ECON 187 Final Project: Music Genre and Popularity Prediction

Yucong Chen, Tianhui Zhao

Due 6/16/2023

1. Introduction

Music Information Retrieval (MIR) is the research of studying information in music. Two Important applications of MIR are music classification and algorithmic composition. Music classification includes genre, mood, artist classification, and it helps distinguish different music. Algorithmic composition is the process of using algorithms and programming to write music. These applications have important goals, including protecting intellectual property for musicians, discovering music cognition and music therapy for patients, and developing music in general. In this project, we aim to predict music genre, specifically Classical music, and predicting music popularity based on different music characteristics.

2. Data and Models

2.1. Description of Data

In summary, this data set includes about 40,000 observations with 11 variables.

- Popularity: a number ranged from 0 to 100 representing how popular the song is; larger value means more popular.
- Danceability: a number ranged from 0 to 1 representing how likely can we dance through the music; larger value means more possible to dance.
- Acousticness: a number ranged from 0 to 1 measuring if the song uses instruments and no electronic components; larger value means more instruments.
- Energy: a number ranged from 0 to 1 measuring if the song makes you want to move forward; larger value means more energy.
- Instrumentalness: a number between 0 to 1 representing if the music is instrumental versus vocal; the higher the more instrumental the music is.
- Liveness: a number interpreting if the music is "live"; higher means more possibility.
- Valence: measures positiveness; higher value means more positive.
- Loudness: records loudness in dB.
- Speechiness: measures the presence of spoken words in a song.
- Tempo: measures how fast the song is in BPM (beats per minute).
- Music Genre: is the music genre of each song. We encode this data three times for different purposes, which will be illustrated below.

```
# Load Data
library(readr)
library(readxl)
library(MASS)
library(lessR)
```

```
##
## lessR 4.2.9                         feedback: gerbing@pdx.edu
## --------------------------------------------------------------
```

```
## > d <- Read("")   Read text, Excel, SPSS, SAS, or R data file
##   d is default data frame, data= in analysis routines optional
##
## Learn about reading, writing, and manipulating data, graphics,
## testing means and proportions, regression, factor analysis,
## customization, and descriptive statistics from pivot tables
##   Enter:  browseVignettes("lessR")
##
## View changes in this and recent versions of lessR
##   Enter: news(package="lessR")
##
## Interactive data analysis
##   Enter: interact()
```

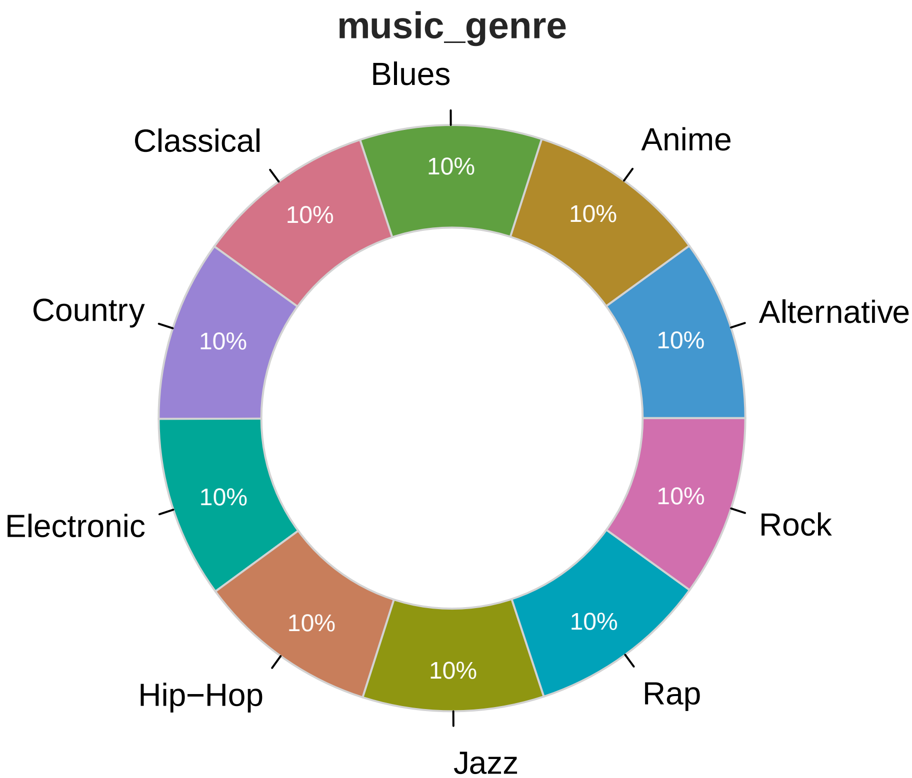
```
data <- read_csv("music_genre.csv")
```

```
## Rows: 50005 Columns: 18
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): artist_name, track_name, key, mode, tempo, obtained_date, music_genre
## dbl (11): instance_id, popularity, acousticness, danceability, duration_ms, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

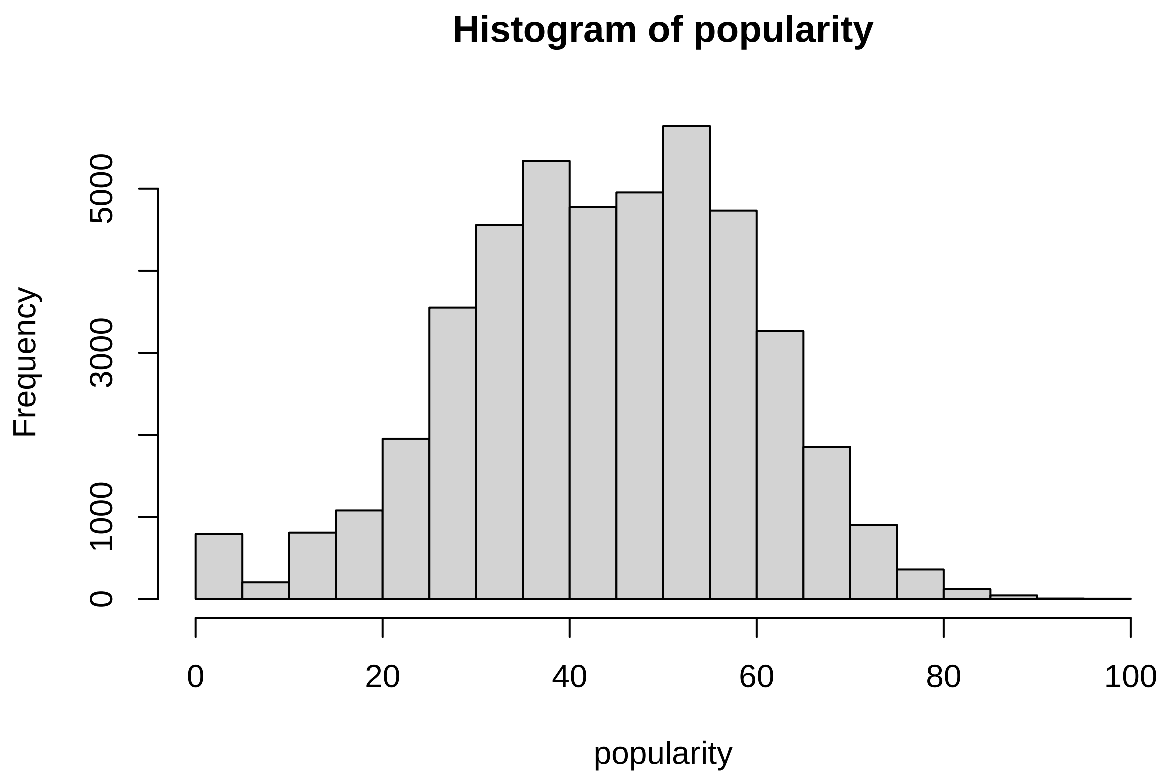
```
data = na.omit(data) # delete NA data
data = data[data$duration_ms > 0, ]
data = data[, -1]
data = data[, -1]
data = data[, -1]
data = data[, -7]
data = data[, -9]
data = data[, -11]

PieChart(music_genre, values = "%", data = data)
```




```
## >>> suggestions
## PieChart(music_genre, hole=0)  # traditional pie chart
## PieChart(music_genre, values="%")  # display %'s on the chart
## PieChart(music_genre)  # bar chart
## Plot(music_genre)  # bubble plot
## Plot(music_genre, values="count")  # lollipop plot
##
## --- music_genre ---
##
## music_genre   Count   Prop
## ---------------------------
## Alternative    4509   0.100
##       Anime    4527   0.100
##       Blues    4517   0.100
##   Classical    4489   0.100
##     Country    4508   0.100
##  Electronic    4517   0.100
##     Hip-Hop    4510   0.100
##        Jazz    4503   0.100
##         Rap    4488   0.100
##        Rock    4493   0.100
## ---------------------------
##       Total   45061   1.000
##
## Chi-squared test of null hypothesis of equal probabilities
##   Chisq = 0.334, df = 9, p-value = 1.000
```

```r
hist(data$popularity, xlab="popularity", main="Histogram of popularity")
```




```r
mean(data$popularity)
```

```
## [1] 44.23364
```

From the pie chart and histogram above, we see that the genres are evenly distributed and the mean of popularity is about 44. Now we will encode our music genre in three ways for different models below. (1) In general, we encode 1 for blues, 2 for anime, 3 for alternative, 4 for rock, 5 for rap, 6 for jazz, 7 for hip-hop, 8 for electronic, 9 for country, and 10 for classical music. (2) When we do the music classification models, we will have genre = 1 for classical music, and 0 otherwise for having a binary variable. (3) When we do models for predicting music popularity, we treat it more like a time variable, so that we can see how new and past music genres affect popularity. So we will have 1 for classical, 2 for blues, rock, jazz, country, and 3 for anime, alternative, rap, hip-hop, electronic.

