CS-GY 6923 Machine Learning
Fall 2022

Professor: Dr. Raman Kannan

Ensemble Learning Report
Music Genre Classification

Author: Atmaja Raman
Net ID: ar6871

\

Table of Contents

Introduction

Ensemble learning is used to improve the performance of a model, or reduce the chance of a model with high error. Ensemble models combine the decisions from multiple models to improve the overall performance. These models are known as weak learners. The intuition is that when you combine several weak learners, they can become strong learners.Each weak learner is fitted on the training set and provides predictions obtained. The final prediction result is computed by combining the results from all the weak learners.

There are several ensemble learning methods that can be used to perform multiclass classification, but this report explores the following methods:

- Cross validation
- Random Forest
- Boosting

These classification models are executed for the Music genre classification dataset, which has ten classes in the target variable - music_genre. The model performances are analyzed and compared using the 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 results obtained in the classification and performance analysis task and summarises its results. The multiclass classification was performed to classify the music_genre target column using the following methods:

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

The performance of the models was analyzed using a set of performance metrics and compared. The results are presented in a tabular form below.

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

Loading the dataset

The cleaned dataset from the EDA task was loaded and used for the classification task and the same dataset will be used for the ensemble learning task as well.

```
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.

Cross Validation

Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is done to reduce overfitting or when the dataset is not large enough. In cross-validation, a fixed number of folds or partitions of the data are created, and the model is trained on each fold, and then the results are averaged to find the overall error estimate. There are different types of cross-validations, but this report covers K-Fold cross-validation.

K-fold cross-validation

This approach involves randomly dividing the set of observations into k-folds of similar sizes. The first fold is treated as a validation set, and the model fits the remaining k − 1 folds. This makes sure that every data point from the dataset appears in the training and validation set i.e, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times. This reduces bias as we are using most of the data for fitting, and also reduces variance as most of the data is also being used in the validation set. This method generally results in a less biased model compared to other methods. The K-fold cross-validation has been done for Multinomial logistic regression, K-Nearest Neighbors, Support Vector Machines, and Decision Tree models that were run in the classification task previously, and the results are compared.

1. Multinomial Logistic Regression

```
control_lr = trainControl(method = "repeatedcv", repeats = 10)
set.seed(6871)
```

```
lr_fit = train(as.factor(music_genre)~., data = train_data, method = "multinom", trControl = control_lr)
```

```
# weights:  140 (117 variable)
initial  value 64801.652272
iter  10 value 38090.612111
iter  20 value 37748.131379
iter  30 value 37490.443686
iter  40 value 36549.999669
iter  50 value 36426.463491
iter  60 value 36370.836528
iter  70 value 36258.594575
iter  80 value 36247.764533
iter  90 value 36238.626325
iter 100 value 36231.036180
final  value 36231.036180
stopped after 100 iterations
# weights:  140 (117 variable)
initial  value 64801.652272
iter  10 value 38101.069103
iter  20 value 37759.598598
iter  30 value 37500.978791
iter  40 value 36561.154012
iter  50 value 36438.815361
iter  60 value 36381.691784
iter  70 value 36270.237345
iter  80 value 36259.186498
iter  90 value 36247.971297
iter 100 value 36243.930578
final  value 36243.930578
stopped after 100 iterations
# weights:  140 (117 variable)
initial  value 64801.652272
iter  10 value 38090.622572
iter  20 value 37748.142852
```

```
print(lr_fit)
```

```
Penalized Multinomial Regression

31268 samples
   12 predictor
   10 classes: '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 28143, 28142, 28143, 28140, 28141, 28141, ...
Resampling results across tuning parameters:

  decay  Accuracy   Kappa
  0e+00  0.5176595  0.4640670
  1e-04  0.5176531  0.4640599
  1e-01  0.5175317  0.4639251

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was decay = 0.
```

```
[24] lr_pred_train = predict(lr_fit, newdata = train_data)
```

```
cmtrain = confusionMatrix(lr_pred_train, as.factor(train_data$music_genre))
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
Balanced Accuracy     0.71595  0.67749  0.66445   0.78594
```

```
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

```
[27] lr_pred_test = predict(lr_fit, newdata = test_data)
```

```
cmtest =confusionMatrix(lr_pred_test, as.factor(test_data$music_genre))
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
```

```
[29] cmtest$byClass
```

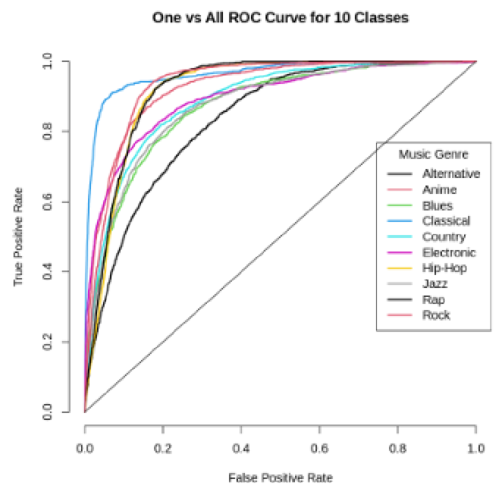
A matrix: 10 × 11 of type dbl

	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

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

0.903220505624288

```
multiclass_roc_plot(test_data, pred_test)
```




```
var(as.numeric(lr_pred_test), as.numeric(test_data$music_genre))

bias(as.numeric(lr_pred_test), as.numeric(test_data$music_genre))
```

4.05280584539056
0.115597014925373

Summary of insights

- The multinomial logistic regression model has undergone 10-fold cross-validation with 10 repeats.
- The train and test accuracy is 0.51 which is similar to the accuracy we got running logistic regression without cross-validation.
- Sensitivity/ recall is highest for target class 4 - classical with a value of 0.81, 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
- Precision is highest for classical (class 4) with a value of 0.71 and lowest for class alternative (class 1) with a value of 0.40 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 test 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 the alternative genre is classified poorly.
- 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.
- Kappa statistic has a value of 0.46 which means it's a decent model compared to random chance.
- The model has a slightly higher bias and a low variance 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 variance and bias values are 4.05 and 0.11 respectively. These values are the same as it was for multinominal logistic regression without cross-validation. It was expected that the variance and bias would decrease but that was not observed in this case.
- 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.

2. K-Nearest Neighbors

KNN

```
set.seed(6871)

train_control = trainControl(method = "repeatedcv", number=10)
```

```
knn_fit = train(as.factor(music_genre)~., data = train_data,
trControl = train_control, method ="knn", metric="Accuracy",
tuneLength = 10)
```

```
print(knn_fit)
```

```
k-Nearest Neighbors

31268 samples
   12 predictor
   10 classes: '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 28143, 28142, 28143, 28140, 28141, 28141, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.4849070  0.4276591
   7  0.4967393  0.4408059
   9  0.5071971  0.4524277
  11  0.5110982  0.4567642
  13  0.5131766  0.4590765
  15  0.5149669  0.4610676
  17  0.5158627  0.4620636
  19  0.5160548  0.4622780
  21  0.5177498  0.4641598
  23  0.5175577  0.4639465

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 21.
```

```
knn_pred_test = predict(knn_fit, newdata = xtst)
```

10 fold cross validation repeating once was performed and from the screenshot above, we can see that during the resampling process, the K parameter was tuned and the most optimal model was modeled when K=21. This K was chosen based on accuracy values on the validation set during the cross-validation process. In the KNN model built without cross-validation previously, we took K=3 as we got the minimal model test error for that value.

```
[33] knn_pred_test = predict(knn_fit, newdata = xtst)
```

```
knn_cm=confusionMatrix(table(as.matrix(ytst),knn_pred_test))
knn_cm
```

```
Confusion Matrix and Statistics

    knn_pred_test
        1    2    3    4    5    6    7    8    9   10
  1   436   16   31    8  250   65  140   58   86  262
  2    49  888  120  146   66   50    1   30    0    5
  3    83  164  591   31  195   77    8  130    7   57
  4    33   42   30 1084    8   18    0   47    1    1
  5   102   20   71    4  844   25   36   49   13  188
  6    91  106   55   14   95  728   61  129   28   42
  7    44    0    4    0   39    8  635   22  498  103
  8    44   27  145  116  131  156   53  603   19   46
  9    70    0    0    0   31   10  562    4  496  173
  10  165    4    9    5  209   23   39   45   57  790

Overall Statistics

               Accuracy : 0.5295
                 95% CI : (0.521, 0.538)
    No Information Rate : 0.1394
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4772

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.39033  0.70087  0.55966  0.76989  0.45182  0.62759
Specificity           0.92543  0.96151  0.93908  0.98499  0.95595  0.94926
Pos Pred Value        0.32249  0.65535  0.44006  0.85759  0.62426  0.53966
Neg Pred Value        0.94348  0.96853  0.96143  0.97330  0.91501  0.96415
Prevalence            0.08336  0.09455  0.07881  0.10507  0.13940  0.08657
Detection Rate        0.03254  0.06627  0.04410  0.08090  0.06299  0.05433
Detection Prevalence  0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Balanced Accuracy     0.65788  0.83119  0.74937  0.87744  0.70388  0.78843
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.41368  0.53984  0.41162   0.47391
Specificity           0.93949  0.94000  0.93030   0.95261
Pos Pred Value        0.46933  0.45000  0.36850   0.58692
Neg Pred Value        0.92529  0.95738  0.94118   0.92724
Prevalence            0.11455  0.08336  0.08993   0.12440
Detection Rate        0.04739  0.04500  0.03701   0.05896
Detection Prevalence  0.10097  0.10000  0.10045   0.10045
Balanced Accuracy     0.67658  0.73992  0.67096   0.71326
```

```
knn_cm$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	0.3903312	0.9254254	0.3224852	0.9434761	0.3224852	0.3903312	0.3531794	0.08335821	0.03253731	0.10089552	0.6578783
Class: 2	0.7008682	0.9615099	0.6553506	0.9685347	0.6553506	0.7008682	0.6773455	0.09455224	0.06626866	0.10111940	0.8311891
Class: 3	0.5596591	0.9390797	0.4400596	0.9614332	0.4400596	0.5596591	0.4927053	0.07880597	0.04410448	0.10022388	0.7493694
Class: 4	0.7698864	0.9849900	0.8575949	0.9733026	0.8575949	0.7698864	0.8113772	0.10507463	0.08089552	0.09432836	0.8774382
Class: 5	0.4518201	0.9559487	0.6242604	0.9150066	0.6242604	0.4518201	0.5242236	0.13940299	0.06298507	0.10089552	0.7038844
Class: 6	0.6275862	0.9492647	0.5396590	0.9641524	0.5396590	0.6275862	0.5803109	0.08656716	0.05432836	0.10067164	0.7884255
Class: 7	0.4136808	0.9394859	0.4693274	0.9252926	0.4693274	0.4136808	0.4397507	0.11455224	0.04738806	0.10097015	0.6765833
Class: 8	0.5398389	0.9399984	0.4500000	0.9573798	0.4500000	0.5398389	0.4908425	0.08335821	0.04500000	0.10000000	0.7399186
Class: 9	0.4116183	0.9302993	0.3684993	0.9411814	0.3684993	0.4116183	0.3888671	0.08992537	0.03701493	0.10044776	0.6709588
Class: 10	0.4739052	0.9526123	0.5869242	0.9272441	0.5869242	0.4739052	0.5243943	0.12440299	0.05895522	0.10044776	0.7132588

