DAT402-Project2

Alexander Coover

2024-04-16

Dataset: The dataset I selected contains 114 different genres of music, with 1000 songs per genre from Spotify. Each row contains various different metrics and information about the song. Some of the metrics include popularity from 0-100, duration of the song, danceability, energy, key, etc. For the exact features used, refer to the variable importance plot.

Project Summary: For my first project in this class, I used Naive Bayes to try my best and make predictions of what genre a song is based on the features of my dataset. I created a valid model, but I know a lot more machine learning practices now, so I decided to use a random forest to make predictions this time. In addition to cleaning up my Project 1 a bit, I also created code for hierarchical clustering from scratch so that I could combine the 114 different genres present in the dataset into a much smaller pool to select from. After narrowing the pool down to the final genre groups, I created naive bayes and random forest models, then showed how successful each one was using both overall accuracy and accuracy per genre.

Conclusion: The hierarchical clustering was incredibly effective in combining similar genres and was key to the success of the project. I have tested the entire project with 20 final genre groups and got an overall accuracy of 58.7% with naive bayes and 81.7% with random forest. I also tested with 10 final genre groups and got 77% accuracy with naive bayes and 88% accuracy with random forest. I decided to create the report with 13 final groups mainly because the graphs aren't as cluttered as they are with 20, and 10 final groups seemed to be forcing genre combinations that didn't make as much sense.

Possible Areas for further improvement of the project:

- Narrowing down features to reduce dimensionality. Time signature, key, and liveness are not good predictors of genre.
- Optimizing the number of final genre groups. I did not do this because of computing limitations and I did not want to wait for it to run.
- Creating more intentional names for the genre groups. Every time 2 are combined, the decision of which genre takes the other's name is decided by alphabetical order. I could probably look into the tree and come up with one of the genres that is within it, not necessarily the current name, and pick that to best suit the genres contained.

```r
library(dplyr)
```

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

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

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.2.3
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
library(rpart)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
## 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
```

```
# read in file
df = read.csv(file = "dataset.csv")
# remove columns that will not be used in the model
df$X = NULL
df$track_id = NULL
df$artists = NULL
df$album_name = NULL
df$track_name = NULL
df$Column1 = NULL
df$mode = NULL
df$explicit = NULL

# filter data to exclude rows with 0 popularity
df = filter(df, popularity > 0)
```

```
# combine genres until there are n genres left using hierarchical clustering
n = 13
while (length(unique(df$track_genre)) > n) {

# create null dataframe of average values for each genre
average_df = data.frame(
  Genre = NULL,
  Avg_popularity = NULL,
  Avg_duration = NULL,
  Avg_danceability = NULL,
  Avg_energy = NULL,
  Avg_key = NULL,
  Avg_loudness = NULL,
  Avg_speechiness = NULL,
  Avg_acousticness = NULL,
  Avg_intrumentalness = NULL,
  Avg_liveness = NULL,
  Avg_valence = NULL,
  Avg_tempo = NULL
)

# fill out the above dataframe
genres = unique(df$track_genre)
for (i in genres) {
  genre_df = filter(df, track_genre == i)
  average_df = rbind(average_df, data.frame(
    Genre = i,
    Avg_popularity = mean(genre_df$popularity),
    Avg_duration = mean(genre_df$duration_ms),
    Avg_danceability = mean(genre_df$danceability),
    Avg_energy = mean(genre_df$energy),
    Avg_key = mean(genre_df$key),
    Avg_loudness = mean(genre_df$loudness),
    Avg_speechiness = mean(genre_df$speechiness),
    Avg_acousticness = mean(genre_df$acousticness),
    Avg_instrumentalness = mean(genre_df$instrumentalness),
    Avg_liveness = mean(genre_df$liveness),
    Avg_valence = mean(genre_df$valence),
    Avg_tempo = mean(genre_df$tempo)
  ))
}

# standardize the averages for each variable so they each contribute equally to the similarity c
alculation
average_df$Avg_popularity = (average_df$Avg_popularity - mean(average_df$Avg_popularity)) / sd(a
verage_df$Avg_popularity)
average_df$Avg_duration = (average_df$Avg_duration - mean(average_df$Avg_duration)) / sd(average
_df$Avg_duration)
average_df$Avg_danceability = (average_df$Avg_danceability - mean(average_df$Avg_danceability))
/ sd(average_df$Avg_danceability)
average_df$Avg_energy = (average_df$Avg_energy - mean(average_df$Avg_energy)) / sd(average_df$Av
g_energy)
```

```
average_df$Avg_key = (average_df$Avg_key - mean(average_df$Avg_key)) / sd(average_df$Avg_key)
average_df$Avg_loudness = (average_df$Avg_loudness - mean(average_df$Avg_loudness)) / sd(average
_df$Avg_loudness)
average_df$Avg_speechiness = (average_df$Avg_speechiness - mean(average_df$Avg_speechiness)) / s
d(average_df$Avg_speechiness)
average_df$Avg_acousticness = (average_df$Avg_acousticness - mean(average_df$Avg_acousticness))
/ sd(average_df$Avg_acousticness)
average_df$Avg_instrumentalness = (average_df$Avg_instrumentalness - mean(average_df$Avg_instrum
entalness)) / sd(average_df$Avg_instrumentalness)
average_df$Avg_liveness = (average_df$Avg_liveness - mean(average_df$Avg_liveness)) / sd(average
_df$Avg_liveness)
average_df$Avg_valence = (average_df$Avg_valence - mean(average_df$Avg_valence)) / sd(average_df
$Avg_valence)
average_df$Avg_tempo = (average_df$Avg_tempo - mean(average_df$Avg_tempo)) / sd(average_df$Avg_t
empo)