```r
# (1) Encode music_genre to 1 to 10
data$music_genre[data$music_genre == "Blues"] <- 1
data$music_genre[data$music_genre == "Anime"] <- 2
data$music_genre[data$music_genre == "Alternative"] <- 3
data$music_genre[data$music_genre == "Rock"] <- 4
data$music_genre[data$music_genre == "Rap"] <- 5
data$music_genre[data$music_genre == "Jazz"] <- 6
data$music_genre[data$music_genre == "Hip-Hop"] <- 7
data$music_genre[data$music_genre == "Electronic"] <- 8
data$music_genre[data$music_genre == "Country"] <- 9
data$music_genre[data$music_genre == "Classical"] <- 10
data$music_genre <- gsub(",", "", data$music_genre)
```

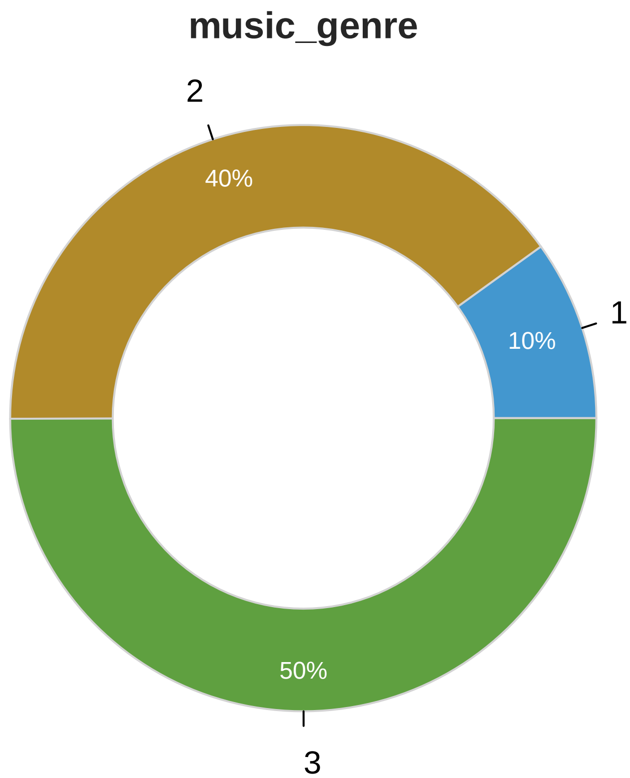
```r
data$music_genre <- as.numeric(as.character(data$music_genre))

# Clean Data set
data$popularity <- gsub(",", "", data$popularity)
data$popularity <- as.numeric(as.character(data$popularity))
data$acousticness <- gsub(",", "", data$acousticness)
data$acousticness <- as.numeric(as.character(data$acousticness))
data$danceability <- gsub(",", "", data$danceability)
data$danceability <- as.numeric(as.character(data$danceability))
data$duration_ms <- gsub(",", "", data$duration_ms)
data$duration_ms <- as.numeric(as.character(data$duration_ms))
data$energy <- gsub(",", "", data$energy)
data$energy <- as.numeric(as.character(data$energy))
data$instrumentalness <- gsub(",", "", data$instrumentalness)
data$instrumentalness <- as.numeric(as.character(data$instrumentalness))
data$liveness <- gsub(",", "", data$liveness)
data$liveness <- as.numeric(as.character(data$liveness))
data$loudness <- gsub(",", "", data$loudness)
data$loudness <- as.numeric(as.character(data$loudness))
data$speechiness <- gsub(",", "", data$speechiness)
data$speechiness <- as.numeric(as.character(data$speechiness))
data$tempo <- gsub(",", "", data$tempo)
data$tempo <- as.numeric(as.character(data$tempo))
data$valence <- gsub(",", "", data$valence)
data$valence <- as.numeric(as.character(data$valence))
data = na.omit(data) # delete NA data

# (2) Encode music_genre = Classical to 1, genre != Classical to 0
data1 <- data
data1$music_genre[data1$music_genre == "Classical"] <- 1
data1$music_genre[data1$music_genre != "1"] <- 0
data1$music_genre <- gsub(",", "", data1$music_genre)
data1$music_genre <- as.numeric(as.character(data1$music_genre))

# (3) Encode as a time variable
data2 = data
data2$music_genre[data2$music_genre == 2] <- 3
data2$music_genre[data2$music_genre == 1] <- 2
data2$music_genre[data2$music_genre == 4] <- 2
data2$music_genre[data2$music_genre == 5] <- 3
data2$music_genre[data2$music_genre == 6] <- 2
data2$music_genre[data2$music_genre == 7] <- 3
data2$music_genre[data2$music_genre == 8] <- 3
data2$music_genre[data2$music_genre == 9] <- 2
data2$music_genre[data2$music_genre == 10] <- 1

data2$music_genre <- gsub(",", "", data2$music_genre)
data2$music_genre <- as.numeric(as.character(data2$music_genre))
PieChart(music_genre, values = "%", data = data2)
```

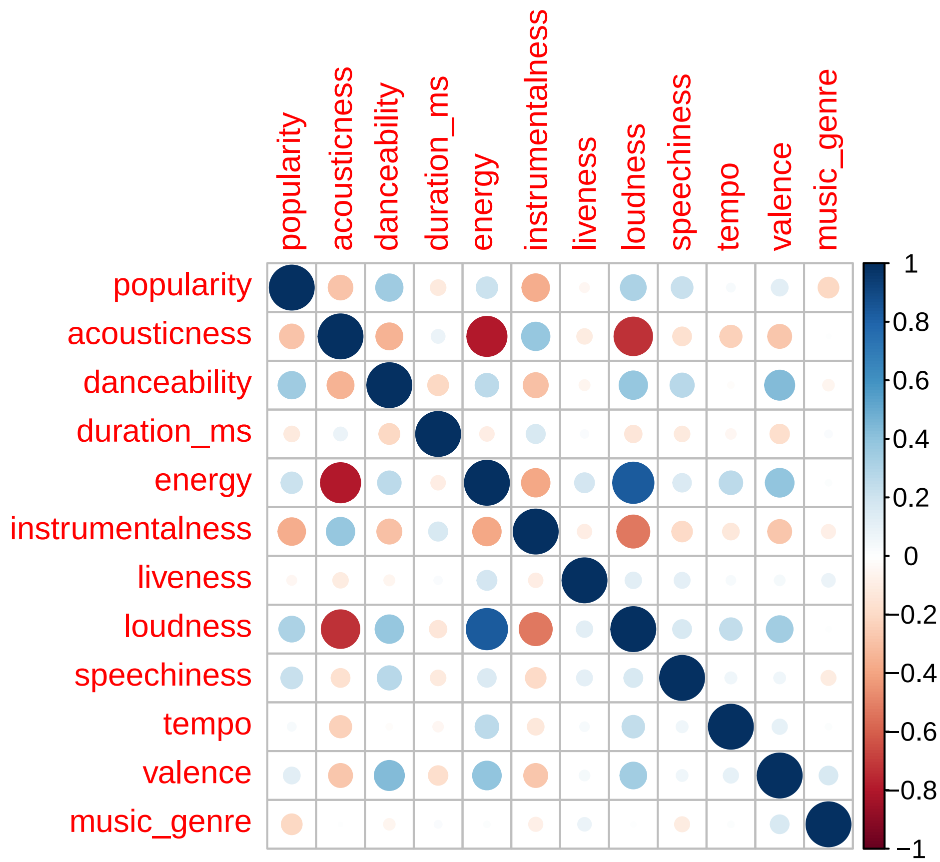



```
## >>> suggestions
## PieChart(music_genre, hole=0)  # traditional pie chart
## PieChart(music_genre, values="%")  # display %'s on the chart
## PieChart(music_genre)  # bar chart
## Plot(music_genre)  # bubble plot
## Plot(music_genre, values="count")  # lollipop plot
##
## --- music_genre ---
##
##                       1      2      3       Total
## Frequencies:       4036  16258  20266       40560
## Proportions:      0.100  0.401  0.500       1.000
##
## Chi-squared test of null hypothesis of equal probabilities
##   Chisq = 10573.330, df = 2, p-value = 0.000
```

```
# Correlation plot for music_genre takes value of 0 and 1
# music_genre = 1 if Classical music, = 0 otherwise
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
corrplot(cor(data1))
```



Here we have our correlation plot for when genre takes value of 0 or 1. We see that music genre does not show significant relationship between any variables, so it is not determined by one variable. Therefore, we propose potential models for predicting music genre can be logit, kNN, classification trees, and more. We also notice that there is a strong relationship between loudness and instrumentalness. What's more, instrumentalness and valence seem to have strong relationship with other variables. There is another interesting observation that valence and music genre has a positive relationship, which is maybe a bit counterintuitive but may be subject to particular songs.

2.2. Predicting Classical Music

Logit Model

Here we present our logit model to predict if a song is classical or not.