```
[36] var(as.numeric(knn_pred_test), as.numeric(test_data$music_genre))

bias(as.numeric(knn_pred_test), as.numeric(test_data$music_genre))
```

4.02085474188874
0.162761194029851

Summary of insights

- The K-Nearest Neighbour model has undergone 10-fold cross-validation with 1 repeat and the optimal K value was found to be 21.
- Sensitivity/ recall is high for the classical (label 4) with a value 0.76 followed by anime (label 2) with value 0.70 which implies the model can predict these two decentlyl.
- 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 classical class with a value of 0.85 and lowest for class alternative with a value of 0.32.
- F1 score (mean of precision and recall) is highest for the classical genre.
- Balanced accuracy is high for classical, anime genres which implies genres are classified well.
- 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's variance has reduced from 8.28 to 4.02 due to cross validation. But the bias of model has increased slightly from 0.0004 to 0.16. With cross validation, we would expect both the variance and bias to reduce.
- The model has a kappa statistic of 0.47 which means its a decent model compared to random chance.
- The accuracy of KNN without cross validation was 0.98 and variance 8.28. It was suspected to be overfitting to the data. After cross validation, the accuracy is down to 0.52 and variance to 4.02. The model is fitting the data better after cross validation.

3. Support Vector Machines

```
set.seed(6871)
ctrl = trainControl(method = "repeatedcv", number=5)
```

```
svm_fit = train(as.factor(music_genre)~., data = train_data, trControl = ctrl, method ="svmRadial", tuneLength = 10)
```

```
print(svm_fit)
```

```
Support Vector Machines with Radial Basis Function Kernel

31268 samples
   12 predictor
   10 classes: '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 1 times)
Summary of sample sizes: 25015, 25014, 25013, 25015, 25015
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.5511379  0.5012620
    0.50  0.5566066  0.5073335
    1.00  0.5619158  0.5132279
    2.00  0.5632588  0.5147176
    4.00  0.5651778  0.5168474
    8.00  0.5615958  0.5128635
   16.00  0.5591334  0.5101210
   32.00  0.5541440  0.5045735
   64.00  0.5462128  0.4957576
  128.00  0.5378976  0.4865167

Tuning parameter 'sigma' was held constant at a value of 0.06843625
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.06843625 and C = 4.
```

```
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	0.3987322	0.9542916	0.4946913	0.9339600	0.4946913	0.3987322	0.4415584	0.10090188	0.04023283	0.08132915	0.6765119
Class: 2	0.7413030	0.9753789	0.7720685	0.9710258	0.7720685	0.7413030	0.7563730	0.10112575	0.07496482	0.09709607	0.8583409
Class: 3	0.5697415	0.9639239	0.6375000	0.9526486	0.6375000	0.5697415	0.6017192	0.10019829	0.05708712	0.08954842	0.7668327
Class: 4	0.8500678	0.9826977	0.8364486	0.9843662	0.8364486	0.8500678	0.8432032	0.09428169	0.08014584	0.09581681	0.9163828
Class: 5	0.5903042	0.9489186	0.5647166	0.9537702	0.5647166	0.5903042	0.5772270	0.10093386	0.05958168	0.10550723	0.7696114
Class: 6	0.6472831	0.9685644	0.6973639	0.9608424	0.6973639	0.6472831	0.6713909	0.10064603	0.06514648	0.09341819	0.8079238
Class: 7	0.6001267	0.9335871	0.5035895	0.9541208	0.5035895	0.6001267	0.5476363	0.10093386	0.06057311	0.12028272	0.7668569
Class: 8	0.5700128	0.9610163	0.6190972	0.9526208	0.6190972	0.5700128	0.5935419	0.10003838	0.05702315	0.09210695	0.7655146
Class: 9	0.4156588	0.9565171	0.5164096	0.9361147	0.5164096	0.4156588	0.4605890	0.10048612	0.04176794	0.08088141	0.6860880
Class: 10	0.7548551	0.9242009	0.5265379	0.9712311	0.5265379	0.7548551	0.6203558	0.10045414	0.07582832	0.14401305	0.8395280

```
pred_svm_train = predict(svm_fit, train_data)
```

```
cmtsvm = confusionMatrix(pred_svm_train,as.factor(train_data$music_genre))
cmtsvm
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1258   87  144   72  199  217  108  109  104  245
        2    21 2344  285  111   59  151    1   57    0    7
        3    47  234 1785   75  165  162    4  312    4   12
        4     6  223   42 2506    4   17    0  191    0    7
        5   476   92  315   14 1863  115   32  160   35  197
        6   131  115  115   46   71 2037   30  330   25   21
        7   300    2    7    0   76   77 1894   75 1244   86
        8   160   45  283  116  152  248   19 1783   13   61
        9   113    1    4    0   24   45  891   11 1306  134
        10  643   19  153    8  543   78  177  100  411 2371

Overall Statistics

               Accuracy : 0.6124
                 95% CI : (0.6069, 0.6178)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5693

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.39873  0.74130  0.56974  0.85007  0.59030  0.64728
Specificity           0.95429  0.97538  0.96392  0.98270  0.94892  0.96856
Pos Pred Value        0.49469  0.77207  0.63750  0.83645  0.56472  0.69736
Neg Pred Value        0.93396  0.97103  0.95265  0.98437  0.95377  0.96084
Prevalence            0.10090  0.10113  0.10020  0.09428  0.10093  0.10065
Detection Rate        0.04023  0.07496  0.05709  0.08015  0.05958  0.06515
Detection Prevalence  0.08133  0.09710  0.08955  0.09582  0.10551  0.09342
Balanced Accuracy     0.67651  0.85834  0.76683  0.91638  0.76961  0.80792
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.60013  0.57001  0.41566   0.75486
Specificity           0.93359  0.96102  0.95652   0.92420
Pos Pred Value        0.50359  0.61910  0.51641   0.52654
Neg Pred Value        0.95412  0.95262  0.93611   0.97123
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.06057  0.05702  0.04177   0.07583
Detection Prevalence  0.12028  0.09211  0.08088   0.14401
Balanced Accuracy     0.76686  0.76551  0.68609   0.83953
```

```
pred_svm_test = predict(svm_fit,test_data,probability = TRUE)
```

```
cmtstsvm = confusionMatrix(pred_svm_test,as.factor(test_data$music_genre))
cmtstsvm
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   494   52   61   33   90   89   54   55   62  122
        2    11  952  143   57   19   88    0   27    0    3
        3    31  136  696   16   92   67    1  135    0    4
        4     2  101   22 1085    1   12    0  102    0    1
        5   201   29  126    6  781   50   19   88    8  107
        6    75   51   65   19   36  831   22  166    5   13
        7   153    0    7    0   28   35  698   28  618   40
        8    67   21  151   43   61  120   23  679    3   27
        9    39    1    4    0    5   15  437    7  477   54
        10  279   12   68    5  239   42   99   53  173  975

Overall Statistics

               Accuracy : 0.5722
                 95% CI : (0.5638, 0.5806)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5247

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.36538  0.70258  0.51824  0.85839  0.57766  0.61601
Specificity           0.94871  0.97111  0.96002  0.98014  0.94738  0.96249
Pos Pred Value        0.44424  0.73231  0.59083  0.81825  0.55194  0.64770
Neg Pred Value        0.93018  0.96669  0.94706  0.98517  0.95236  0.95725
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03687  0.07104  0.05194  0.08097  0.05828  0.06201
Detection Prevalence  0.08299  0.09701  0.08791  0.09896  0.10560  0.09575
Balanced Accuracy     0.65704  0.83685  0.73913  0.91926  0.76252  0.78925
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.51589  0.50672  0.35438   0.72437
Specificity           0.92455  0.95721  0.95338   0.91953
Pos Pred Value        0.43435  0.56820  0.45910   0.50129
Neg Pred Value        0.94446  0.94584  0.92970   0.96761
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05209  0.05067  0.03560   0.07276
Detection Prevalence  0.11993  0.08918  0.07754   0.14515
Balanced Accuracy     0.72022  0.73197  0.65388   0.82195
```

```
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	0.3653846	0.9487052	0.4442446	0.9301758	0.4442446	0.3653846	0.4009740	0.10089552	0.03686567	0.08298507	0.6570449
Class: 2	0.7025830	0.9711083	0.7323077	0.9666942	0.7323077	0.7025830	0.7171375	0.10111940	0.07104478	0.09701493	0.8368457
Class: 3	0.5182427	0.9600232	0.5908319	0.9470627	0.5908319	0.5182427	0.5521618	0.10022388	0.05194030	0.08791045	0.7391330
Class: 4	0.8583861	0.9801417	0.8182504	0.9851748	0.8182504	0.8583861	0.8378378	0.09432836	0.08097015	0.09895522	0.9192639
Class: 5	0.5776627	0.9473772	0.5519435	0.9523571	0.5519435	0.5776627	0.5645103	0.10089552	0.05828358	0.10559701	0.7625199
Class: 6	0.6160119	0.9624927	0.6477007	0.9572501	0.6477007	0.6160119	0.6314590	0.10067164	0.06201493	0.09574627	0.7892523
Class: 7	0.5158906	0.9245455	0.4343497	0.9444586	0.4343497	0.5158906	0.4716216	0.10097015	0.05208955	0.11992537	0.7202181
Class: 8	0.5067164	0.9572139	0.5682008	0.9458419	0.5682008	0.5067164	0.5357002	0.10000000	0.05067164	0.08917910	0.7319652
Class: 9	0.3543834	0.9533765	0.4590953	0.9296982	0.4590953	0.3543834	0.4000000	0.10044776	0.03559701	0.07753731	0.6538799
Class: 10	0.7243685	0.9195288	0.5012853	0.9676124	0.5012853	0.7243685	0.5925251	0.10044776	0.07276119	0.14514925	0.8219486

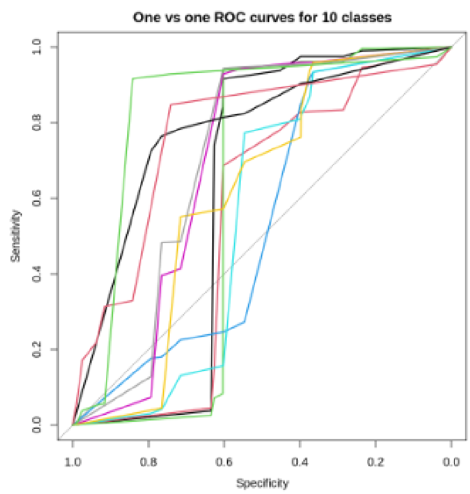
```
roc_svm = multiclass.roc(test_data$music_genre, as.numeric(pred_svm_test))
```