# create a dataframe showing the similarity between genres using euclidean distance
numeric_cols = average_df[, -1]
distance_matrix = dist(numeric_cols, method = "euclidean")
similarity_df = as.data.frame(as.matrix(distance_matrix))
row.names(similarity_df) = average_df$Genre
colnames(similarity_df) = average_df$Genre


find_most_similar = function(similarity_df) {
  most_similar = Inf
  genre1 = NULL
  genre2 = NULL
  for (i in rownames(similarity_df)) {
    for (j in colnames(similarity_df)) {
      if (similarity_df[i,j] < most_similar && similarity_df[i,j] > 0) {
        most_similar = similarity_df[i,j]
        genre1 = i
        genre2 = j
      }
    }
  }
  return(c(genre1, genre2, most_similar))
}

find_most_similar(similarity_df)
new_broader_genre = find_most_similar(similarity_df)[1]
genre_to_replace = find_most_similar(similarity_df)[2]
similarity = find_most_similar(similarity_df)[3]
print(paste("Replacing occurances of ", genre_to_replace, " with ", new_broader_genre, ". Simila
rity: ", similarity))
df$track_genre = replace(df$track_genre, df$track_genre == genre_to_replace, new_broader_genre)
}
```

```
## [1] "Replacing occurances of  indie  with  indie-pop . Similarity:  0.295150771432879"
## [1] "Replacing occurances of  reggaeton  with  latino . Similarity:  0.360386230386269"
## [1] "Replacing occurances of  punk  with  punk-rock . Similarity:  0.43494546571906"
## [1] "Replacing occurances of  singer-songwriter  with  acoustic . Similarity:  0.604014364603
56"
## [1] "Replacing occurances of  songwriter  with  acoustic . Similarity:  0.335098822044035"
## [1] "Replacing occurances of  indian  with  folk . Similarity:  0.743215567519213"
## [1] "Replacing occurances of  swedish  with  country . Similarity:  0.765475151767245"
## [1] "Replacing occurances of  reggae  with  latino . Similarity:  0.775675241199158"
## [1] "Replacing occurances of  alternative  with  alt-rock . Similarity:  0.811111820726925"
## [1] "Replacing occurances of  soul  with  indie-pop . Similarity:  0.865612965728785"
## [1] "Replacing occurances of  spanish  with  j-pop . Similarity:  0.877638594655847"
## [1] "Replacing occurances of  turkish  with  french . Similarity:  0.87842358746309"
## [1] "Replacing occurances of  sad  with  chill . Similarity:  0.889970768446573"
## [1] "Replacing occurances of  electro  with  edm . Similarity:  0.917916841437151"
## [1] "Replacing occurances of  pop  with  indie-pop . Similarity:  0.948861758846107"
## [1] "Replacing occurances of  metal  with  hard-rock . Similarity:  0.962560321304039"
## [1] "Replacing occurances of  house  with  edm . Similarity:  0.965421874764299"
## [1] "Replacing occurances of  j-pop  with  country . Similarity:  0.96864422377523"
## [1] "Replacing occurances of  country  with  blues . Similarity:  1.03072775574547"
## [1] "Replacing occurances of  rock  with  blues . Similarity:  1.03613725852686"
## [1] "Replacing occurances of  dubstep  with  dub . Similarity:  1.09084827733249"
## [1] "Replacing occurances of  pop-film  with  k-pop . Similarity:  1.14570448768527"
## [1] "Replacing occurances of  groove  with  alt-rock . Similarity:  1.15285509954936"
## [1] "Replacing occurances of  funk  with  dancehall . Similarity:  1.16864705534779"
## [1] "Replacing occurances of  jazz  with  acoustic . Similarity:  1.18623141198232"
## [1] "Replacing occurances of  electronic  with  anime . Similarity:  1.19195285443708"
## [1] "Replacing occurances of  hip-hop  with  dance . Similarity:  1.19116918947892"
## [1] "Replacing occurances of  hard-rock  with  alt-rock . Similarity:  1.19298084036037"
## [1] "Replacing occurances of  j-rock  with  alt-rock . Similarity:  1.13543041688193"
## [1] "Replacing occurances of  psych-rock  with  british . Similarity:  1.21479666634781"
## [1] "Replacing occurances of  sertanejo  with  pagode . Similarity:  1.21514096266077"
## [1] "Replacing occurances of  hardstyle  with  happy . Similarity:  1.23222551231388"
## [1] "Replacing occurances of  mandopop  with  cantopop . Similarity:  1.23499212764434"
## [1] "Replacing occurances of  cantopop  with  acoustic . Similarity:  1.1942025420296"
## [1] "Replacing occurances of  folk  with  acoustic . Similarity:  1.25598147532823"
## [1] "Replacing occurances of  ska  with  party . Similarity:  1.25962449833838"
## [1] "Replacing occurances of  garage  with  alt-rock . Similarity:  1.28460386824837"
## [1] "Replacing occurances of  industrial  with  heavy-metal . Similarity:  1.29055540273386"
## [1] "Replacing occurances of  dancehall  with  dance . Similarity:  1.2887012055146"
## [1] "Replacing occurances of  k-pop  with  indie-pop . Similarity:  1.28392163237837"
## [1] "Replacing occurances of  emo  with  anime . Similarity:  1.28757145303707"
## [1] "Replacing occurances of  french  with  anime . Similarity:  1.2306153285303"
## [1] "Replacing occurances of  blues  with  anime . Similarity:  1.12085736781412"
## [1] "Replacing occurances of  world-music  with  gospel . Similarity:  1.269588320583"
## [1] "Replacing occurances of  edm  with  deep-house . Similarity:  1.27840648211938"
## [1] "Replacing occurances of  indie-pop  with  anime . Similarity:  1.29252258217724"
## [1] "Replacing occurances of  british  with  acoustic . Similarity:  1.28609482636226"
## [1] "Replacing occurances of  progressive-house  with  deep-house . Similarity:  1.2921860204
1904"
## [1] "Replacing occurances of  r-n-b  with  mpb . Similarity:  1.28643152762875"
## [1] "Replacing occurances of  detroit-techno  with  chicago-house . Similarity:  1.2974323337
```