```
# Divide data set into training and testing
training = data1 [data$music_genre == 1, ]
training2 = data1[data$music_genre != 1, ]
data1$index = (1:40560)
train = data1 [data1$index %% 2 == 0, ]
test = data1 [data1$index %% 2 != 0, ]
```

```
# Logit regression
logit_model <- glm(music_genre ~ popularity + acousticness + danceability
                   + duration_ms + energy + instrumentalness + liveness
                   + loudness + speechiness + valence,
                   family=binomial(link="logit"), data = train)
summary(logit_model)
```

```
##
## Call:
## glm(formula = music_genre ~ popularity + acousticness + danceability +
##     duration_ms + energy + instrumentalness + liveness + loudness +
##     speechiness + valence, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1111  -0.4567  -0.2834  -0.1650   3.3995
##
## Coefficients:
##                      Estimate    Std. Error z value             Pr(>|z|)
## (Intercept)       1.1203048813  0.2639513331   4.244           0.00002192 ***
## popularity       -0.0515586490  0.0018316034 -28.149 < 0.0000000000000002 ***
## acousticness     -0.5871963587  0.1293019729  -4.541           0.00000559 ***
## danceability     -2.3810369042  0.2036140037 -11.694 < 0.0000000000000002 ***
## duration_ms       0.0000010078  0.0000002168   4.649           0.00000334 ***
## energy           -1.9516031187  0.2350871971  -8.302 < 0.0000000000000002 ***
## instrumentalness -2.1209237924  0.1182677417 -17.933 < 0.0000000000000002 ***
## liveness          1.1959095785  0.1383227257   8.646 < 0.0000000000000002 ***
## loudness          0.0312907082  0.0091158989   3.433             0.000598 ***
## speechiness      -4.6914480005  0.4387415952 -10.693 < 0.0000000000000002 ***
## valence           3.5800010113  0.1383572598  25.875 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13163  on 20279  degrees of freedom
## Residual deviance: 10651  on 20269  degrees of freedom
## AIC: 10673
##
## Number of Fisher Scoring iterations: 6
```

```
# Access Logistic Regression
logit.probs = predict(logit_model, test, type="response")
logit.pred = rep(0, length(logit.probs))
logit.pred[logit.probs > 0.5] = 1
pred_tb = table(logit.pred, test$music_genre)
print(pred_tb)
```

```
##
## logit.pred     0     1
##          0 18065  1921
##          1   192   102
```

```
# Calculate and display training error (misclassification rate)
train_er = (pred_tb[2,1] + pred_tb[1,2]) / sum(pred_tb)

# Calculate the False positive rate:
# The fraction of negative examples that are classified as positive
f_up = (pred_tb[1,2]) / (pred_tb[1,2] + pred_tb[2,2])

# Calculate the False negative rate:
```

```r
# The fraction of positive examples that are classified as negative
f_down = (pred_tb[2,1]) / (pred_tb[1,1] + pred_tb[2,1])

# Print the results
cat(paste0("The training error is ", train_er,
       ".\nThe correct up rate is ", 1 - f_up,
       ".\nThe correct down rate is ", 1-f_down))
```

```
## The training error is 0.104191321499014.
## The correct up rate is 0.0504201680672269.
## The correct down rate is 0.989483485786274
```

```r
# Overall fraction of correct predictions
pred.corr = mean(logit.pred == test$music_genre)
cat(paste0("\nThe Overall fraction of correct predictions is ", pred.corr))
```

```
##
## The Overall fraction of correct predictions is 0.895808678500986
```

From the summary above, we see that all estimates are statistically significant, so all variables are important for our prediction. Among all estimates, popularity, acousticness, danceability, energy, instrumentalness, and speechiness have negative estimates, while duration, liveness, loudness, and valence have positive estimates. Speechiness, valence, and instrumentalness has the largest values.

LDA

```r
# LDA
library(MASS)
lda.fit = lda(music_genre ~ popularity + acousticness + danceability
              + duration_ms + energy + instrumentalness + liveness
              + loudness + speechiness + valence, data = train)
print(lda.fit)
```

```
## Call:
## lda(music_genre ~ popularity + acousticness + danceability +
##     duration_ms + energy + instrumentalness + liveness + loudness +
##     speechiness + valence, data = train)
##
## Prior probabilities of groups:
##          0          1
## 0.90024655 0.09975345
##
## Group means:
##   popularity acousticness danceability duration_ms    energy instrumentalness
## 0   45.28285    0.3044016    0.5624038    244771.3 0.5983053       0.19423077
## 1   34.93327    0.3042658    0.5292007    252316.9 0.6146049       0.08947659
##    liveness loudness speechiness   valence
## 0 0.1898827 -9.17238  0.09759182 0.4422794
## 1 0.2353842 -8.88416  0.06145358 0.5781762
##
## Coefficients of linear discriminants:
##                              LD1
## popularity       -0.0475302171202
## acousticness     -0.5506971143222
## danceability     -1.4267570392594
```

```
## duration_ms       0.0000008856902
## energy           -1.5575334279799
## instrumentalness -1.6584089984606
## liveness          1.2467923693925
## loudness          0.0142453523573
## speechiness      -2.3221966953301
## valence           2.7291260691811
```

```r
lda.pred = predict(lda.fit, test)
pred_tb = table(lda.pred$class, test$music_genre)
print(pred_tb)
```

```
##
##         0     1
##   0 18036  1909
##   1   221   114
```

```r
# Measure prediction
train_er = (pred_tb[2,1] + pred_tb[1,2]) / sum(pred_tb)
f_up = (pred_tb[1,2]) / (pred_tb[1,2] + pred_tb[2,2])
f_down = (pred_tb[2,1]) / (pred_tb[1,1] + pred_tb[2,1])

# Print the results
cat(paste0("The training error is ", train_er,
       ".\nThe correct up rate is ", 1 - f_up,
       ".\nThe correct down rate is ", 1-f_down))
```

```
## The training error is 0.105029585798817.
## The correct up rate is 0.0563519525457242.
## The correct down rate is 0.987895053951909
```

```r
# Overall fraction of correct predictions
pred.corr = mean(lda.pred$class == test$music_genre)
cat(paste0("\nThe Overall fraction of correct predictions is ", pred.corr))
```

```
##
## The Overall fraction of correct predictions is 0.894970414201183
```

QDA

```r
# QDA
qda.fit = qda(music_genre ~ popularity + acousticness + danceability
              + duration_ms + energy + instrumentalness + liveness
              + loudness + speechiness + valence, data = train)
qda.class = predict(qda.fit, test)$class
pred_tb = table(qda.class, test$music_genre)
print(pred_tb)
```

```
##
## qda.class     0     1
##         0 16964  1032
##         1  1293   991
```

```r
# Measure prediction
train_er = (pred_tb[2,1] + pred_tb[1,2]) / sum(pred_tb)
f_up = (pred_tb[1,2]) / (pred_tb[1,2] + pred_tb[2,2])
f_down = (pred_tb[2,1]) / (pred_tb[1,1] + pred_tb[2,1])
```

```r
# Print the results
cat(paste0("The training error is ", train_er,
       ".\nThe correct up rate is ", 1 - f_up,
       ".\nThe correct down rate is ", 1-f_down))
```

```
## The training error is 0.114644970414201.
## The correct up rate is 0.489866534849234.
## The correct down rate is 0.929177849591937
```

```r
# Overall fraction of correct predictions
pred.corr = mean(qda.class == test$music_genre)
cat(paste0("\nThe Overall fraction of correct predictions is ", pred.corr))
```

```
##
## The Overall fraction of correct predictions is 0.885355029585799
```

kNN

```r
# kNN
library(class)
train = (data1$index %% 2 == 0)
attach(data1)
train.X = cbind(popularity, acousticness, danceability, duration_ms, energy,
                instrumentalness, liveness, loudness, speechiness, valence)[train, ]
test.X = cbind(popularity, acousticness, danceability, duration_ms, energy,
               instrumentalness, liveness, loudness, speechiness, valence)[!train, ]

# k = 1
knn.pred = knn(train.X, test.X, data1$music_genre[train], k = 1)
table(knn.pred, data1$music_genre[!train])
```

```
##
## knn.pred     0     1
##        0 16508  1685
##        1  1749   338
```

```r
mean(knn.pred == data1$music_genre[!train])
```

```
## [1] 0.8306706
```

```r
# k = 10
knn.pred = knn(train.X, test.X, data1$music_genre[train], k = 10)
table(knn.pred, data1$music_genre[!train])
```

```
##
## knn.pred     0     1
##        0 18245  2017
##        1    12     6
```

```r
mean(knn.pred == data1$music_genre[!train])
```

```
## [1] 0.8999507
```

```r
# k = 100
knn.pred = knn(train.X, test.X, data1$music_genre[train], k = 100)
table(knn.pred, data1$music_genre[!train])
```

```
##
## knn.pred     0     1
##        0 18257  2023
##        1     0     0
```