```
auc(roc_svm)
```

0.742422237935271

```
svm_roc_tst=roc_svm$rocs
```

```
plot.roc(svm_roc_tst[[1]], col=1, main="One vs one ROC curves for 10 classes")

for(i in 2:11)
{
  num=paste("1/",as.character(i),sep="")
  lines.roc(svm_roc_tst[[i]],col=i)
}
```




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

4.51433511968481
0.23634328358209

Summary of insights

- The SVM model underwent 5 fold cross validation with 1 repeat and its train accuracy is 0.61 and test accuracy is 0.57.
- During the resampling process, the parameters were tuned and the most optimal model was modeled when sigma=0.06 and C=4. These values were chosen based on accuracy values on the validation set during the cross-validation process.
- Sensitivity/ recall is high for the classical (label 4) with a value of 0.85 followed by the rock (label 10) with a value of 0.75 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 rock (label 10) with a value of 0.72 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 both training and test with values of 0.81 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 rap genre which implies classifier can't classify it that well.
- The area under the curve is 0.74.
- 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. It is comparable to the values of the previous SVM model.
- The model has a kappa statistic of 0.52 which means it's a decent model compared to random chance.
- 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.

4. Decision Trees

```
[ ] set.seed(6871)

    train_ctrl = trainControl(method = "repeatedcv", number = 10, repeats = 2)
```

```
[ ] tune_grid = expand.grid(cp=c(0.001))
```

```
[ ] dt_model = train(as.factor(music_genre)~.,data=train_data, method="rpart", trControl= train_ctrl,tuneGrid = tune_grid)
```

```
dt_model
```

```
CART

31268 samples
   12 predictor
   10 classes: '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 2 times)
Summary of sample sizes: 28143, 28142, 28143, 28140, 28141, 28141, ...
Resampling results:

  Accuracy   Kappa
  0.4978409  0.4420216

Tuning parameter 'cp' was held constant at a value of 0.001
```

```
[ ] cart_train = predict(dt_model, data = train_data)
```

```
carttrn = confusionMatrix(cart_train, as.factor(train_data$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   267    2   24    0   84  103 1159   98  697  158
        8   102   43  321  138  136  243   12  977    8   23
        9   209    2   19    1   67   66 1498   74 1789  220
        10  761   21  134   22  651  127  279  163  560 2350

Overall Statistics

               Accuracy : 0.5135
                 95% CI : (0.5079, 0.519)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4594

 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           0.36724  0.31234  0.56938   0.74817
Specificity           0.94903  0.96354  0.92334   0.90337
Pos Pred Value        0.44715  0.48777  0.45349   0.46369
Neg Pred Value        0.93036  0.92650  0.95048   0.96981
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.03707  0.03125  0.05722   0.07516
Detection Prevalence  0.08290  0.06406  0.12617   0.16208
Balanced Accuracy     0.65813  0.63794  0.74636   0.82577
```

```
[ ] cart_test = predict(object = dt_model, newdata = test)
```

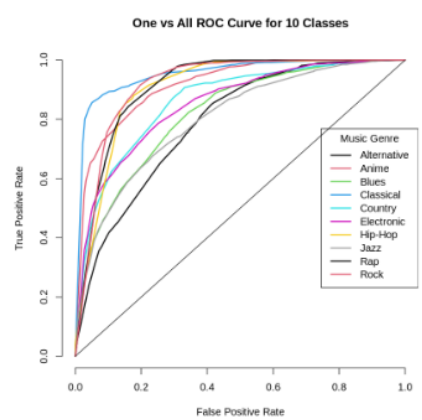
```
carttst = confusionMatrix(cart_test, as.factor(test_data$music_genre))
carttst
```

```
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   123    0   17    1   22   33  476   29  322   55
        8    56   13  153   55   72  109   14  408    3   10
        9    77    1   10    0   41   28  644   30  746  101
        10  318   11   59    9  259   66  134   66  234 1017

Overall Statistics

               Accuracy : 0.5042
                 95% CI : (0.4957, 0.5127)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4491

 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           0.35181  0.30448  0.55423    0.7556
Specificity           0.95003  0.95978  0.92268    0.9041
Pos Pred Value        0.44156  0.45689  0.44458    0.4680
Neg Pred Value        0.92883  0.92548  0.94881    0.9707
Prevalence            0.10097  0.10000  0.10045    0.1004
Detection Rate        0.03552  0.03045  0.05567    0.0759
Detection Prevalence  0.08045  0.06664  0.12522    0.1622
Balanced Accuracy     0.65092  0.63213  0.73846    0.8298
```

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

0.873000599783887

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




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

4.52169144946215
0.333805970149254

Summary of insights

- The decision tree model underwent 10 fold cross-validation with 2 repeats.
- 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.
- The area under the curve is 0.87 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 cross validated model has 4.52 variance and 0.33 bias which is a bit higher than the previous model which had 4.36 and 0.26. This is a bit weird because we would expect cross validation to decrease variance and bias.
- The model has a kappa statistic of 0.44 which means its a decent model compared to random chance.

Random Forest

Random forest is an ensemble of decision tree algorithms. It is an extension of bagging of decision trees and can be used for classification and regression problems. Random forest has a large number of decision trees. Each individual tree in the random forest gives a class prediction and the class with the most votes becomes our model's prediction. It searches for the best feature among a random subset of features when searching for the most important feature while splitting a node.

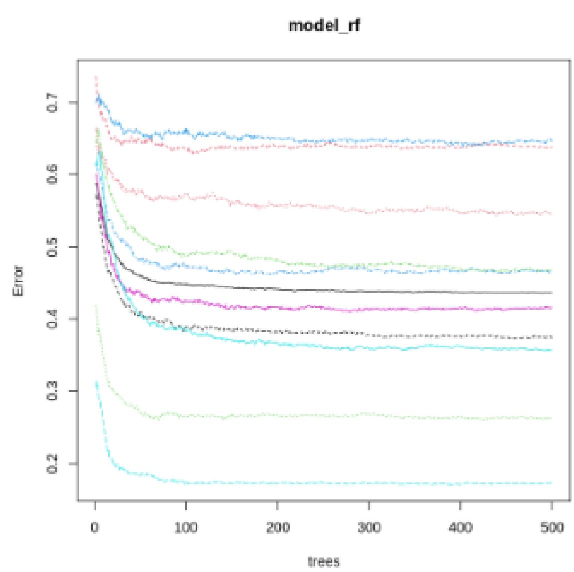
In the line plot below, we can see that error sharply decreases till ntree=50 and continues to decrease till ntree=500 (you can see the slight downward slope)

