.36	0.26
Decision Tree - PC	0.43	0.36	0.84	3.36	0.35
K-Nearest Neighbors - 12 features	0.97	0.97	-	8.28	0.0004
K-Nearest Neighbors - 7 features	0.99	0.99	-	8.28	0.0002
K-Nearest Neighbors - PC	0.98	0.98	-	8.28	0.0009
Support Vector Machines	0.56	0.52	0.92	4.41	0.17
Support Vector Machines - PC	0.57	0.52	0.92	4.45	0.18

From the table above and the analysis done so far:

- The accuracy and kappa value of K-Nearest Neighbours is the highest as compared to other models.

- The variance of KNN is higher and the bias is lower than the other models. This is because of the bias-variance trade-off which causes a decrease in bias when there is an increase in variance and vice versa.

- The high variance and low bias might indicate overfitting and this occurs for smaller values of k (here k=3). An increase in K may result in an increase in bias (training error) and a decrease in variance (test error). But we have already picked the optimal K value by finding the minimum test error possible.

- The SVM model, seems to be the second best, closely followed by Logistic regression and finally decision tree.

- However both SVM and KNN do not work well with large datasets.

- SVM underperforms as large datasets would have more overlapping of target classes which makes it difficult to find a hyperplane

- KNN will underperform with large datasets as the cost of calculating the distance between a new point and an existing point is expensive and causes degradation of performance.

- Logistic regression does not perform well with complex data with non linear relationships and multicollinearity and most of datasets in real time do not follow these conditions.

- Overall for the music genre dataset of around 45,000 instances and 10 target classes KNN seems to give the best classification results.

References

1. https://www.r-bloggers.com/2021/05/principal-component-analysis-pca-in-r/
2. https://www.statology.org/principal-components-analysis-in-r/
3. https://builtin.com/data-science/step-step-explanation-principal-component-analysis
4. https://www.pluralsight.com/guides/validating-machine-learning-models-with-r
5. https://stats.stackexchange.com/questions/71946/overfitting-a-logistic-regression-model
6. https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124
7. https://cran.r-project.org/web/packages/multiROC/multiROC.pdf
8. https://bookdown.org/sarahwerth2024/CategoricalBook/multinomial-logit-regression-r.html
9. https://www.r-bloggers.com/2020/05/multinomial-logistic-regression-with-r/
10. https://stackoverflow.com/questions/72179298/how-do-i-make-and-plot-roc-curves-in-r-for-multiclass-classification
11. https://cran.r-project.org/web/packages/rpart/rpart.pdf
12. https://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/#:~:text=cp%3A%20Complexity%20Parameter,misclassification%20at%20every%20terminal%20node.
13. https://www.pluralsight.com/guides/explore-r-libraries:-rpart
14. http://www.milbo.org/rpart-plot/prp.pdf
15. https://cran.r-project.org/web/packages/rpart.plot/rpart.plot.pdf
16. https://towardsdatascience.com/beginners-guide-to-k-nearest-neighbors-in-r-from-zero-to-hero-d92cd4074bdb
17. https://rstudio-pubs-static.s3.amazonaws.com/16444_caf85a306d564eb490eebdbaf0072df2.html

18. https://www.edureka.co/blog/knn-algorithm-in-r/

19. https://rpubs.com/ksanto/376630

20. https://www.ibm.com/topics/knn

21. https://stackoverflow.com/questions/36984210/computing-roc-curve-for-k-nn-classifier

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

23. https://rpubs.com/cliex159/865583

24. https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm/

25. https://www.analyticsvidhya.com/blog/2021/06/confusion-matrix-for-multi-class-classification/#:~:text=FP%3A%20The%20False%2Dpositive%20value,are%20calculating%20the%20values%20for.

26. https://www.statology.org/null-residual-deviance/

27. https://www.datatechnotes.com/2019/02/accuracy-metrics-in-classification.html

28. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

29. https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124

30. https://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html#kappa