```
mean(knn.pred == data1$music_genre[!train])
```

```
## [1] 0.9002465
```

So far, all results are generally similar. All models predict poorly in classical songs. The highest LDA model only does a little better than taking a random guess which also has 50% accuracy.

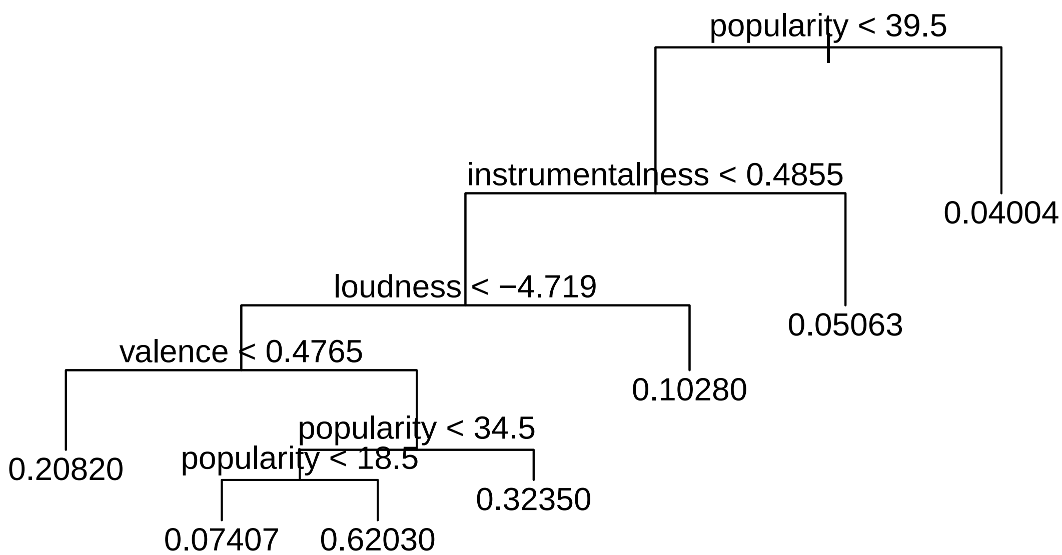
Classification Tree

```
library(tree)
data1 = data1 [, -13]

# Build the classification tree model
tree.data = tree(music_genre~., data1)
summary(tree.data)
```

```
##
## Regression tree:
## tree(formula = music_genre ~ ., data = data1)
## Variables actually used in tree construction:
## [1] "popularity"       "instrumentalness" "loudness"         "valence"
## Number of terminal nodes:  7
## Residual mean deviance:  0.07111 = 2884 / 40550
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.62030 -0.05063 -0.04004  0.00000 -0.04004  0.96000
```

```
# Plot the classification tree model
plot(tree.data)
text(tree.data, pretty=0)
```



From the diagram above, we see that the tree Model has the best training error rate. However, since LDA has the highest correct up rate in predicting classical songs, we select our best performing model as LDA. We also notice that popularity is essential in classifying, which motivated us to predict music popularity in the next section.

2.3. Predicting Music Popularity

We will first presenting a K-means model that takes the ten values of music_genre as 1 to 10. We hope to find some patterns using the unsupervised machine learning model.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:lessR':
##
##     recode, rename

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(cluster)

# Principal Component Analysis
pcclust <- prcomp(data, scale = FALSE)
summary(pcclust)
```

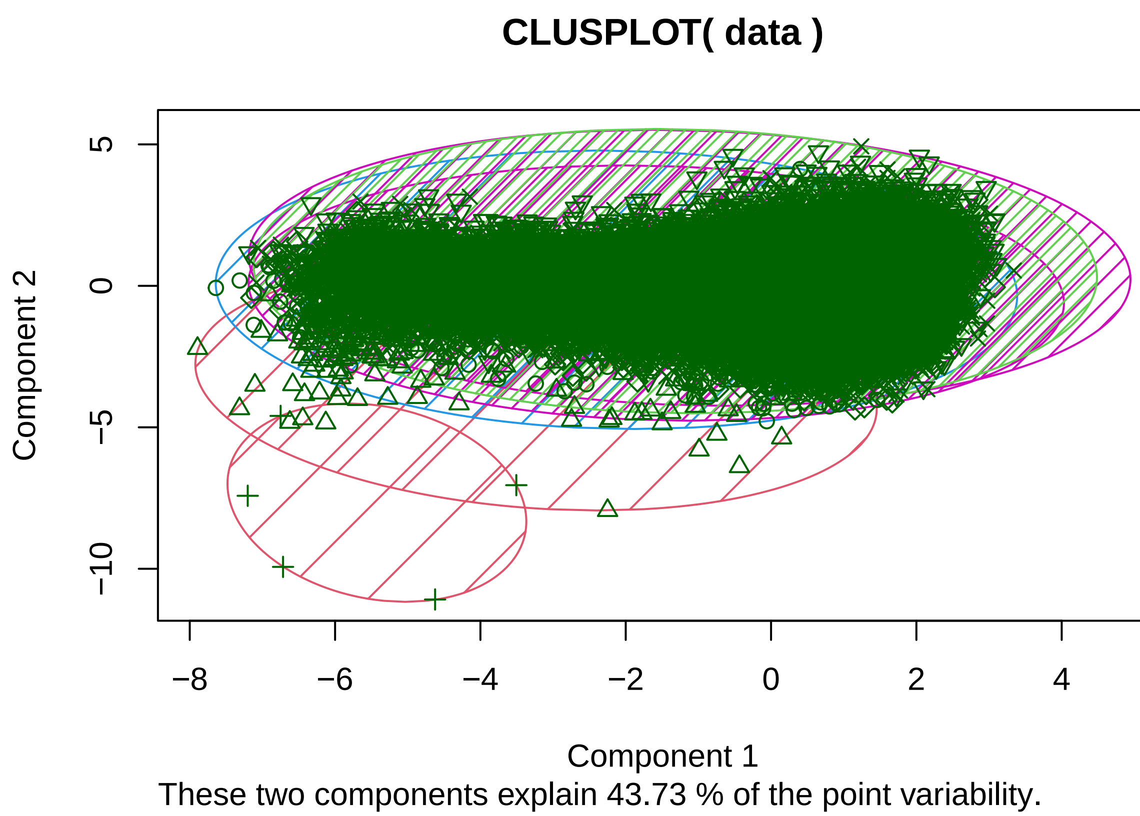
```
## Importance of components:
##                            PC1   PC2   PC3   PC4   PC5    PC6    PC7    PC8
## Standard deviation     110121 30.66 15.57 5.693 2.631 0.2696 0.2523 0.2279
## Proportion of Variance      1  0.00  0.00 0.000 0.000 0.0000 0.0000 0.0000
## Cumulative Proportion       1  1.00  1.00 1.000 1.000 1.0000 1.0000 1.0000
##                            PC9  PC10   PC11    PC12
## Standard deviation     0.1651 0.144 0.1045 0.08873
## Proportion of Variance 0.0000 0.000 0.0000 0.00000
## Cumulative Proportion  1.0000 1.000 1.0000 1.00000
```

```r
pcclust$rotation
```

```
##                                 PC1            PC2            PC3
## popularity        0.00001564965103 -0.02016729710  0.99055474972
## acousticness     -0.00000026371208  0.00265612226 -0.00663902745
## danceability      0.00000033040570  0.00010794740  0.00404356493
## duration_ms      -0.99999999975989 -0.00001369405  0.00001619356
## energy            0.00000022680866 -0.00232835714  0.00396987092
## instrumentalness -0.00000050062644  0.00136100648 -0.00757637438
## liveness         -0.00000004116043 -0.00021525820 -0.00034382314
## loudness          0.00000775548792 -0.04952842175  0.13392626149
## speechiness       0.00000010146815 -0.00020108123  0.00139313300
## tempo             0.00001298909122 -0.99851863802 -0.02665161497
## valence           0.00000039487229 -0.00081664420  0.00191242401
## music_genre      -0.00000240582880  0.00925478286  0.00392818301
##                               PC4             PC5              PC6
## popularity        0.131545044945 -0.032486087649  0.0048601065186
## acousticness      0.037191364436 -0.004695996673 -0.0554681801722
## danceability     -0.008391763845  0.006066193819 -0.1170610416856
## duration_ms      -0.000005361021 -0.000001230707 -0.0000003327843
## energy           -0.035047922133  0.007460954667  0.0432123856710
## instrumentalness  0.022879386337  0.005961901814  0.9414897860603
## liveness         -0.004096099603 -0.000129718518 -0.0448928770052
## loudness         -0.965155024289  0.211231331419  0.0217828950789
## speechiness      -0.001113738974  0.003091606514 -0.0411986858376
## tempo             0.047416953723 -0.000784252418 -0.0000040660275
## valence          -0.012576368527 -0.006588379912 -0.3009869508084
## music_genre       0.213404982990  0.976791851351 -0.0120707965417
##                              PC7             PC8             PC9
## popularity       -0.0010331492561  0.0011377310002  0.002368554537
## acousticness     -0.7417308047476  0.5560882293299  0.163912233204
## danceability      0.2042259608582  0.2475043291055 -0.398857845762
## duration_ms       0.0000000841217  0.0000002385965 -0.000000114334
## energy            0.3634017810441 -0.1033153232944  0.274985922786
## instrumentalness  0.1305630764492  0.2950667152246  0.028867823423
## liveness          0.0420625582788 -0.0732659050435  0.849435066446
```