```
[ ] model_rf = randomForest(as.factor(music_genre) ~ ., ntree = 500, importance = TRUE, data = train_data)
```

```
print(model_rf)
```

```
Call:
 randomForest(formula = as.factor(music_genre) ~ ., data = train_data,      ntree = 500, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 43.66%
Confusion matrix:
      1    2    3    4    5    6    7    8    9   10 class.error
1  1145   25   75    8  381  156  268  206  179  712   0.6370840
2    90 2327  193  239   98  144    2   48    0   21   0.2640734
3   174  273 1676   51  262  192    6  329    4  166   0.4650495
4    72  121   84 2436   13   52    0  158    1   11   0.1736771
5   230   42  159    4 1843   48   49  160   54  567   0.4160330
6   188  115  197   21  101 1971   55  355   38  106   0.3736892
7   137    1    4    0   38   29 1425   23 1362  137   0.5484791
8   112   44  337  211  156  399   69 1663   19  118   0.4683504
9   123    4    1    0   42   14 1539   11 1108  300   0.6473584
10  406   14   48    8  275   24  108   61  176 2021   0.3565743
```

```
[ ]  plot(model_rf)
```



```
[ ] predtrn_rf = predict(model_rf, train_data)
```

```
cmtrn = confusionMatrix(predtrn_rf, as.factor(train_data$music_genre))
cmtrn
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  3155    0   23    0    0    0    0    0    0    1
        2     0 3162    0    0    0    0    0    0    0    0
        3     0    0 3109    0    0    0    0    0    0    1
        4     0    0    0 2948    0    0    0    2    0    0
        5     0    0    0    0 3136    0    0    5    6   53
        6     0    0    0    0    0 3141    2   42    1    0
        7     0    0    0    0    0    1 3087    4  495   13
        8     0    0    0    0    1    5    1 3073    0    0
        9     0    0    0    0    2    0   65    0 2626   21
        10    0    0    1    0   17    0    1    2   14 3052

Overall Statistics

               Accuracy : 0.9751
                 95% CI : (0.9733, 0.9768)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9723

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            1.0000   1.0000  0.99234  1.00000   0.9937   0.9981
Specificity            0.9991   1.0000  0.99996  0.99993   0.9977   0.9984
Pos Pred Value         0.9925   1.0000  0.99968  0.99932   0.9800   0.9859
Neg Pred Value         1.0000   1.0000  0.99915  1.00000   0.9993   0.9998
Prevalence             0.1009   0.1011  0.10020  0.09428   0.1009   0.1006
Detection Rate         0.1009   0.1011  0.09943  0.09428   0.1003   0.1005
Detection Prevalence   0.1017   0.1011  0.09946  0.09435   0.1023   0.1019
Balanced Accuracy      0.9996   1.0000  0.99615  0.99996   0.9957   0.9982
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.97814  0.98242  0.83577   0.97167
Specificity           0.98175  0.99975  0.99687   0.99876
Pos Pred Value        0.85750  0.99773  0.96758   0.98866
Neg Pred Value        0.99751  0.99805  0.98193   0.99684
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.09873  0.09828  0.08398   0.09761
Detection Prevalence  0.11513  0.09850  0.08680   0.09873
Balanced Accuracy     0.97994  0.99108  0.91632   0.98521
```

```
[ ] predtst_rf = predict(model_rf, test_data)
```

```
cmtst = confusionMatrix(predtst_rf, as.factor(test_data$music_genre))
cmtst
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   520   42   66   26   97   73   69   39   63  168
        2    12  995  128   38   17   77    0   19    0    4
        3    36  107  705   25   77   78    0  133    1   28
        4     1   99   29 1104    1   11    0  100    0    2
        5   152   32  100    4  829   30   17   76   15  132
        6    82   44   78   20   12  849    8  179    3   14
        7   114    1    5    1   19   21  561   22  685   35
        8    82   22  157   40   59  145   20  702    6   24
        9    62    0    4    0   20   19  603   11  445   82
        10  291   13   71    6  221   46   75   59  128  857

Overall Statistics

               Accuracy : 0.5647
                 95% CI : (0.5563, 0.5731)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5163

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.38462  0.73432  0.52494  0.87342  0.61317  0.62936
Specificity           0.94663  0.97551  0.95977  0.97998  0.95369  0.96349
Pos Pred Value        0.44712  0.77132  0.59244  0.81960  0.59769  0.65865
Neg Pred Value        0.93201  0.97027  0.94775  0.98673  0.95646  0.95872
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03881  0.07425  0.05261  0.08239  0.06187  0.06336
Detection Prevalence  0.08679  0.09627  0.08881  0.10052  0.10351  0.09619
Balanced Accuracy     0.66562  0.85491  0.74236  0.92670  0.78343  0.79642
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.41463  0.52388  0.33061   0.63670
Specificity           0.92504  0.95398  0.93355   0.92451
Pos Pred Value        0.38320  0.55847  0.35714   0.48500
Neg Pred Value        0.93365  0.94746  0.92587   0.95796
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.04187  0.05239  0.03321   0.06396
Detection Prevalence  0.10925  0.09381  0.09299   0.13187
Balanced Accuracy     0.66984  0.73893  0.63208   0.78060
```

```
cmtst$byClass
```

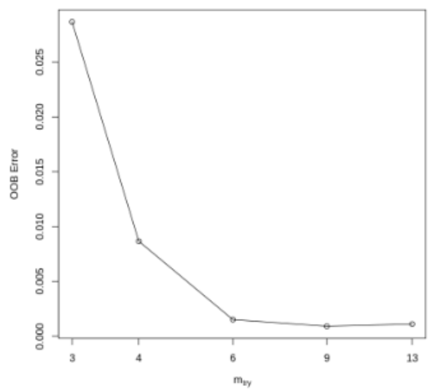
A matrix: 10 × 11 of type dbl

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3846154	0.9466301	0.4471195	0.9320095	0.4471195	0.3846154	0.4135189	0.10089552	0.03880597	0.08679104	0.6656228
Class: 2	0.7343173	0.9755085	0.7713178	0.9702725	0.7713178	0.7343173	0.7523629	0.10111940	0.07425373	0.09626866	0.8549129
Class: 3	0.5249442	0.9597744	0.5924370	0.9477477	0.5924370	0.5249442	0.5566522	0.10022388	0.05261194	0.08880597	0.7423593
Class: 4	0.8734177	0.9799769	0.8195991	0.9867253	0.8195991	0.8734177	0.8456530	0.09432836	0.08238806	0.10052239	0.9266973
Class: 5	0.6131657	0.9536853	0.5976929	0.9564638	0.5976929	0.6131657	0.6053304	0.10089552	0.06186567	0.10350746	0.7834255
Class: 6	0.6293551	0.9634885	0.6586501	0.9587152	0.6586501	0.6293551	0.6436694	0.10067164	0.06335821	0.09619403	0.7964218
Class: 7	0.4146341	0.9250436	0.3831967	0.9336461	0.3831967	0.4146341	0.3982961	0.10097015	0.04186567	0.10925373	0.6698389
Class: 8	0.5238806	0.9539801	0.5584726	0.9474594	0.5584726	0.5238806	0.5406238	0.10000000	0.05238806	0.09380597	0.7389303
Class: 9	0.3306092	0.9335490	0.3571429	0.9258680	0.3571429	0.3306092	0.3433642	0.10044776	0.03320896	0.09298507	0.6320791
Class: 10	0.6367013	0.9245064	0.4850028	0.9579644	0.4850028	0.6367013	0.5505943	0.10044776	0.06395522	0.13186567	0.7806039

Using TuneRF

```
set.seed(6871)
best_mtry = tuneRF(train_data, train_data$music_genre, stepFactor=1.5, improve= 0.01, ntreeTry=500, trace=FALSE, plot=TRUE)
```

```
-2.308524 0.01
0.8247347 0.01
0.3982187 0.01
-0.2208605 0.01
```




```
[ ] print(best_mtry)
```

```
    mtry    OOBError
3      3 0.028670500
4      4 0.008665647
6      6 0.001518788
9      9 0.000913978
13    13 0.001115840
```

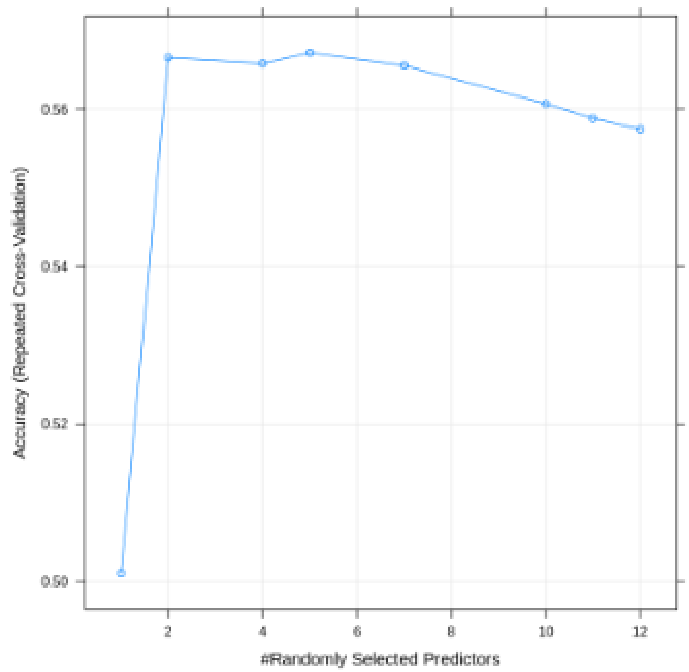
Using Random Search with CV