```
0311"
## [1] "Replacing occurances of  dub  with  alt-rock . Similarity:  1.32310057101714"
## [1] "Replacing occurances of  grunge  with  alt-rock . Similarity:  1.26438667296365"
## [1] "Replacing occurances of  hardcore  with  alt-rock . Similarity:  1.28097555856973"
## [1] "Replacing occurances of  tango  with  honky-tonk . Similarity:  1.31016271615727"
## [1] "Replacing occurances of  heavy-metal  with  goth . Similarity:  1.37078670025539"
## [1] "Replacing occurances of  samba  with  pagode . Similarity:  1.38041306924842"
## [1] "Replacing occurances of  mpb  with  brazil . Similarity:  1.435488836783"
## [1] "Replacing occurances of  piano  with  ambient . Similarity:  1.47860905859283"
## [1] "Replacing occurances of  deep-house  with  anime . Similarity:  1.51101696401501"
## [1] "Replacing occurances of  anime  with  alt-rock . Similarity:  1.39400297166766"
## [1] "Replacing occurances of  latino  with  dance . Similarity:  1.51886403191412"
## [1] "Replacing occurances of  disco  with  dance . Similarity:  1.46673409123026"
## [1] "Replacing occurances of  punk-rock  with  power-pop . Similarity:  1.5606225675711"
## [1] "Replacing occurances of  techno  with  minimal-techno . Similarity:  1.57649652194079"
## [1] "Replacing occurances of  latin  with  afrobeat . Similarity:  1.61357226056799"
## [1] "Replacing occurances of  happy  with  drum-and-bass . Similarity:  1.62466984760941"
## [1] "Replacing occurances of  goth  with  death-metal . Similarity:  1.62476518096323"
## [1] "Replacing occurances of  german  with  acoustic . Similarity:  1.6422050276203"
## [1] "Replacing occurances of  dance  with  alt-rock . Similarity:  1.63527521513946"
## [1] "Replacing occurances of  brazil  with  alt-rock . Similarity:  1.64054838040501"
## [1] "Replacing occurances of  power-pop  with  party . Similarity:  1.67422238427854"
## [1] "Replacing occurances of  party  with  j-idol . Similarity:  1.58169213484495"
## [1] "Replacing occurances of  rockabilly  with  bluegrass . Similarity:  1.70198084068816"
## [1] "Replacing occurances of  malay  with  acoustic . Similarity:  1.71501671738973"
## [1] "Replacing occurances of  guitar  with  classical . Similarity:  1.72324784419688"
## [1] "Replacing occurances of  romance  with  honky-tonk . Similarity:  1.75089002905071"
## [1] "Replacing occurances of  rock-n-roll  with  acoustic . Similarity:  1.77700707517512"
## [1] "Replacing occurances of  show-tunes  with  acoustic . Similarity:  1.76985111583938"
## [1] "Replacing occurances of  death-metal  with  club . Similarity:  1.79146059767075"
## [1] "Replacing occurances of  j-idol  with  club . Similarity:  1.72956640165779"
## [1] "Replacing occurances of  kids  with  j-dance . Similarity:  1.78745999979347"
## [1] "Replacing occurances of  metalcore  with  club . Similarity:  1.79147823461514"
## [1] "Replacing occurances of  club  with  alt-rock . Similarity:  1.84752183401179"
## [1] "Replacing occurances of  synth-pop  with  alt-rock . Similarity:  1.75874150805749"
## [1] "Replacing occurances of  j-dance  with  afrobeat . Similarity:  1.83441295035164"
## [1] "Replacing occurances of  trip-hop  with  alt-rock . Similarity:  1.86667403073991"
## [1] "Replacing occurances of  chill  with  acoustic . Similarity:  1.91186995939678"
## [1] "Replacing occurances of  alt-rock  with  acoustic . Similarity:  1.96117669098015"
## [1] "Replacing occurances of  bluegrass  with  acoustic . Similarity:  1.86348467393392"
## [1] "Replacing occurances of  gospel  with  acoustic . Similarity:  1.95098627423886"
## [1] "Replacing occurances of  chicago-house  with  breakbeat . Similarity:  1.95160105737741"
## [1] "Replacing occurances of  trance  with  minimal-techno . Similarity:  1.96929643088776"
## [1] "Replacing occurances of  salsa  with  afrobeat . Similarity:  2.00169783406871"
## [1] "Replacing occurances of  disney  with  classical . Similarity:  2.09745204749532"
## [1] "Replacing occurances of  pagode  with  forro . Similarity:  2.08364478074936"
## [1] "Replacing occurances of  afrobeat  with  acoustic . Similarity:  2.12598189587774"
## [1] "Replacing occurances of  honky-tonk  with  classical . Similarity:  2.2072547119667"
## [1] "Replacing occurances of  forro  with  acoustic . Similarity:  2.22735827471665"
## [1] "Replacing occurances of  opera  with  classical . Similarity:  2.34440089096163"
## [1] "Replacing occurances of  new-age  with  ambient . Similarity:  2.31168633139877"
```