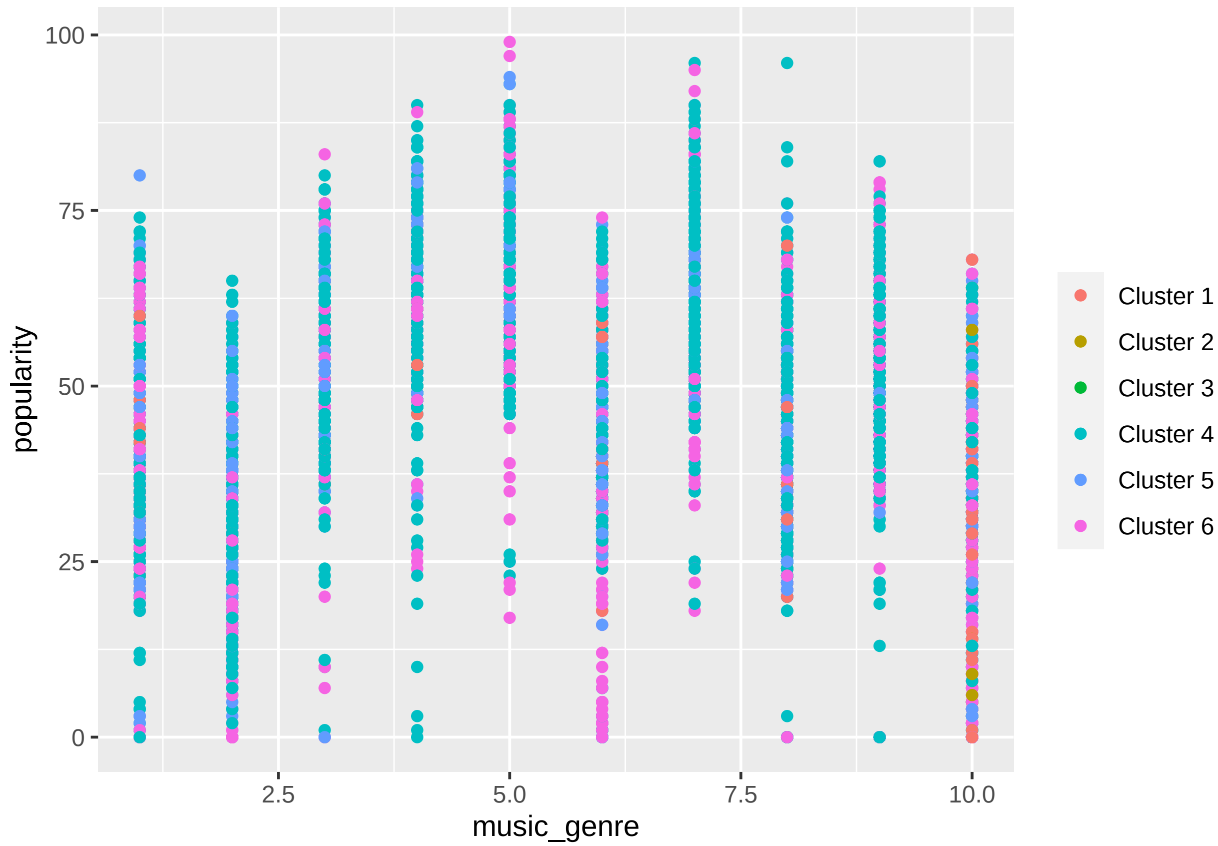
```
## loudness          -0.0463308943289  0.0212098492225 -0.003849756406
## speechiness        0.0266765988820 -0.0221243851747  0.022723879308
## tempo             -0.0006829344495  0.0005063246454 -0.000329633925
## valence            0.5043476989388  0.7248861977177  0.124539203320
## music_genre        0.0049002116501  0.0005257474648  0.002780364250
##                               PC10             PC11              PC12
## popularity         0.00180596070035  0.0003019163850  0.00042164824858
## acousticness       0.05659300863667 -0.2981438866638  0.13079918905593
## danceability      -0.74012388926666 -0.2321546285963  0.34991394636355
## duration_ms       -0.00000009174196 -0.0000000420745 -0.00000002737618
## energy             0.31659374351425 -0.7362367521574  0.36905170915297
## instrumentalness -0.06502252921841  0.0343595702176 -0.05124231062438
## liveness          -0.48521420079136  0.1711218606577  0.06763929761385
## loudness          -0.00349250543030  0.0165364352415 -0.00981935218943
## speechiness       -0.24217893847421 -0.4980510387635 -0.83058227111752
## tempo             -0.00059536530389  0.0000704942580  0.00021761969790
## valence            0.22454509994072  0.1906718546011 -0.16458458949708
## music_genre        0.00587859050663  0.0047414781113 -0.00038504075873
```

```r
# Create clustering and show plot
k6 <- kmeans(data,6, iter.max=100, nstart = 50, algorithm = "Lloyd")
clusplot(data, k6$cluster, color=TRUE, shade=TRUE, labels=0, lines=0)
```




```r
# Plot two most important variables
ggplot(data, aes(x = music_genre, y = popularity)) +
  geom_point(stat = "identity", aes(color = as.factor(k6$cluster))) +
  scale_color_discrete(name = " ",
```

```
                        breaks=c("1", "2", "3", "4", "5","6"),
                        labels=c("Cluster 1", "Cluster 2", "Cluster 3",
                                 "Cluster 4", "Cluster 5","Cluster 6"))
```



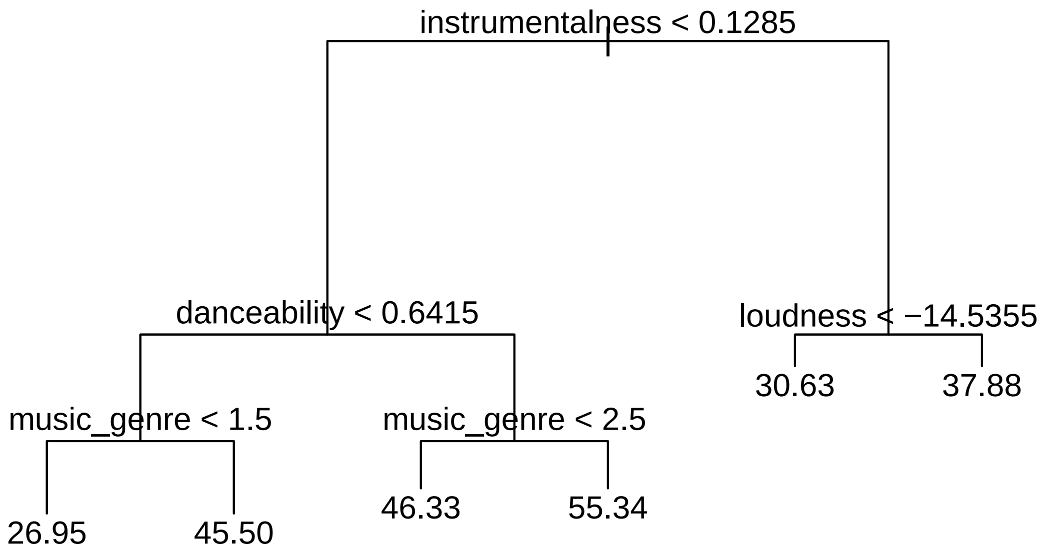
While from PCA, we get that popularity and music_genre are the most important variables, we do not see much distinction and/or clear divisions from the K-means model. Therefore, we would move on and encode music_genre as 1,2,3, illustrated above, hoping to find more patterns.

Regression Tree

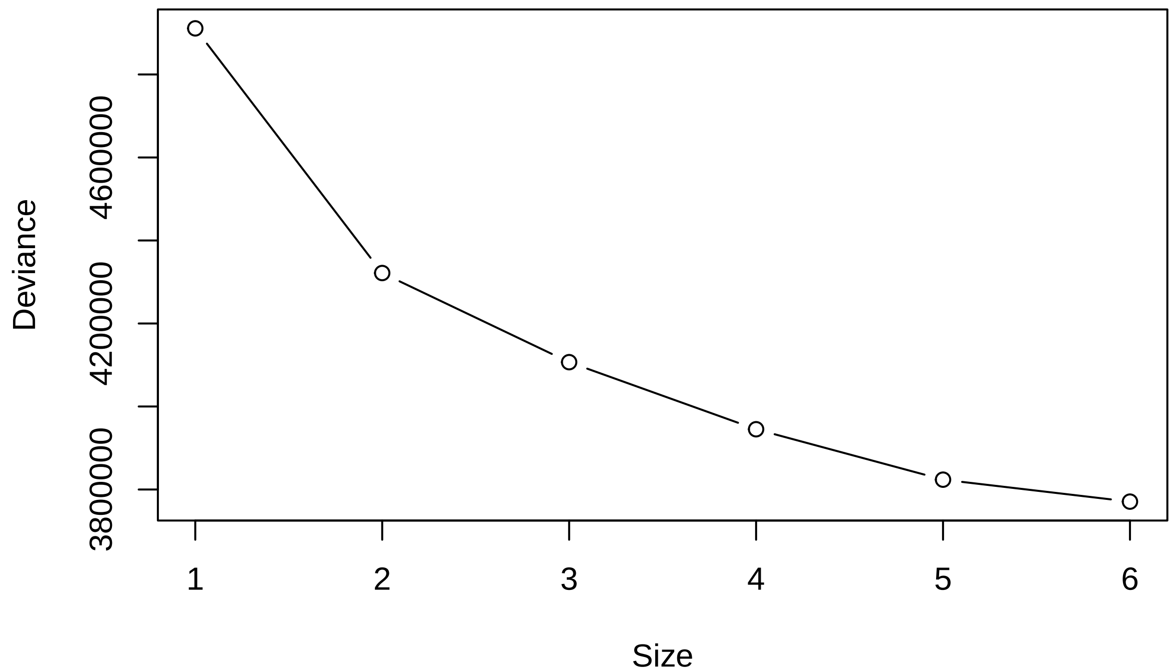
```
# Split the data into train and test sets
train = sample(1:nrow(data2), nrow(data2)/2)
tree.data2 = tree(popularity~., data2, subset=train)
summary(tree.data2)
```

```
##
## Regression tree:
## tree(formula = popularity ~ ., data = data2, subset = train)
## Variables actually used in tree construction:
## [1] "instrumentalness" "danceability"     "music_genre"      "loudness"
## Number of terminal nodes:  6
## Residual mean deviance:  185.4 = 3759000 / 20270
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -55.3400  -8.3320   0.5024   0.0000   9.5020  44.5000
```