```
# using random search cv to find mtry
set.seed(6871)

control = trainControl(method="repeatedcv", number=10, search="random")
mtry = sqrt(ncol(train_data))
rf_random = train(as.factor(music_genre)~., data=train_data, method="rf", metric="Accuracy", trControl=control, tuneLength =10)
```

```
print(rf_random)
plot(rf_random)
```

```
Random Forest

31268 samples
   12 predictor
   10 classes: '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 28143, 28142, 28143, 28140, 28141, 28141, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   1    0.5010248  0.4455810
   2    0.5665550  0.5183817
   4    0.5657869  0.5175255
   5    0.5671613  0.5190505
   7    0.5655306  0.5172367
  10    0.5606696  0.5118354
  11    0.5588146  0.5097727
  12    0.5574391  0.5082450

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 5.
```

```
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 5.
```



There are multiple ways to tune the model parameters. We have used tuneRF method and random search with CV to find the mtry value. Mtry is the number of variables randomly sampled as candidates at each split. From the first method using tuneRF we obtained a mtry value of 9. We can see from the out of bag error vs mtry plot that the error is lowest for mtry=9. In the second method using random search , we trained the model for 10 folds 1 repeat and set mtry to a range of values till square root of number of columns. From the plot we can see that the accuracy increases steeply and then plateaus a bit and starts decreasing after a point. The highest accuracy was obtained when mtry=5. So with these optimal values of ntree and mtry we obtained we train the model again and test it.

```
[ ] predtrain_rf = predict(rf_random, train_data)
```

```
[ ] cmtrain = confusionMatrix(predtrain_rf, as.factor(train_data$music_genre))
    cmtrain
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  3155    0   23    0    0    0    0    0    0    1
        2     0 3162    0    0    0    0    0    0    0    0
        3     0    0 3110    0    0    0    0    0    0    2
        4     0    0    0 2947    0    0    0    1    0    0
        5     0    0    0    0 3135    0    0    6    7   51
        6     0    0    0    0    0 3141    1   43    1    0
        7     0    0    0    0    0    2 3076    5  486   10
        8     0    0    0    1    0    4    0 3073    0    2
        9     0    0    0    0    1    0   75    0 2635   22
        10    0    0    0    0   20    0    4    0   13 3053

Overall Statistics

               Accuracy : 0.975
                 95% CI : (0.9732, 0.9767)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9722

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            1.0000   1.0000  0.99266  0.99966   0.9933   0.9981
Specificity            0.9991   1.0000  0.99993  0.99996   0.9977   0.9984
Pos Pred Value         0.9925   1.0000  0.99936  0.99966   0.9800   0.9859
Neg Pred Value         1.0000   1.0000  0.99918  0.99996   0.9993   0.9998
Prevalence             0.1009   0.1011  0.10020  0.09428   0.1009   0.1006
Detection Rate         0.1009   0.1011  0.09946  0.09425   0.1003   0.1005
Detection Prevalence   0.1017   0.1011  0.09953  0.09428   0.1023   0.1019
Balanced Accuracy      0.9996   1.0000  0.99629  0.99981   0.9955   0.9982
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.97465  0.98242  0.83864   0.97198
Specificity           0.98211  0.99975  0.99652   0.99868
Pos Pred Value        0.85946  0.99773  0.96414   0.98803
Neg Pred Value        0.99711  0.99805  0.98223   0.99688
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.09838  0.09828  0.08427   0.09764
Detection Prevalence  0.11446  0.09850  0.08741   0.09882
Balanced Accuracy     0.97838  0.99108  0.91758   0.98533
```

```
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	1.0000000	0.9991463	0.9924505	1.0000000	0.9924505	1.0000000	0.9962109	0.10090188	0.10090188	0.10166944	0.9995732
Class: 2	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	0.10112575	0.10112575	0.10112575	1.0000000
Class: 3	0.9926588	0.9999289	0.9993573	0.9991831	0.9993573	0.9926588	0.9959968	0.10019829	0.09946271	0.09952667	0.9962939
Class: 4	0.9996608	0.9999647	0.9996608	0.9999647	0.9996608	0.9996608	0.9996608	0.09428169	0.09424971	0.09428169	0.9998127
Class: 5	0.9933460	0.9977234	0.9799937	0.9992518	0.9799937	0.9933460	0.9866247	0.10093386	0.10026225	0.10230907	0.9955347
Class: 6	0.9980934	0.9983998	0.9858757	0.9997863	0.9858757	0.9980934	0.9919469	0.10064603	0.10045414	0.10189331	0.9982466
Class: 7	0.9746515	0.9821073	0.8594579	0.9971108	0.8594579	0.9746515	0.9134373	0.10093386	0.09837534	0.11446207	0.9783794
Class: 8	0.9824169	0.9997512	0.9977273	0.9980488	0.9977273	0.9824169	0.9900129	0.10003838	0.09827939	0.09850326	0.9910841
Class: 9	0.8386378	0.9965157	0.9641420	0.9822323	0.9641420	0.8386378	0.8970213	0.10048612	0.08427146	0.08740565	0.9175767
Class: 10	0.9719834	0.9986845	0.9880259	0.9968770	0.9880259	0.9719834	0.9799390	0.10045414	0.09763976	0.09882308	0.9853340

```
[ ]  predtest_rf = predict(rf_random, test_data)
```

```
cmtest = confusionMatrix(predtest_rf, as.factor(test_data$music_genre))
cmtest
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   532   43   72   26  107   85   70   40   63  166
        2     9  985  113   38   13   66    0   12    0    4
        3    29  119  698   29   75   85    0  136    1   26
        4     2   89   34 1097    3   11    0  102    0    2
        5   153   34  116    5  810   24   15   78   11  132
        6    84   49   80   18   14  836    9  162    2   12
        7   120    1    7    1   16   24  563   25  672   38
        8    79   23  153   44   61  153   19  719    7   23
        9    57    0    2    0   26   16  602    8  456   84
        10  287   12   68    6  227   49   75   58  134  859

Overall Statistics

               Accuracy : 0.5638
                 95% CI : (0.5554, 0.5722)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5153

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.39349  0.72694  0.51973  0.86788  0.59911  0.61972
Specificity           0.94422  0.97883  0.95853  0.97998  0.95286  0.96432
Pos Pred Value        0.44186  0.79435  0.58264  0.81866  0.58781  0.66035
Neg Pred Value        0.93276  0.96957  0.94714  0.98615  0.95492  0.95772
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03970  0.07351  0.05209  0.08187  0.06045  0.06239
Detection Prevalence  0.08985  0.09254  0.08940  0.10000  0.10284  0.09448
Balanced Accuracy     0.66886  0.85288  0.73913  0.92393  0.77598  0.79202
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.41611  0.53657  0.33878    0.6382
Specificity           0.92496  0.95340  0.93405    0.9240
Pos Pred Value        0.38378  0.56128  0.36451    0.4839
Neg Pred Value        0.93380  0.94876  0.92674    0.9581
Prevalence            0.10097  0.10000  0.10045    0.1004
Detection Rate        0.04201  0.05366  0.03403    0.0641
Detection Prevalence  0.10948  0.09560  0.09336    0.1325
Balanced Accuracy     0.67054  0.74498  0.63641    0.7811
```

```
[ ]  cmtest$byClass
```

A matrix: 10 × 11 of type dbl

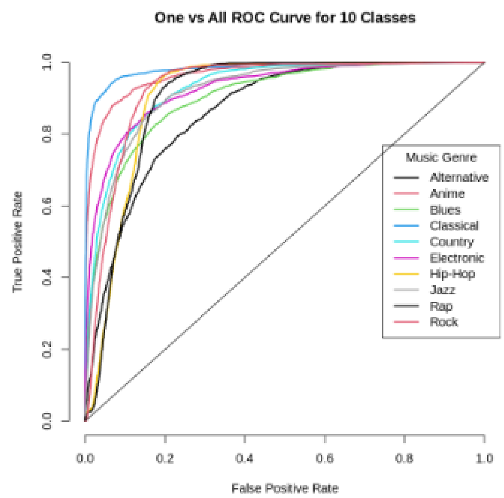
	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: 1	0.3934911	0.9442231	0.4418605	0.9327648	0.4418605	0.3934911	0.4162754	0.10089552	0.03970149	0.08985075	0.6688571
Class: 2	0.7269373	0.9788294	0.7943548	0.9695724	0.7943548	0.7269373	0.7591522	0.10111940	0.07350746	0.09253731	0.8528833
Class: 3	0.5197319	0.9585303	0.5826377	0.9471398	0.5826377	0.5197319	0.5493900	0.10022388	0.05208955	0.08940299	0.7391311
Class: 4	0.8678797	0.9799769	0.8186567	0.9861526	0.8186567	0.8678797	0.8425499	0.09432836	0.08186567	0.10000000	0.9239283
Class: 5	0.5991124	0.9528552	0.5878084	0.9549160	0.5878084	0.5991124	0.5934066	0.10089552	0.06044776	0.10283582	0.7759838
Class: 6	0.6197183	0.9643183	0.6603476	0.9577221	0.6603476	0.6197183	0.6393881	0.10067164	0.06238806	0.09447761	0.7920183
Class: 7	0.4161123	0.9249606	0.3837764	0.9337970	0.3837764	0.4161123	0.3992908	0.10097015	0.04201493	0.10947761	0.6705365
Class: 8	0.5365672	0.9533997	0.5612802	0.9487581	0.5612802	0.5365672	0.5486456	0.10000000	0.05365672	0.09559701	0.7449834
Class: 9	0.3387816	0.9340468	0.3645084	0.9267429	0.3645084	0.3387816	0.3511744	0.10044776	0.03402985	0.09335821	0.6364142
Class: 10	0.6381872	0.9240086	0.4839437	0.9581075	0.4839437	0.6381872	0.5504646	0.10044776	0.06410448	0.13246269	0.7810979

```
[25] rf_prob_test = predict(object = rf_random, newdata = test_data, type = "prob")
```

```
[26] roc_rf = multiclass.roc(test_data$music_genre, rf_prob_test)
     auc(roc_rf)
```