```
## [1] "Replacing occurances of  drum-and-bass  with  black-metal . Similarity:  2.450373509519
6"
```

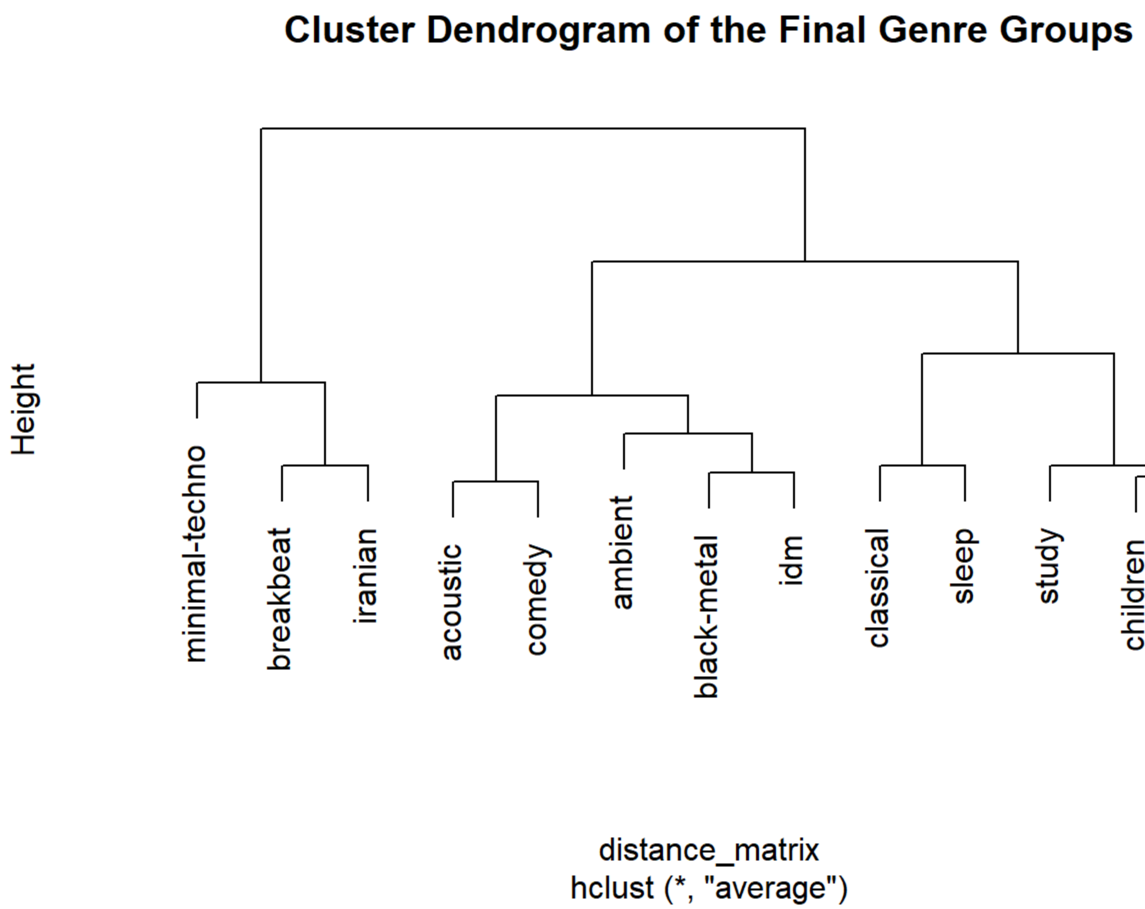
```
average_df
```

```
##             Genre Avg_popularity Avg_duration Avg_danceability Avg_energy
## 1        acoustic     1.25200023   -0.1744996       0.49074975  0.3044457
## 2         ambient     1.23670440    0.1059730      -0.51782859 -1.5339255
## 3     black-metal    -0.38587438    0.9383920      -1.01980250  1.2074061
## 4       breakbeat    -1.03591188    1.4896402       1.18145180  0.7272336
## 5        children     0.99956963   -1.3394346       1.23623011 -0.5622149
## 6       classical    -0.34004291   -0.6115806      -0.05659981 -1.2227353
## 7          comedy    -0.21284543   -0.1782840       0.46966421  0.4359696
## 8   drum-and-bass    -0.09859309    0.1278346       0.27871630  1.2993588
## 9       grindcore    -1.02896413   -1.3709053      -1.15004119  1.4256325
## 10            idm    -0.92072315    0.3865181       0.21550563 -0.1855622
## 11        iranian    -1.70459096    1.5787713      -1.27678293 -0.5918666
## 12 minimal-techno     1.00300628    1.0131938       0.92329646  0.6763312
## 13          sleep     1.29964810   -0.6975179      -1.81447926 -1.1369751
## 14          study    -0.06338273   -1.2681010       1.03992001 -0.8430977
##         Avg_key Avg_loudness Avg_speechiness Avg_acousticness
## 1   0.355800737   0.71893340     -0.26830368     -0.328909080
## 2  -0.969492159  -1.25916906     -0.46730867      1.337190666
## 3   0.459351530   0.85838082     -0.23391524     -1.173454594
## 4   1.662086242   0.45248034     -0.28628854     -1.115965715
## 5  -2.132122334   0.27814188     -0.21366195      0.475921448
## 6  -0.830623044  -0.41156514     -0.39398513      1.389425229
## 7  -0.470749427   0.14328005      3.44475562      1.181798354
## 8   1.016218500   1.20205970     -0.10365029     -1.112971393
## 9  -0.008454808   0.91185414      0.04284445     -1.235503393
## 10  0.062180251  -0.20818950     -0.29548634     -0.009641503
## 11  0.158699831  -0.79071108     -0.31592128      0.269326877
## 12  1.055073897   0.56667086     -0.33886275     -1.047865561
## 13 -0.973778770  -2.53330324     -0.40477970      0.847418044
## 14  0.615809556   0.07113685     -0.16543648      0.523230621
##    Avg_instrumentalness Avg_liveness Avg_valence   Avg_tempo
## 1            -1.3896359 -0.186772043  0.95548206  0.31814876
## 2             0.7011211 -0.619275988 -0.67658577 -0.39077708
## 3            -0.0348533 -0.006257917 -0.82973639  0.64466288
## 4             0.2782837 -0.534213764  0.98610393  0.59843180
## 5            -1.4301080 -0.468474118  2.00912725  0.29459755
## 6            -0.6657660 -0.383781072  0.64419149 -0.35419406
## 7            -1.6640401  3.122383225  0.67152536 -1.03963286
## 8            -0.7161355  0.085621772 -0.07289295  2.08547779
## 9             0.3245678  0.283600912 -0.68635619  0.07693816
## 10            0.8492862 -0.550453584 -0.18521062  0.32888337
## 11            0.8673247 -0.583722126 -1.21455197 -0.53569371
## 12            0.4557231 -0.472051054 -0.27447826  0.66126095
## 13            1.1712882  0.923745839 -1.69402897 -2.31327844
## 14            1.2529441 -0.610350083  0.36741103 -0.37482512
```