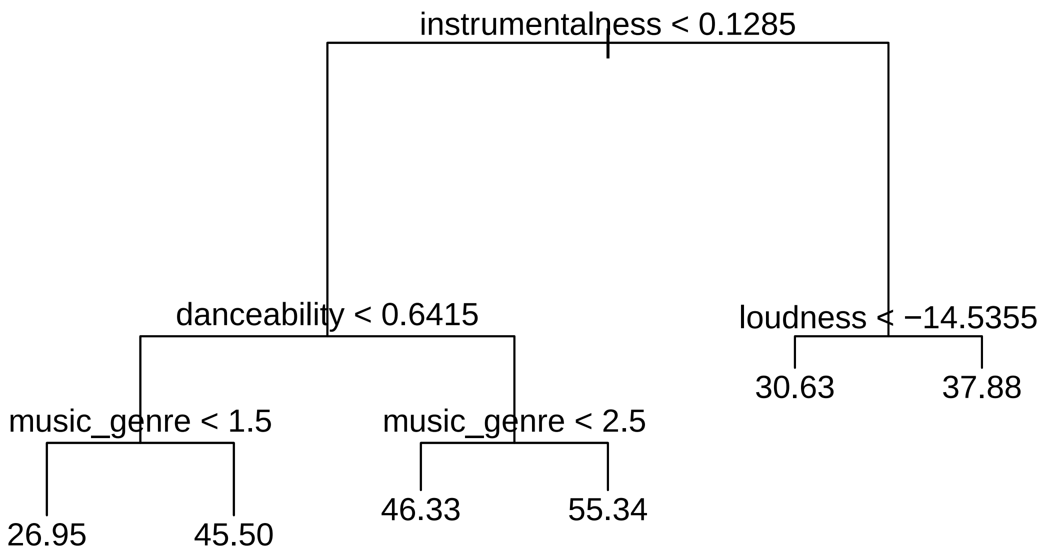
```r
# Plot the regression tree model
plot(tree.data2)
text(tree.data2, pretty=0)
```




```r
# Use Cross-Validation to choose tree complexity
cv.data2 = cv.tree(tree.data2)
plot(cv.data2$size, cv.data2$dev, xlab="Size", ylab="Deviance", type="b")
```

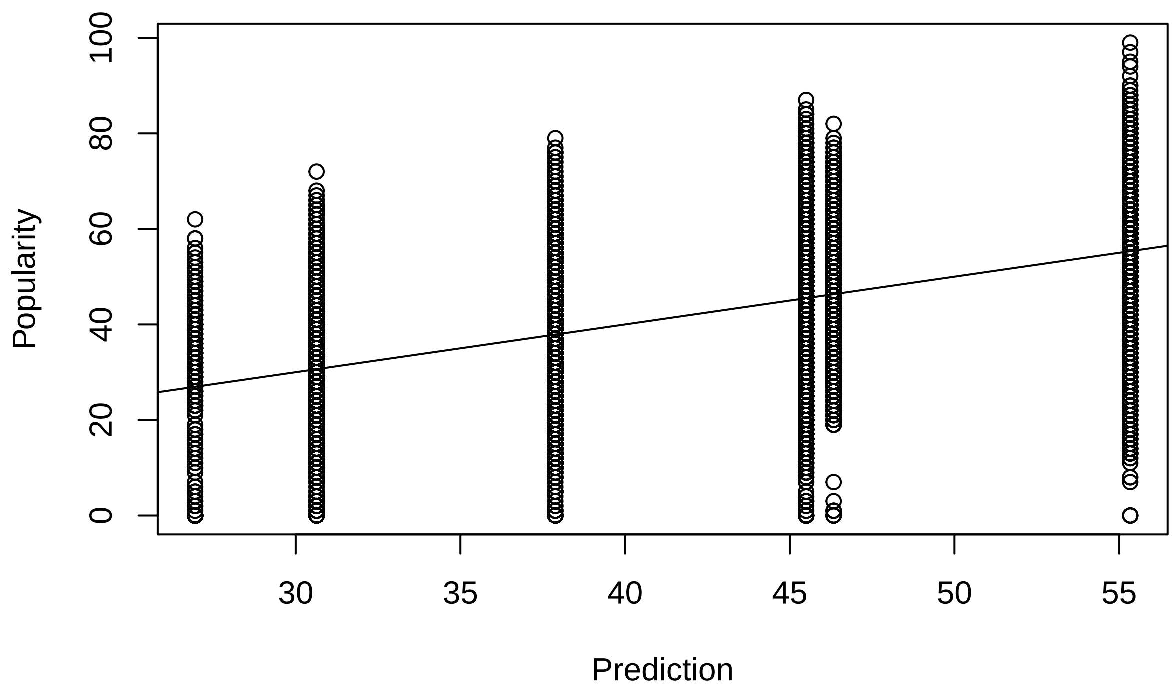



```
# Build and show the pruned tree model
prune.data2 = prune.tree(tree.data2, best=6)
plot(prune.data2)
text(prune.data2, pretty=0)
```




```
# Predict on the test set
yhat=predict(tree.data2, newdata=data2[-train, ])
data2.test=data2[-train, "popularity"]

# Plot the prediction results
plot(yhat, data2.test$popularity, xlab="Prediction", ylab="Popularity")
abline(0,1)
```




```
mse.rt = mean((yhat-data2.test$popularity)^2) #MSE
mse.rt
```

```
## [1] 184.3637
```

In the Regression Tree, we see that instrumentalness is important to classify popularity score. CV plot shows that deviance decreases as size increases, and the best size is 6. MSE is about 180.

Random forest

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

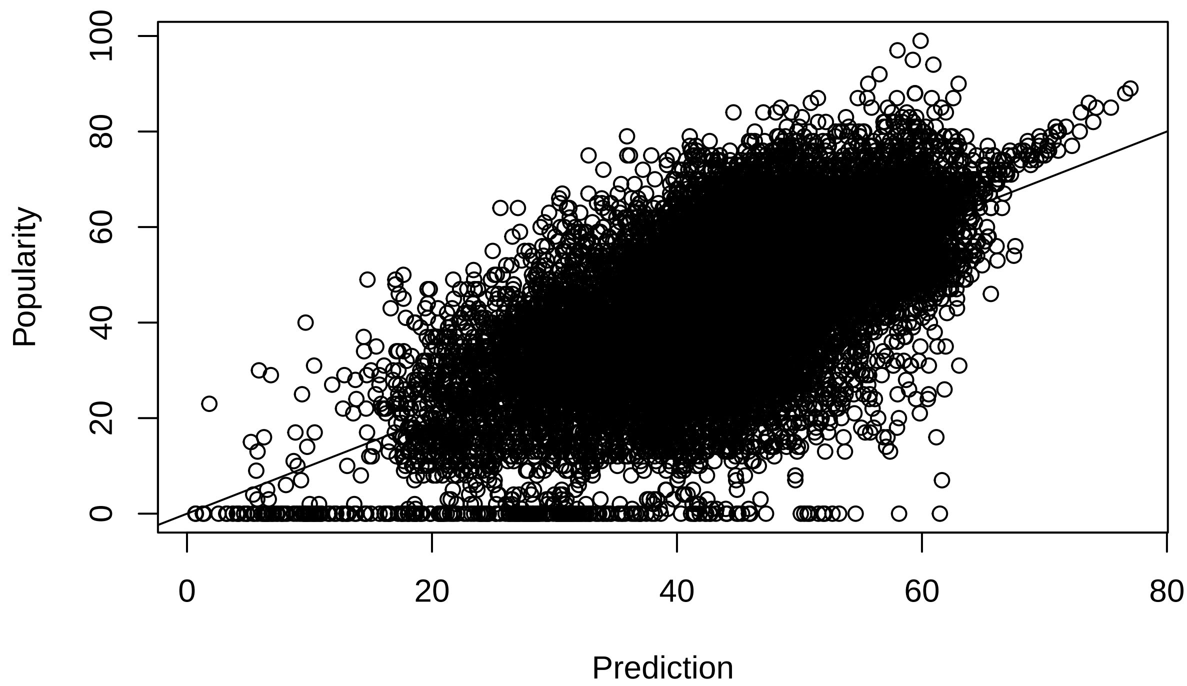
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
# Build the model (Bagging)
bag.data = randomForest(popularity~., data=data2, subset=train, mtry=10, importance =TRUE)
bag.data
```