0.92325019485754

```
multiclass_roc_plot(test_data, rf_prob_test)
```




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

4.31528249490662
0.215223880597015

Summary of insights

- The random forest model underwent 10 fold cross-validation with 1 repeat and random search was done to find optimal mtry value based on accuracy of model.
- The training accuracy is 0.97, and the testing accuracy is 0.56 and this indicates some overfitting.
- Sensitivity/ recall is highest for the classical class (label 4) with a value 0.86 followed by the anime class (label 2) with a value of 0.72 for test.
- Sensitivity value of 1 for alternative and anime class (label 1 & 2) and 0.97-0.99 for other classes and 0.83 for Jazz (label 8) for training
- 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 0.81 and lowest for class rap with a value of 0.36 for testing which implies that the classical genre is predicted well by the model and the rap genre is not predicted that well.
- F1 score (harmonic mean of precision and recall) is highest for the classical genre and lowest for the rap genre for the train and test which means the classifier works well for classical genre.
- The area under the curve is 0.92 which implies the model's predictions are pretty decent.
- The balanced accuracy in test is high for classical and anime classes.
- From the multiclass roc plot, we can see that the model predicts classical, anime and hip hop better than alternative, rap.
- The model has similar accuracy for base model and model that was trained with the optimal mtry and ntree values selected.
- The model has 4.31 variance and 0.21 bias.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 0.51 which means its a decent model compared to random chance.
- We can see training accuracy and kappa values are a lot higher than test indicating overfitting.

Boosting

Boosting is an ensemble learning method where a random sample of data is selected, fitted with a model and then trained sequentially and each model tries to compensate for the weaknesses of its predecessor. Weak learners have low prediction accuracy, and Strong learners have higher prediction accuracy. Boosting converts a system of weak learners into a single strong learning system. It assigns weights to the output of individual trees. Then it gives incorrect classifications from the first decision tree a higher weight and inputs it to the next tree.

There are three types of boosting:

- Adaptive boosting
- Gradient boosting
- Extreme gradient boosting

In this task, Extreme gradient boosting has been implemented to perform multiclass classification of music genre. It is improvised over the other two models in terms of computational speed. It grows the tree upto max_depth and then prune backward until the improvement in loss function is below a threshold and supports regularization. The evaluation metric used is mlogloss which is used for multiclass classification problems

Parameters that can be tuned:

- Eta - It controls the learning rate. After every round, it shrinks the feature weights to reach the best optimum.
- Gamma - It controls regularization
- Max_depth - It controls the depth of the tree. Larger the depth, more complex the model, higher chances of overfitting

```
[ ] matrix_train = xgb.DMatrix(data = as.matrix(train_data[,1:12]), label = as.integer(as.factor(train_data$music_genre)))
    matrix_train

    xgb.DMatrix  dim: 31268 x 12  info: label  colnames: yes
```

```
[ ] matrix_test = xgb.DMatrix(data = as.matrix(test_data[,1:12]), label = as.integer(as.factor(test_data$music_genre)))
    matrix_test

    xgb.DMatrix  dim: 13400 x 12  info: label  colnames: yes
```

```
[ ] num_classes = length(unique(train_data$music_genre))
    num_classes

    10
```

```
set.seed(6871)
model_xgb = xgboost(data = matrix_train, nrounds = 50, verbose = 1, params = list(objective = "multi:softmax", num_class = num_classes+1))
```

```
[1]	train-mlogloss:1.869685
[2]	train-mlogloss:1.637611
[3]	train-mlogloss:1.481940
[4]	train-mlogloss:1.369009
[5]	train-mlogloss:1.283160
[6]	train-mlogloss:1.209753
[7]	train-mlogloss:1.154048
[8]	train-mlogloss:1.104335
[9]	train-mlogloss:1.065822
[10]	train-mlogloss:1.030215
[11]	train-mlogloss:1.001091
[12]	train-mlogloss:0.975661
[13]	train-mlogloss:0.952072
[14]	train-mlogloss:0.931756
[15]	train-mlogloss:0.911882
[16]	train-mlogloss:0.894613
[17]	train-mlogloss:0.880713
[18]	train-mlogloss:0.867513
[19]	train-mlogloss:0.853917
[20]	train-mlogloss:0.842969
[21]	train-mlogloss:0.831275
[22]	train-mlogloss:0.822312
[23]	train-mlogloss:0.812249
[24]	train-mlogloss:0.803599
[25]	train-mlogloss:0.795851
[26]	train-mlogloss:0.785817
[27]	train-mlogloss:0.778032
[28]	train-mlogloss:0.770833
[29]	train-mlogloss:0.763269
[30]	train-mlogloss:0.754188
[31]	train-mlogloss:0.747069
[32]	train-mlogloss:0.739869
[33]	train-mlogloss:0.734184
[34]	train-mlogloss:0.728746
[35]	train-mlogloss:0.722789
[36]	train-mlogloss:0.715969
[37]	train-mlogloss:0.710948
[38]	train-mlogloss:0.704883
[39]	train-mlogloss:0.700088
[40]	train-mlogloss:0.695203
[41]	train-mlogloss:0.689397
[42]	train-mlogloss:0.684692
[43]	train-mlogloss:0.680551
[44]	train-mlogloss:0.675301
[45]	train-mlogloss:0.669280
[46]	train-mlogloss:0.664025
[47]	train-mlogloss:0.659835
[48]	train-mlogloss:0.654725
[49]	train-mlogloss:0.651459
[50]	train-mlogloss:0.647871
```

An XGboost model was run with nrounds=50 and the objective parameter as multisoftmax which is apt for multiclass classification problem. Softmax turns logits into probabilities which will sum to 1.On basis of this, it makes the prediction which classes has the highest probabilities. We get a train accuracy of 0.77 and test accuracy as 0.61 and auc of 0.79, variance of 4.89 and bias of 0.28. Now we further hypertune parameters such as nrounds, gamma, eta to see if the model gives a better performance. The model was trained over a range of eta (0.01 and 0.001) and maxdepth of 2,3 and 4 and objective as multisoftprob. Different nround values were also tried and mlogloss kept decreasing uptil 500 nrounds. The best max_depth was found to be 4 and best eta was 0.01. Then 5 fold cross validation was performed with these new parameters.

```
[ ] pred_train_xgb = predict(model_xgb, matrix_train)
```

```
cmtr_xgb = confusionMatrix(as.factor(pred_train_xgb), as.factor(train_data$music_genre))
cmtr_xgb
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1  1799   50  108   37  125  112   32   69   45  140
        2    15 2776  107   41    9   41    1   15    0    6
        3    25  109 2312   41   68  116    0  152    0    2
        4     3   76   16 2734    2    8    0   83    0    1
        5   286   53  192   13 2389   55   22  107   19  117
        6    89   61   84   25   24 2511    7  173    4    6
        7   184    0    6    0   45   32 2415   41  633   53
        8   105   29  172   52   76  179    8 2423    9   26
        9   132    2    3    0   45   27  590   10 2213  108
        10  517    6  133    5  373   66   81   55  219 2682

Overall Statistics

               Accuracy : 0.7757
                 95% CI : (0.771, 0.7803)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7507

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.57021  0.87793  0.73795  0.92741   0.7570  0.79790
Specificity           0.97446  0.99164  0.98177  0.99333   0.9693  0.98318
Pos Pred Value        0.71474  0.92195  0.81841  0.93534   0.7344  0.84149
Neg Pred Value        0.95284  0.98634  0.97114  0.99245   0.9726  0.97751
Prevalence            0.10090  0.10113  0.10020  0.09428   0.1009  0.10065
Detection Rate        0.05753  0.08878  0.07394  0.08744   0.0764  0.08031
Detection Prevalence  0.08050  0.09630  0.09035  0.09348   0.1040  0.09543
Balanced Accuracy     0.77233  0.93478  0.85986  0.96037   0.8631  0.89054
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.76521  0.77462  0.70433   0.85387
Specificity           0.96464  0.97669  0.96740   0.94827
Pos Pred Value        0.70842  0.78694  0.70703   0.64830
Neg Pred Value        0.97340  0.97499  0.96698   0.98308
Prevalence            0.10093  0.10004  0.10049   0.10045
Detection Rate        0.07724  0.07749  0.07078   0.08577
Detection Prevalence  0.10903  0.09847  0.10010   0.13231
Balanced Accuracy     0.86493  0.87565  0.83586   0.90107
```

```
[ ] pred_test_xgb = predict(model_xgb, matrix_test)
```

```
cmtst_xgb = confusionMatrix(as.factor(pred_test_xgb), as.factor(test_data$music_genre))
cmtst_xgb
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   587   35   61   29   94   68   37   44   35   95
        2    12 1027   85   51   14   65    0   11    0    3
        3    25  114  771   22   49   65    0  121    1   10
        4     6   72   28 1091    2   10    0   92    0    1
        5   140   27  110    2  875   27   17   80    4  112
        6    77   43   69   17   20  902   10  150    4   14
        7   103    0    3    1   16   24  681   22  587   31
        8    71   23  136   43   59  132   18  758    3   28
        9    48    1    6    0   24   17  504   13  573   69
        10  283   13   74    8  199   39   86   49  139  983

Overall Statistics

               Accuracy : 0.6155
                 95% CI : (0.6072, 0.6238)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5728

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.43417  0.75793  0.57409  0.86313   0.6472  0.66864
Specificity           0.95867  0.97999  0.96624  0.98261   0.9569  0.96648
Pos Pred Value        0.54101  0.80994  0.65450  0.83794   0.6277  0.69066
Neg Pred Value        0.93788  0.97296  0.95320  0.98570   0.9603  0.96304
Prevalence            0.10090  0.10112  0.10022  0.09433   0.1009  0.10067
Detection Rate        0.04381  0.07664  0.05754  0.08142   0.0653  0.06731
Detection Prevalence  0.08097  0.09463  0.08791  0.09716   0.1040  0.09746
Balanced Accuracy     0.69642  0.86896  0.77017  0.92287   0.8021  0.81756
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.50333  0.56567  0.42571   0.73031
Specificity           0.93467  0.95746  0.94342   0.92617
Pos Pred Value        0.46390  0.59638  0.45657   0.52483
Neg Pred Value        0.94368  0.95202  0.93635   0.96851
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05082  0.05657  0.04276   0.07336
Detection Prevalence  0.10955  0.09485  0.09366   0.13978
Balanced Accuracy     0.71900  0.76157  0.68456   0.82824
```