```
similarity_df
```

```
##                acoustic  ambient black-metal breakbeat children classical
## acoustic       0.000000 4.548084    3.595068  3.746872 3.309705  3.481861
## ambient        4.548084 0.000000    5.097943  5.908850 4.789071  2.779575
## black-metal    3.595068 5.097943    0.000000  3.331079 5.953531  4.776964
## breakbeat      3.746872 5.908850    3.331079  0.000000 5.902553  5.041977
## children       3.309705 4.789071    5.953531  5.902553 0.000000  3.216209
## classical      3.481861 2.779575    4.776964  5.041977 3.216209  0.000000
## comedy         5.674253 6.812785    6.486918  6.915304 6.130349  5.675288
## drum-and-bass  2.995088 5.965190    2.450374  3.097348 5.422391  5.111595
## grindcore      4.193522 5.492256    2.582484  4.579386 5.711883  4.706539
## idm            3.584147 3.447825    2.881584  2.997926 4.812773  2.960806
## iranian        5.395415 3.908792    3.611021  4.521302 6.705292  4.099746
## minimal-techno 2.819453 4.725215    2.687981  2.543756 5.352713  4.754701
## sleep          6.568918 3.418893    6.413379  7.810586 6.984248  4.999930
## study          3.709083 3.593530    4.630676  4.155043 4.383285  2.965055
##                  comedy drum-and-bass grindcore      idm  iranian
## acoustic       5.674253      2.995088  4.193522 3.584147 5.395415
## ambient        6.812785      5.965190  5.492256 3.447825 3.908792
## black-metal    6.486918      2.450374  2.582484 2.881584 3.611021
## breakbeat      6.915304      3.097348  4.579386 2.997926 4.521302
## children       6.130349      5.422391  5.711883 4.812773 6.705292
## classical      5.675288      5.111595  4.706539 2.960806 4.099746
## comedy         0.000000      6.520646  6.246268 6.284970 7.043946
## drum-and-bass  6.520646      0.000000  3.445661 3.613628 5.335947
## grindcore      6.246268      3.445661  0.000000 3.423917 4.501607
## idm            6.284970      3.613628  3.423917 0.000000 2.581334
## iranian        7.043946      5.335947  4.501607 2.581334 0.000000
## minimal-techno 6.793920      2.649359  4.147705 2.877986 4.580334
## sleep          7.258629      7.812955  6.170857 5.467318 5.020855
## study          6.387551      4.674219  4.253771 2.497464 4.466007
##                minimal-techno    sleep    study
## acoustic             2.819453 6.568918 3.709083
## ambient              4.725215 3.418893 3.593530
## black-metal          2.687981 6.413379 4.630676
## breakbeat            2.543756 7.810586 4.155043
## children             5.352713 6.984248 4.383285
## classical            4.754701 4.999930 2.965055
## comedy               6.793920 7.258629 6.387551
## drum-and-bass        2.649359 7.812955 4.674219
## grindcore            4.147705 6.170857 4.253771
## idm                  2.877986 5.467318 2.497464
## iranian              4.580334 5.020855 4.466007
## minimal-techno       0.000000 6.666622 3.706868
## sleep                6.666622 0.000000 5.500628
## study                3.706868 5.500628 0.000000
```

```
# This section is a repetition of the above section, except it uses the hclust function so that
I could make the plot showing the final genres

# create null dataframe of average values for each genre
average_df = data.frame(
  Genre = NULL,
  Avg_popularity = NULL,
  Avg_duration = NULL,
  Avg_danceability = NULL,
  Avg_energy = NULL,
  Avg_key = NULL,
  Avg_loudness = NULL,
  Avg_speechiness = NULL,
  Avg_acousticness = NULL,
  Avg_intrumentalness = NULL,
  Avg_liveness = NULL,
  Avg_valence = NULL,
  Avg_tempo = NULL
)

# fill out the above dataframe
genres = unique(df$track_genre)
for (i in genres) {
  genre_df = filter(df, track_genre == i)
  average_df = rbind(average_df, data.frame(
    Genre = i,
    Avg_popularity = mean(genre_df$popularity),
    Avg_duration = mean(genre_df$duration_ms),
    Avg_danceability = mean(genre_df$danceability),
    Avg_energy = mean(genre_df$energy),
    Avg_key = mean(genre_df$key),
    Avg_loudness = mean(genre_df$loudness),
    Avg_speechiness = mean(genre_df$speechiness),
    Avg_acousticness = mean(genre_df$acousticness),
    Avg_instrumentalness = mean(genre_df$instrumentalness),
    Avg_liveness = mean(genre_df$liveness),
    Avg_valence = mean(genre_df$valence),
    Avg_tempo = mean(genre_df$tempo)
  ))
}

# Create distance/ similarity matrix
numeric_cols = average_df[, -1]
distance_matrix = dist(numeric_cols, method = "euclidean")

# Perform clustering and create graph
x = hclust(distance_matrix, method = "average", members = NULL)
plot(x, labels = genres,
     axes = FALSE,
     main = "Cluster Dendrogram of the Final Genre Groups",
     sub = NULL, xlab = NULL, ylab = "Height")
```