```
##
## Call:
##  randomForest(formula = popularity ~ ., data = data2, mtry = 10,      importance = TRUE, subset = tra
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 143.1208
##                     % Var explained: 40.89
```

```
# Predict on test set
yhat.bag = predict(bag.data, newdata=data2[-train, ])

# Plot the prediction
plot(yhat.bag, data2[-train, ]$popularity, xlab="Prediction", ylab="Popularity")
abline(0,1)
```




```
mse.rf11 = mean((yhat.bag-data2[-train, ])$popularity^2) #MSE
mse.rf11
```

```
## [1] 141.1258
```

```
# Use mtry=6 to compare models
rf.data = randomForest(popularity~.,data=data2, subset=train, mtry=6, importance=TRUE)
yhat.rf = predict(rf.data, newdata=data2[-train, ])
```

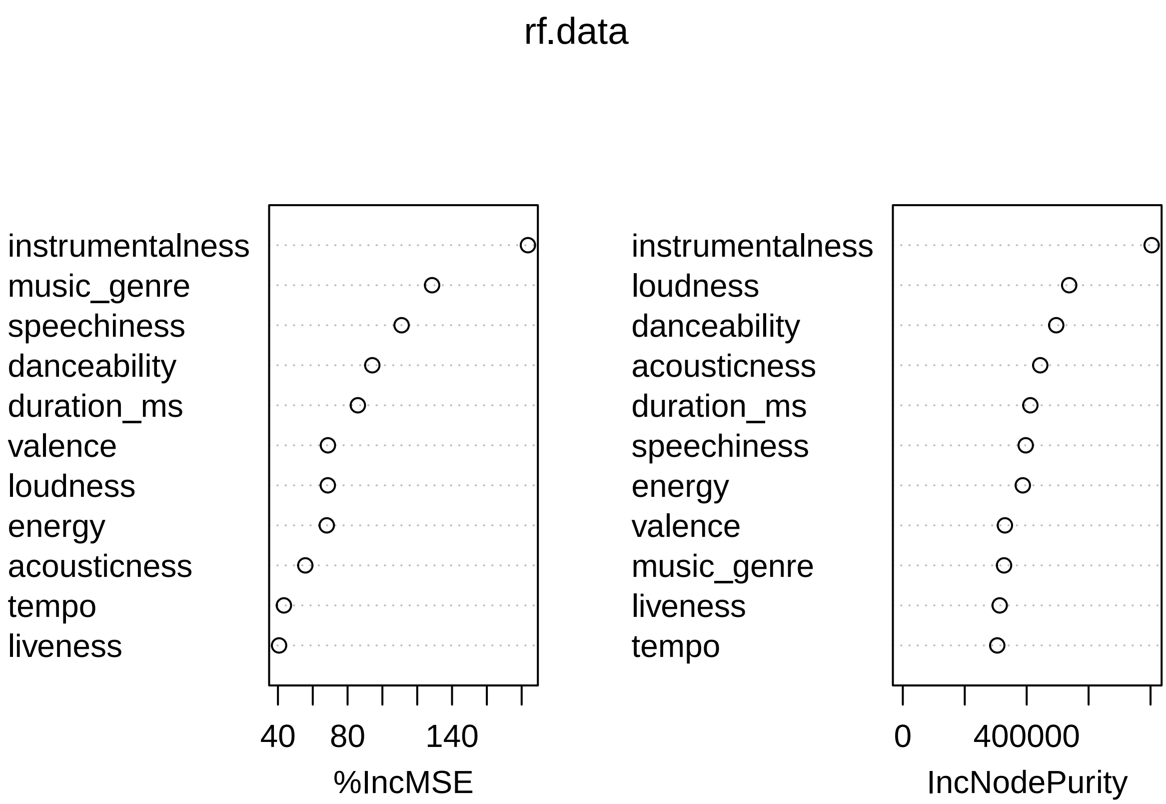
```r
# Measure the model performance
mse.rf6 = mean((yhat.rf-data2[-train, ]$popularity)^2)  #MSE
mse.rf6
```

```
## [1] 140.3113
```

```r
importance(rf.data)
```

```
##                     %IncMSE IncNodePurity
## acousticness       55.72038      443850.4
## danceability       94.19603      495426.4
## duration_ms        85.88779      411848.5
## energy             68.03217      387474.7
## instrumentalness 183.58840      803583.3
## liveness           40.64691      312845.4
## loudness           68.63054      537324.6
## speechiness       111.04120      396903.7
## tempo              43.42336      304980.0
## valence            68.70298      329700.9
## music_genre       128.53092      326908.9
```

```r
varImpPlot(rf.data)
```



From the random forest, we see that our model explain about half of the data set, which is not ideal. However, the MSE is much lower than regression tree. The most important variables are instrumentalness, music_genre, and speechiness. The errors do not show specific patterns. In general, random forest yields similar results with regression tree in terms of importance of variables and MSE, and we choose random forest over regression tree.

GAM

```
library(gam)
```

```
## Loading required package: splines

## Loading required package: foreach

## Loaded gam 1.22-2
```

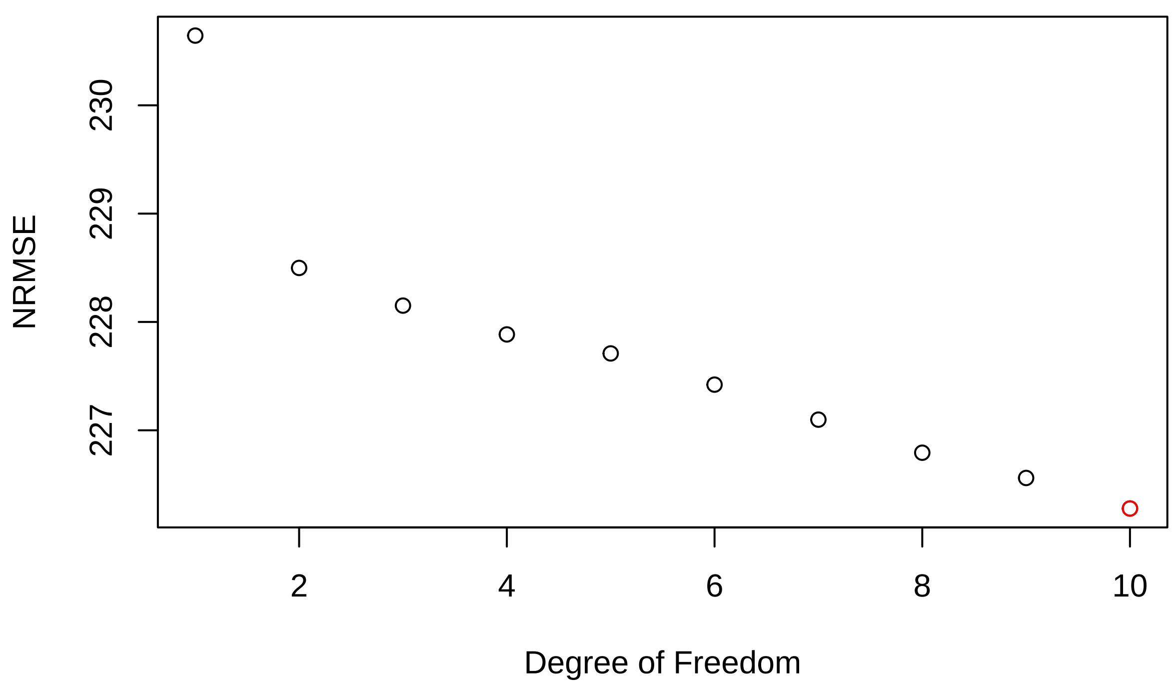
```
library(foreach)
library(boot)
set.seed(12345)

errors = rep(NA,10)
for (i in 1:10){
  models = gam(popularity ~ s(speechiness,i), data = data2[train,])
    errors[i] = cv.glm(data[train,], models, K=10)$delta[1]
}