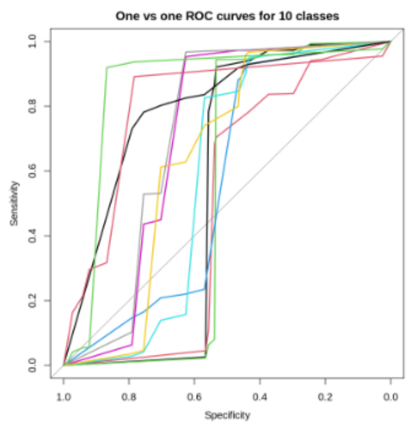
```
[ ] xgb_roc = multiclass.roc(as.numeric(test_data$music_genre), pred_test_xgb)
```

```
[ ] auc(xgb_roc)
```

0.765412085034738

```
[ ] xgb_roc_tst=xgb_roc$rocs
```

```
plot.roc(xgb_roc_tst[[1]], col=1, main="One vs one ROC curves for 10 classes")

for(i in 2:11)
{
  num=paste("1/",as.character(i),sep="")
  lines.roc(xgb_roc_tst[[i]],col=i)
}
```




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

4.89182803795783
0.288955223880597

```
max.depths = c(2,3,4)
etas = c(0.01, 0.001)
xgb_params = list("objective" = "multi:softprob","eval_metric" = "mlogloss", "num_class" = num_classes +1)
watchlist = list(train = matrix_train, valid = matrix_test)

best_params = 0
best_score = 0

count = 1
for( depth in max.depths ){
    for( num in etas){

        bst_grid = xgb.train(data = matrix_train,
                              max.depth = depth,
                              eta=num,
                              nround = 500,
                              watchlist = watchlist,
                              params = xgb_params,
                              early_stopping_rounds = 50,
                              verbose=0)

        if(count == 1){
            best_params = bst_grid$params
            best_score = bst_grid$best_score
            count = count + 1
            }
        else if( bst_grid$best_score < best_score){
            best_params = bst_grid$params
            best_score = bst_grid$best_score
        }
    }
}

best_params
best_score
```

$objective
    'multi:softprob'
$eval_metric
    'mlogloss'
$num_class
    11
$max_depth
    4
$eta
    0.01
$validate_parameters
    TRUE

1.12133915640414

```
xgb_cv_model = xgb.cv(params = list(objective = "multi:softprob", num_class = num_classes + 1),
data = matrix_train, nrounds = 500, max_depth=4, eta=0.01, nfold = 5, prediction = TRUE, verbose=1,
print_every_n = 100, early_stopping_rounds = 20)
```

```
[1]	train-mlogloss:2.381439+0.000047	test-mlogloss:2.381882+0.000207
Multiple eval metrics are present. Will use test_mlogloss for early stopping.
Will train until test_mlogloss hasn't improved in 20 rounds.

[101]	train-mlogloss:1.626824+0.001598	test-mlogloss:1.652192+0.003630
[201]	train-mlogloss:1.357305+0.001613	test-mlogloss:1.397423+0.005604
[301]	train-mlogloss:1.213534+0.001811	test-mlogloss:1.266024+0.006881
[401]	train-mlogloss:1.126236+0.002142	test-mlogloss:1.189438+0.007501
[500]	train-mlogloss:1.065715+0.002228	test-mlogloss:1.138706+0.008289
```

```
[ ] print(xgb_cv_model)
```

```
##### xgb.cv 5-folds
    iter train_mlogloss_mean train_mlogloss_std test_mlogloss_mean
       1            2.381439       4.666120e-05           2.381882
       2            2.365484       9.779402e-05           2.366364
       3            2.349990       1.677133e-04           2.351320
       4            2.334929       2.434834e-04           2.336673
       5            2.320273       3.032425e-04           2.322445
---
     496            1.067751       2.208360e-03           1.140358
     497            1.067176       2.212254e-03           1.139886
     498            1.066680       2.226760e-03           1.139486
     499            1.066192       2.215554e-03           1.139088
     500            1.065715       2.228458e-03           1.138706
    test_mlogloss_std
         0.0002065310
         0.0003995618
         0.0005514875
         0.0006909063
         0.0007977802
---
         0.0082511566
         0.0082595449
         0.0082480870
         0.0082724342
         0.0082887410
Best iteration:
 iter train_mlogloss_mean train_mlogloss_std test_mlogloss_mean
  500            1.065715        0.002228458           1.138706
 test_mlogloss_std
       0.008288741
```

```
[ ] OOF_pred = data.frame(xgb_cv_model$pred) %>% mutate(max_prob = max.col(., ties.method = "last"),label = train_data$music_genre + 1)
head(OOF_pred)
```

A data.frame: 6 × 13

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	max_prob	label
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<dbl>
1	0.005667632	0.014225382	0.073727295	0.24857217	0.010000563	0.017651882	0.5883462	0.005914307	0.02384668	0.005838550	0.006209348	7	7
2	0.008089828	0.010184684	0.097377308	0.07467560	0.010335529	0.008700877	0.7050706	0.008871172	0.05968373	0.008332180	0.008678457	7	7
3	0.002697297	0.036768783	0.004744875	0.01217625	0.003257451	0.004750345	0.7574473	0.032326311	0.09030898	0.007036017	0.048486371	7	7
4	0.008062874	0.246866167	0.030153869	0.03655961	0.016528782	0.162027672	0.1602805	0.275698096	0.02991840	0.018486040	0.015418059	8	7
5	0.004153100	0.006745202	0.411396325	0.07290916	0.006071773	0.007331206	0.4490260	0.012585230	0.01200295	0.013288095	0.004490932	7	7
6	0.007790297	0.011845345	0.197966114	0.05182024	0.028304169	0.009360069	0.6327074	0.008092253	0.03567597	0.008014151	0.008423994	7	7

```
confusionMatrix(factor(OOF_prediction$max_prob),factor(OOF_prediction$label))
```

```
Confusion Matrix and Statistics

          Reference
Prediction    2    3    4    5    6    7    8    9   10   11
        2  1332   91  154   65  202  200  106  135   93  246
        3    37 2386  218  133   31  109    3   32    3    9
        4    55  228 1839   82  149  175    4  304    3   25
        5    10  177   41 2438    5   26    0  185    0    7
        6   354   89  244   18 1981   86   33  147   32  194
        7   163  125  157   45   43 2041   24  349   13   25
        8   234    2    8    0   56   57 1573   54 1345  117
        9   175   44  313  153  139  326   25 1785   20   71
        10  162    0    6    1   48   40 1251   23 1321  176
        11  633   20  153   13  502   87  137  114  312 2271

Overall Statistics

               Accuracy : 0.6066
                 95% CI : (0.6012, 0.612)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5629

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 2 Class: 3 Class: 4 Class: 5 Class: 6 Class: 7
Sensitivity           0.42219  0.75459  0.58698  0.82700  0.62769  0.64855
Specificity           0.95404  0.97954  0.96357  0.98407  0.95742  0.96643
Pos Pred Value        0.50762  0.80581  0.64211  0.84389  0.62335  0.68375
Neg Pred Value        0.93636  0.97259  0.95444  0.98203  0.95817  0.96090
Prevalence            0.10090  0.10113  0.10020  0.09428  0.10093  0.10065
Detection Rate        0.04260  0.07631  0.05881  0.07797  0.06336  0.06527
Detection Prevalence  0.08392  0.09470  0.09160  0.09239  0.10164  0.09547
Balanced Accuracy     0.68811  0.86706  0.77527  0.90554  0.79256  0.80749
                     Class: 8 Class: 9 Class: 10 Class: 11
Sensitivity           0.49842  0.57065   0.42043   0.72302
Specificity           0.93337  0.95501   0.93931   0.92992
Pos Pred Value        0.45647  0.58505   0.43626   0.53536
Neg Pred Value        0.94310  0.95240   0.93552   0.96781
Prevalence            0.10093  0.10004   0.10049   0.10045
Detection Rate        0.05031  0.05709   0.04225   0.07263
Detection Prevalence  0.11021  0.09758   0.09684   0.13567
Balanced Accuracy     0.71589  0.76283   0.67987   0.82647
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4    5    6    7    8    9   10
        1   518   47   69   32   86   80   24   46   28   76
        2     3  937  127   43    9   72    0   14    0    3
        3    13  138  676   29   44   90    0  124    1    2
        4     3  100   37 1095    3   13    0  122    0    4
        5   167   45  138    6  818   25   21   90   10   73
        6    85   49   68   16   21  859   13  177    7   10
        7   132    1    8    1   31   25  763   32  585   35
        8    79   22  137   34   60  112   14  656    2   20
        9    50    1    8    0   20   17  411    9  524   67
        10  302   15   75    8  260   56  107   70  189 1056

Overall Statistics

               Accuracy : 0.5897
                 95% CI : (0.5813, 0.598)
    No Information Rate : 0.1011
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5441

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity           0.38314  0.69151  0.50335  0.86630  0.60503  0.63677
Specificity           0.95950  0.97750  0.96342  0.97676  0.95227  0.96299
Pos Pred Value        0.51491  0.77566  0.60519  0.79521  0.58722  0.65824
Neg Pred Value        0.93271  0.96572  0.94570  0.98594  0.95553  0.95949
Prevalence            0.10090  0.10112  0.10022  0.09433  0.10090  0.10067
Detection Rate        0.03866  0.06993  0.05045  0.08172  0.06104  0.06410
Detection Prevalence  0.07507  0.09015  0.08336  0.10276  0.10396  0.09739
Balanced Accuracy     0.67132  0.83451  0.73339  0.92153  0.77865  0.79988
                     Class: 7 Class: 8 Class: 9 Class: 10
Sensitivity           0.56393  0.48955  0.38930   0.78455
Specificity           0.92944  0.96020  0.95163   0.91024
Pos Pred Value        0.47303  0.57746  0.47335   0.49392
Neg Pred Value        0.94994  0.94423  0.93313   0.97425
Prevalence            0.10097  0.10000  0.10045   0.10045
Detection Rate        0.05694  0.04896  0.03910   0.07881
Detection Prevalence  0.12037  0.08478  0.08261   0.15955
Balanced Accuracy     0.74669  0.72488  0.67047   0.84739
```

```
[ ] xgb_roc = multiclass.roc(as.numeric(test_data$music_genre), pred_test_xgb)
```

```
auc(xgb_roc)
```