```
# set seed and select training/test data
set.seed(123)
n = nrow(df)
tr = sample(x=1:n, size=floor(.8*n), replace=FALSE)
train = df[tr,]
test = df[-tr,]
nt = nrow(test)
```

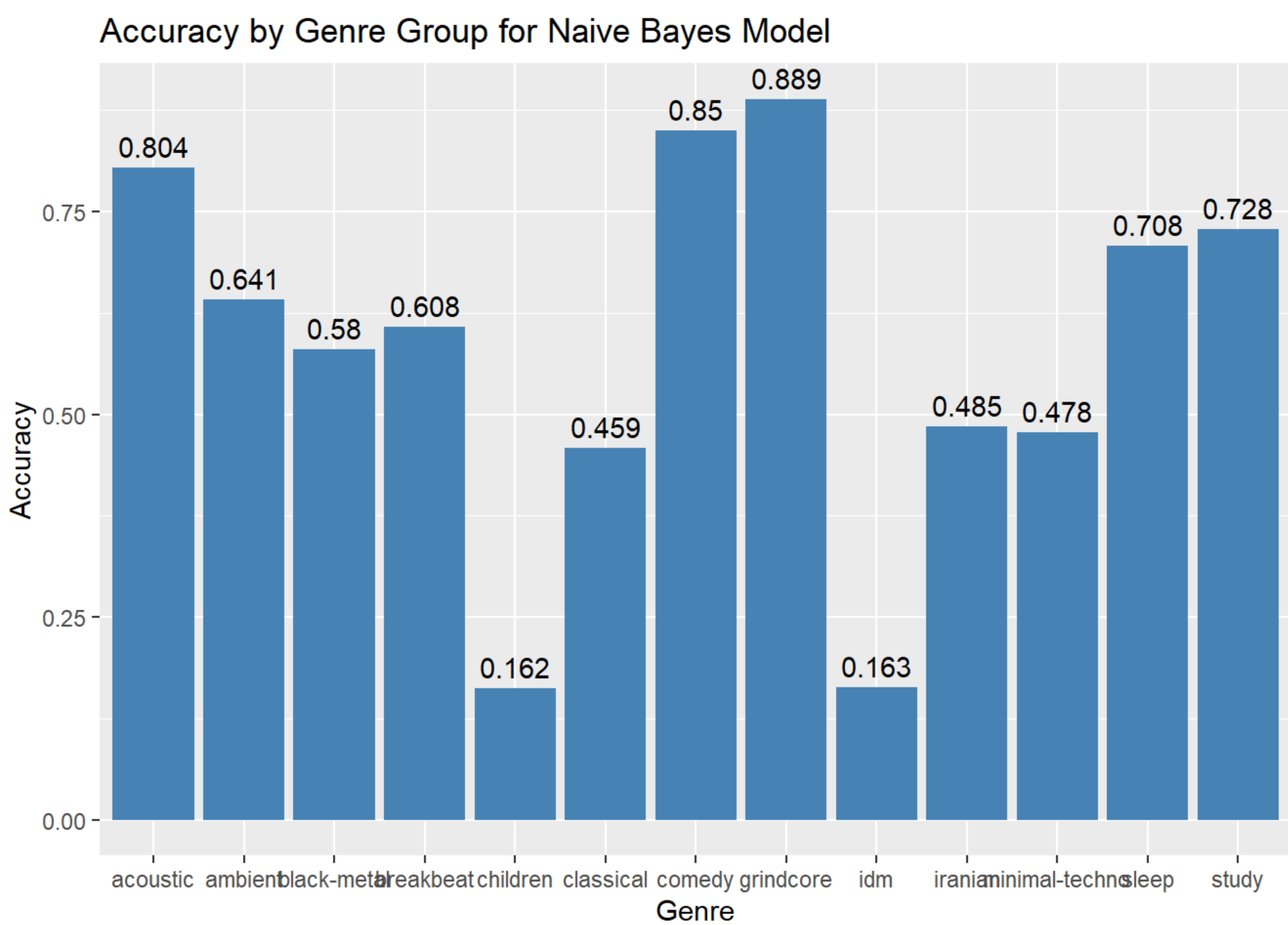
```
# create naive bayes model using training data
model = naiveBayes(track_genre ~., data = train)
# make predictions on the test data
predictions = predict(model, newdata = test)

# create a dataframe with columns containing the model's prediction and the actual genre
comparedf = data.frame(predictions,test$track_genre)
colnames(comparedf) = c("Prediction","Actual")
nmodified = nrow(comparedf)

# compare the predictions and actual values to count how many are correct
count = 0
for (j in 1:nmodified) {
  if (comparedf$Prediction[j] == comparedf$Actual[j]) {
    count = count+1
  }
}

# display overall accuracy
overall_acc = count/nmodified
print(paste("Overall model accuracy: ", overall_acc))
```

```
## [1] "Overall model accuracy:  0.742549499897938"
```

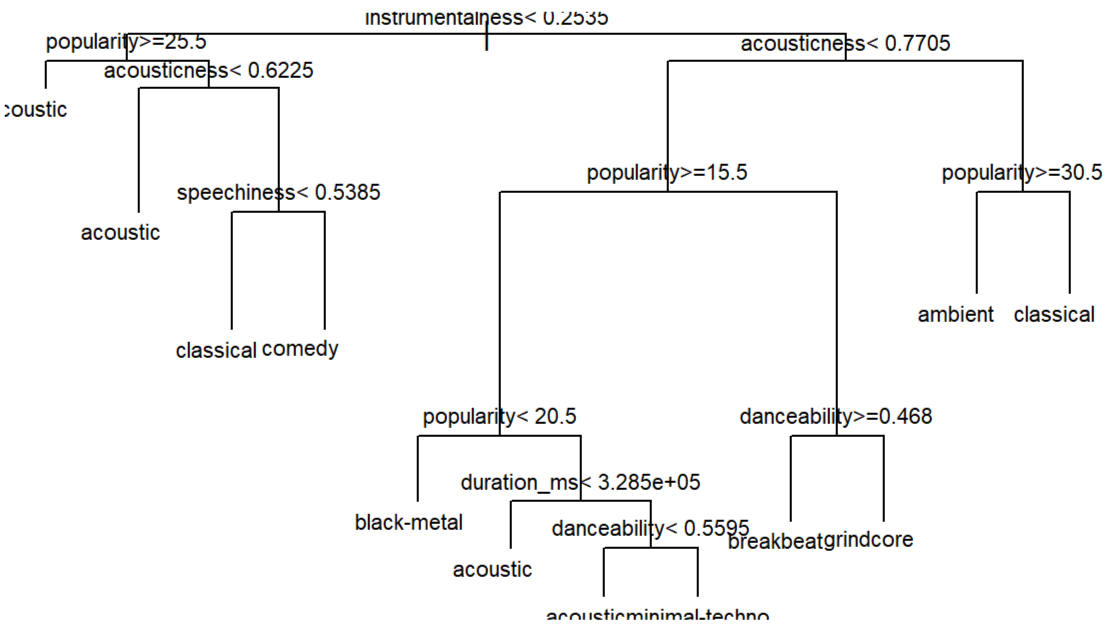
```
# create a vector of all of the genres in the dataset
genres = unique(comparedf$Actual)