# Plot the cv errors
min.pt = which.min(errors)
plot(errors, xlab = "Degree of Freedom", ylab = "NRMSE",
     main = cat("Model Choice = ", min.pt))
```

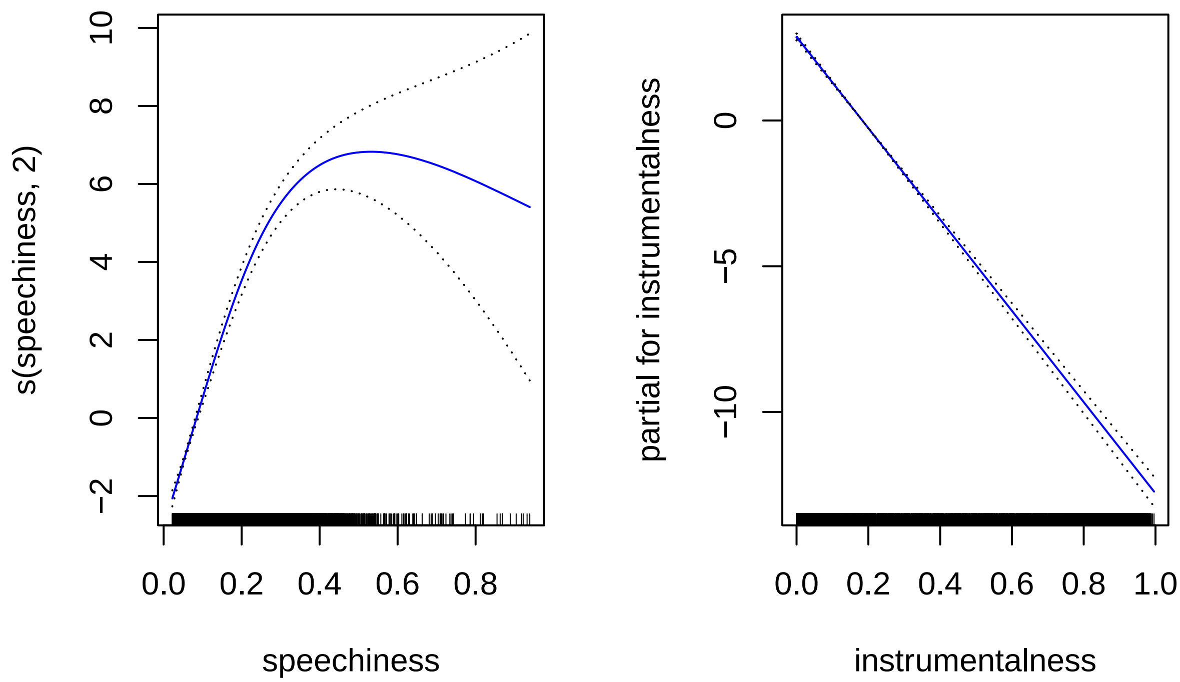
```
## Model Choice =  10
```

```
points(x = min.pt, y = errors[min.pt], col = "red")
```

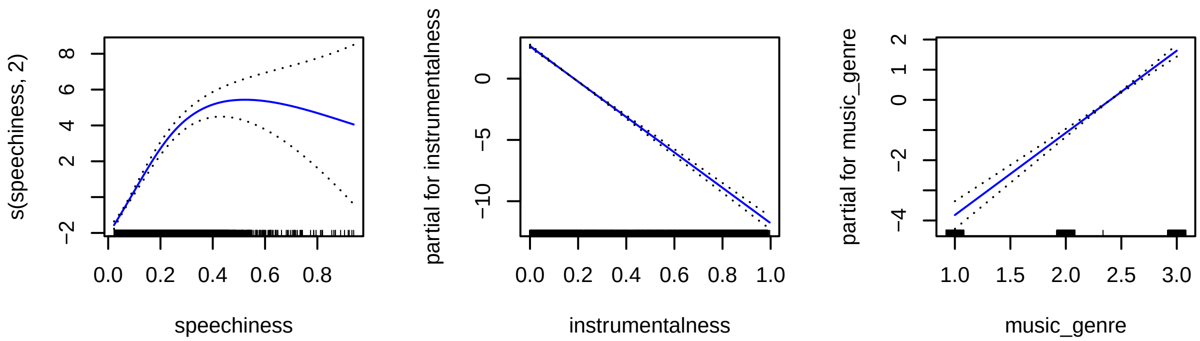


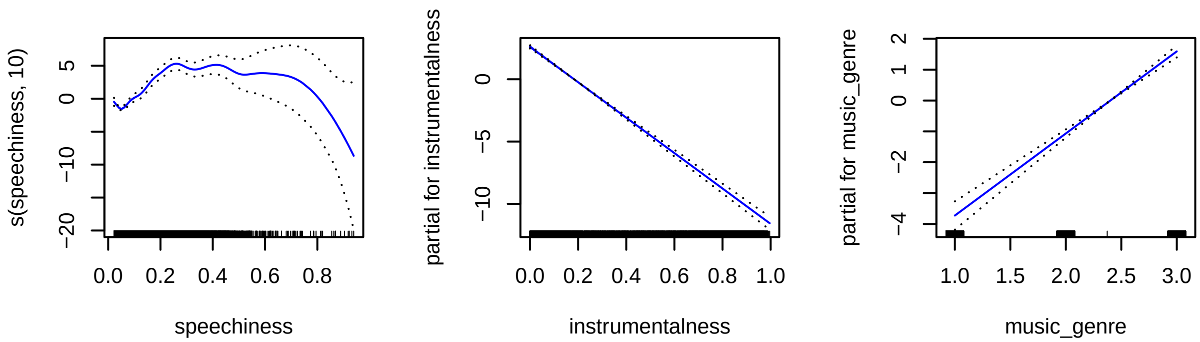
```r
# Since the most significant drop appears in 2 and rmse is decreasing
# choose degree = 2 for simplicity
gam1 <- gam(popularity ~ s(speechiness,2) + instrumentalness, data = data2[train,])
gam2 <- gam(popularity ~ s(speechiness,2) + instrumentalness + music_genre, data = data2[train,])
gam3 <- gam(popularity ~ s(speechiness,10) + instrumentalness + music_genre, data = data2[train,])
sum1 = summary(gam1)
sum2 = summary(gam2)
sum3 = summary(gam3)

# Plot the model
par(mfrow = c(1, 2))
plot(gam1, se = TRUE, col = "blue")
```




```r
par(mfrow = c(2, 3))
plot(gam2, se = TRUE, col = "blue")
plot(gam3, se = TRUE, col = "blue")
```






```r
# Plot models' fitting values
# Graph is omitted because spots overshadow each other and in extreme large size,
# which does not provide useful information for model analysis

#par(mfrow = c(1, 1))
#plot(data2[-train,]$speechiness, data2[-train,]$popularity, cex = .5, col = "gray",
#     xlab = "speechiness", ylab = "popularity", main ="Gam Fitting on Popularity")
#plot(gam1$fitted.values, col = "pink", lwd = 2, cex=0.65)
#points(gam2$fitted.values, col = "lightblue", lwd = 2, cex=0.65)
#points(gam3$fitted.values, col = "lightgreen", lwd = 2, cex=0.65)
#legend("topright", c("gam1", "gam2", "gam3"), fill = c("red","blue","green"), cex = 0.8)

# Prediction using gam
gam1.pred = predict(gam1, newdata = data2[-train,])
gam2.pred = predict(gam2, newdata = data2[-train,])
gam3.pred = predict(gam3, newdata = data2[-train,])

# Plot models' fitting values
# Graph is omitted because spots overshadow each other and in extreme large size,
# which does not provide useful information for model analysis

#par(mfrow = c(1, 1))
#plot(data2[-train,]$speechiness, data2[-train,]$popularity, cex = .5, col = "gray",
#     xlab = "speechiness", ylab = "popularity", main ="Gam Forecast on Popularity")
#lines(gam1.pred, col = "red", lwd = 2)
#lines(gam2.pred, col = "blue", lwd = 2)
```

```r
#lines(gam3.pred, col = "green", lwd = 2)
#legend("topright", c("gam1", "gam2", "gam3"), fill = c("red","blue","green"), cex = 0.8)

# Measure the model performance
mse.gm1 = mean((data2[-train,]$popularity - gam1.pred)^2)
mse.gm2 = mean((data2[-train,]$popularity - gam2.pred)^2)
mse.gm3 = mean((data2[-train,]$popularity - gam3.pred)^2)

# Compare the all gam models
anova(gam1, gam2, gam3, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: popularity ~ s(speechiness, 2) + instrumentalness
## Model 2: popularity ~ s(speechiness, 2) + instrumentalness + music_genre
## Model 3: popularity ~ s(speechiness, 10) + instrumentalness + music_genre
##   Resid. Df Resid. Dev     Df Deviance        F                Pr(>F)
## 1     20276    4132463
## 2     20275    4080027 1.0000    52435 261.2377 < 0.00000000000000022 ***
## 3     20267    4067979 7.9997    12048   7.5034       0.0000000004777 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
model.name = c("gam1", "gam2", "gam3")
MSE = c(mse.gm1, mse.gm2, mse.gm3)
data.frame(model.name, MSE)
```

```
##   model.name      MSE
## 1       gam1 202.0068
## 2       gam2 199.8754
## 3       gam3 199.3717
```

```r
# Compare all models in predicting popularity
model.name = c("Regression Tree", "Random Forecast", "Gam")
MSE = c(mse.rt, mse.rf6, mse.gm3)
data.frame(model.name, MSE)
```

```
##        model.name      MSE
## 1 Regression Tree 184.3637
## 2 Random Forecast 140.3113
## 3             Gam 199.3717
```

Based on gam models' MSE, we can see that gam3 has best performing (lowest RMSE) while the difference among their RMSE is minor.

Generally, the GAM models show that our prediction is about 14 (= sqrt(MSE)) score deviant to the real popularity score, which is not bad. However, in terms of MSE, we would still suggest random forest as our best-fit model.

3. Conclusion and Further Work

In concluion, for predicting music genre, we select our LDA as the best-fitting model. Even though tree models have straightforward presentation, their accuracy is not well enough to be considered. For predicting popularity, random forest is the best in terms of MSE.

For future work, we would suggest to use more data, specifically data on years of release, to better evaluate our models for predicting popularity. We also note that the models may cause more confusion if one song has more than one genre, and this is a problem to be solved, maybe using a K-means clustering.

4. References

Vicsuperman. (2021, November 2). *Prediction of music genre.* Kaggle. https://www.kaggle.com/vicsuperman/prediction-of-music-genre