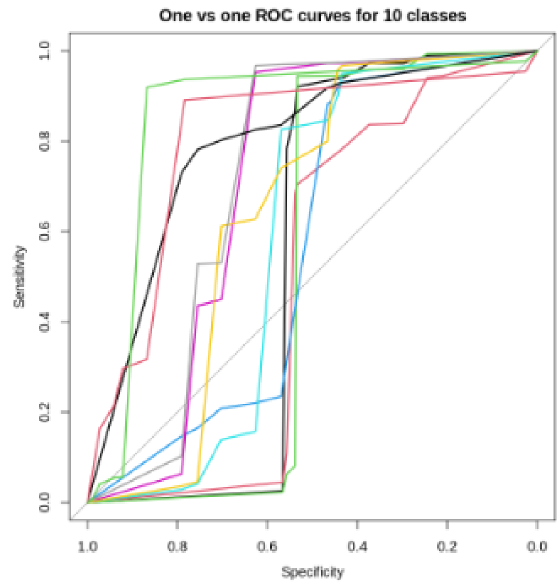
0.765412085034738

```
[ ] xgb_roc_tst=xgb_roc$rocs
```

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

4.89182803795783
0.288955223880597

```
plot.roc(xgb_roc_tst[[1]], col=1, main="One vs one ROC curves for 10 classes")
for(i in 2:11)
{
  num=paste("1/",as.character(i),sep="")
  lines.roc(xgb_roc_tst[[i]],col=i)
}
```



Summary of insights

- On running the cross validated model we find the train accuracy dropped to 0.60 and test accuracy to 0.58. On comparing the train, test accuracies of the original model compared to cross validated model we see overfitting has reduced.
- The sensitivity/recall is highest for classical genre (label 4) with a value of 0.86 and rock genre (label 10) for 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 which implies that the classical genre is predicted well by the model.
- F1 score (harmonic mean of precision and recall) is highest for the classical genre the train and test which means the classifier works well for classical genre.
- The area under the curve is 0.76 which implies model's predictions are decent.
- The balanced accuracy in test is high for classical, rock and anime classes.
- From the multiclass roc plot, we can see that the model predicts classical, anime and rock better than alternative.
- The model has 4.76 variance and 0.37 bias and the variance has decreased a little and bias increased a little compared to the base model. 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 0.54 which means its a decent model compared to random chance.

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 - CV	Train - 0.51 Test - 0.51	0.46	0.90	4.05	0.11
K-Nearest Neighbors - CV	Test - 0.52	0.47	-	4.02	0.16
Support Vector Machines - CV	Train - 0.61 Test - 0.57	0.52	0.74	4.51	0.23
Decision Trees - CV	Train - 0.51 Test - 0.50	0.45	0.87	4.52	0.33
Random Forest	Train - 0.97 Test - 0.56	0.51	0.92	4.27	0.20
Random Forest - CV	Train - 0.97 Test - 0.56	0.51	0.92	4.31	0.21
XGBoost	Train - 0.77 Test - 0.61	0.57	0.76	4.89	0.28
XGBoost - CV	Train - 0.61 Test - 0.58	0.54	0.76	4.76	0.37

From the table above and the analysis done so far:

- We can see that random forest has a high train accuracy of 0.97 but a test accuracy of 0.56 indicating overfitting.
- This model was then cross validated to help with the overfitting, but still the model seems to be overfitting, so that did not help much. The parameters such

as mtry or ntree can be tuned further or regularized random forest can be done to reduce overfitting

- XGboost has a train accuracy of 0.77 and test accuracy of 0.61 and this is also overfitting on the data
- We can see the cross validated XGboost model has a train accuracy of 0.61 and test of 0.58. Although its not any improvement in performance, it is still less overfitting than the original XGBoost.
- KNN model in the last task had a very high test accuracy of 0.99 and it was suspected to be overfitting the data. The cross validated KNN has a test accuracy of 0.52 but its variance is the lowest of all models.
- Cross validation reduces bias as we use most of the data for fitting, and it also reduces variance as most of the data is also being used in the validation set. In the KNN model, after cross validation the variance reduced to 4.02 from 8.28 previously.
- Overall all models seems to have similar performance in terms of accuracy, variance and bias.
- The AUC of random forest and logistic regression is high ~0.90 but their accuracies are lower around ~0.50, this occurs when the classifier performs well on the positive class leading to high AUC, at the cost of a high false negatives rate or a low number of true negatives.
- Cross validation did not impact the performance of the multinominal logistic regression model in this case.
- SVM with cross validation although has decent performance is computationally very expensive and time consuming model.

References

1. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/
2. https://afit-r.github.io/random_forests#:~:text=mtry%20%3A%20the%20number%20of%20variables,as%20candidates%20at%20each%20split.
3. https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/
4. https://analyticsindiamag.com/complete-guide-to-xgboost-with-implementation-in-r/
5. https://rpubs.com/mharris/multiclass_xgboost
6. https://www.kaggle.com/code/camnugent/gradient-boosting-and-parameter-tuning-in-r/notebook
7. https://xgboost.readthedocs.io/en/stable/R-package/xgboostPresentation.html
8. http://inferate.blogspot.com/2015/05/k-fold-cross-validation-with-decision.html
9. https://rpubs.com/ippromek/336732
10. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/
11. https://rstudio-pubs-static.s3.amazonaws.com/456044_9c275b0718a64e6286751bb7c60ae42a.html
12. https://www.rdocumentation.org/packages/xgboost/versions/1.6.0.1/topics/xgb.train
13. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/
14. http://inferate.blogspot.com/2015/05/k-fold-cross-validation-with-decision.html
15. https://www.kaggle.com/code/hamelg/intro-to-r-part-29-decision-trees/notebook
16. https://www.datacamp.com/tutorial/decision-trees-R

17. https://www.edureka.co/blog/implementation-of-decision-tree/
18. http://www.science.smith.edu/~jcrouser/SDS293/labs/lab14-r.html
19. https://appsilon.com/r-xgboost/
20. https://cran.r-project.org/web/packages/xgboost/xgboost.pdf
21. https://neptune.ai/blog/ensemble-learning-guide
22. http://www.sthda.com/english/articles/38-regression-model-validation/157-cross-validation-essentials-in-r/
23. https://rforhr.com/kfold.html
24. https://statsmaths.github.io/stat389-s21/notebooks/notebook05.html
25. https://remiller1450.github.io/s230f19/caret3.html
26. https://rpubs.com/markloessi/506713
27. https://www.rdocumentation.org/packages/caret/versions/4.47/topics/train
28. https://www.r-bloggers.com/2021/04/random-forest-in-r/
29. https://www.edureka.co/blog/random-forest-classifier/
30. https://www.guru99.com/r-random-forest-tutorial.html
31. https://www.listendata.com/2014/11/random-forest-with-r.html