# create an empty vector where the accuracies will be stored by genre
accuracy_vec = numeric(length(genres))

# iterate through the genres, getting the accuracy for each genre
for (i in seq_along(genres)) {
    tempdf = filter(comparedf, Actual == genres[i])
    count = sum(tempdf$Prediction == tempdf$Actual)
    accuracy = count / nrow(tempdf)
    accuracy_vec[i] = accuracy
}

# create and display a dataframe of the accuracy scores for each genre
newdf = data.frame(genres, accuracy_vec)
newdf
```

```
##            genres accuracy_vec
## 1         acoustic    0.8041182
## 2          ambient    0.6409807
## 3      black-metal    0.5798319
## 4        breakbeat    0.6079447
## 5         children    0.1625000
## 6        classical    0.4589041
## 7           comedy    0.8504673
## 8        grindcore    0.8888889
## 9              idm    0.1627907
## 10         iranian    0.4848485
## 11 minimal-techno    0.4780115
## 12           sleep    0.7077922
## 13           study    0.7277487
```

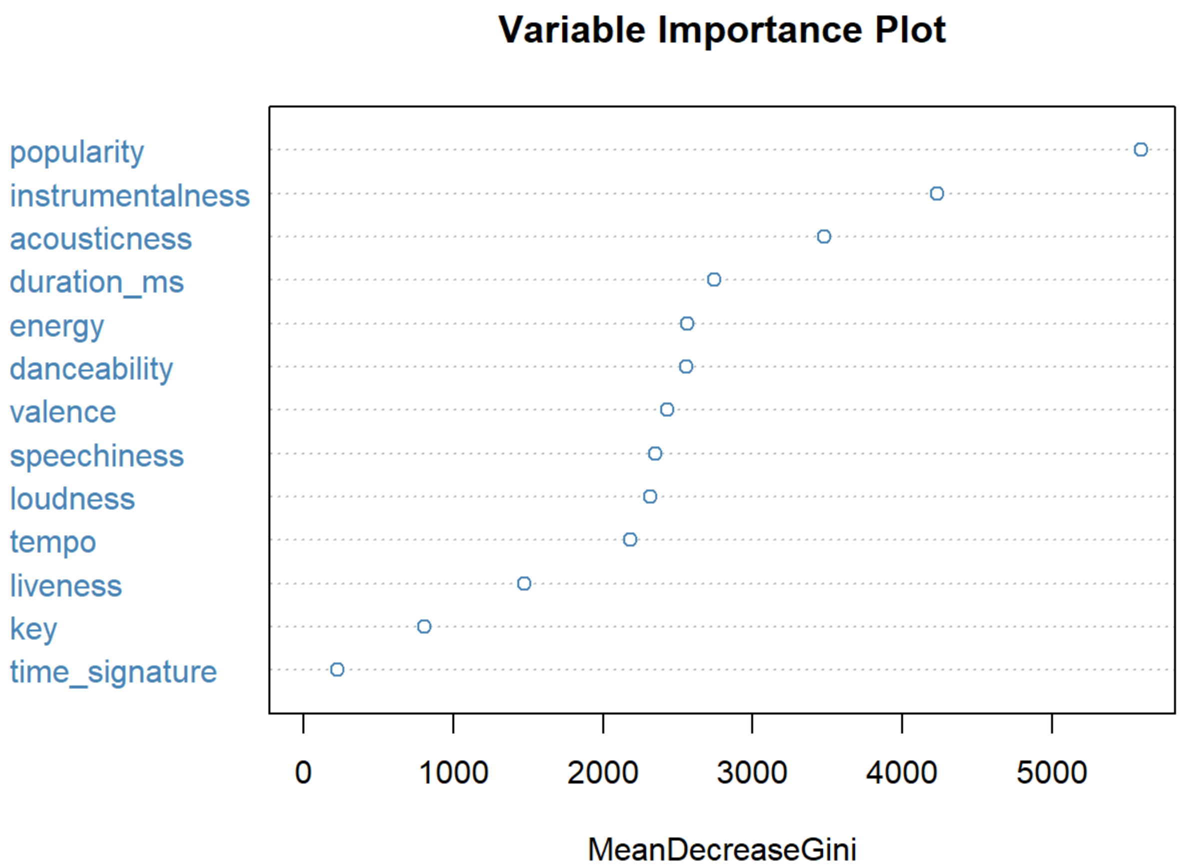
```
# sort the dataframe by accuracy and round to 3 decimals
newdf = newdf[order(accuracy_vec), ]
newdf$accuracy_vec = round(newdf$accuracy_vec, 3)

# create a barplot showing all of the genres and how accurate the model is for each one
ggplot(newdf, aes(x = genres, y = accuracy_vec)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = accuracy_vec), vjust = -0.5) +
  labs(title = "Accuracy by Genre Group for Naive Bayes Model", x = "Genre", y = "Accuracy")
```



```
# Create and plot an example of a single tree
ttree = rpart(track_genre~., data=train, method="class")
plot(ttree)
text(ttree, pretty=0, cex=.7)
```




```
# make sure track_genre is in the right format for randomForest function
train$track_genre = as.factor(train$track_genre)
test$track_genre = as.factor(test$track_genre)

# create random forest model and make predictions stored in yhat_rf
rf = randomForest(track_genre~., data=train, ntree = 100)
yhat_rf = predict(rf, newdata=test)

# create a plot of variable importance given by random forest model
varImpPlot(rf, main="Variable Importance Plot", col="steelblue")
```



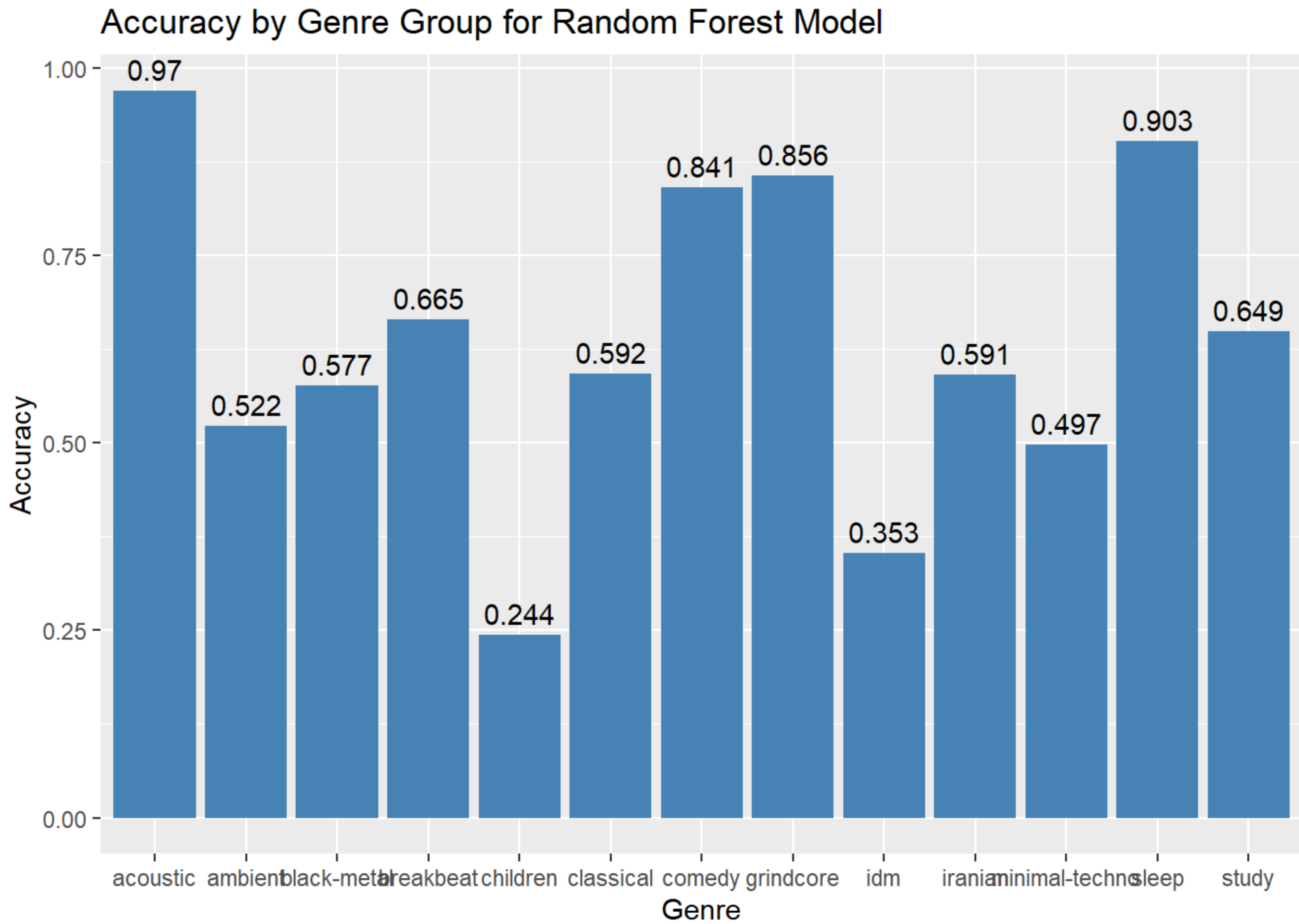

```
# create a dataframe that stores the predictions and actual values
comparedf_rf = data.frame(
  Predicted = yhat_rf,
  Actual = test$track_genre)

# compare the predictions and actual values to count how many are correct
count = 0
nmodified = nrow(comparedf_rf)
for (j in 1:nmodified) {
  if (comparedf_rf$Predicted[j] == comparedf_rf$Actual[j]) {
    count = count+1
  }
}

# display overall accuracy
overall_acc = count/nmodified
print(paste("Overall model accuracy: ", overall_acc))
```

```
## [1] "Overall model accuracy:  0.878291488058788"
```

```r
# create a vector of all of the genres in the dataset
genres = unique(comparedf_rf$Actual)

# create an empty vector where the accuracies will be stored by genre
accuracy_vec = numeric(length(genres))

# iterate through the genres, getting the accuracy for each genre
for (i in seq_along(genres)) {
    tempdf = filter(comparedf_rf, Actual == genres[i])
    count = sum(tempdf$Predicted == tempdf$Actual)
    accuracy = count / nrow(tempdf)
    accuracy_vec[i] = accuracy
}

# create and display a dataframe of the accuracy scores for each genre
newdf = data.frame(genres, accuracy_vec)
newdf
```

```
##            genres accuracy_vec
## 1        acoustic    0.9699213
## 2         ambient    0.5218914
## 3     black-metal    0.5770308
## 4       breakbeat    0.6649396
## 5        children    0.2437500
## 6       classical    0.5916096
## 7          comedy    0.8411215
## 8       grindcore    0.8555556
## 9             idm    0.3534884
## 10        iranian    0.5909091
## 11 minimal-techno    0.4971319
## 12          sleep    0.9025974
## 13          study    0.6492147
```

```r
# sort the dataframe by accuracy and round to 3 decimals
newdf = newdf[order(accuracy_vec), ]
newdf$accuracy_vec = round(newdf$accuracy_vec, 3)

# create a barplot showing all of the genres and how accurate the model is for each one
ggplot(newdf, aes(x = genres, y = accuracy_vec)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = accuracy_vec), vjust = -0.5) +
  labs(title = "Accuracy by Genre Group for Random Forest Model", x = "Genre", y = "Accuracy")
```


Accuracy by Genre Group for Random Forest Model
1.00
0.75
0.50
0.25
0.00
Accuracy
0.97
0.522
0.577
0.665
0.244
0.592
0.841
0.856
0.353
0.591
0.497
0.903
0.649
acoustic
ambient
black-metal
breakbeat
children
classical
comedy
grindcore
idm
iranian
minimal-techno
sleep
study
Genre