Music Genres Classification

Filippo Boldrini, Federico Sandrinelli, Gabriele Sanguin

Contents

Introduction

Music genre classification is a task for various music platforms and it plays a crucial role in many domains, such as music recommendation systems and personalized user experiences. The objective of this report is to study this task from a statistical point of view using a dataset of music taken from Spotify. We will explore the features of our data and investigate different techniques to classify five major music genres.

We will first provide an overview of our dataset, with the description and pre-processing of the data. Then we will proceed with exploratory data analysis and the building of the models to predict the genre class with different approaches. Finally we will evaluate the performance of our models and make a comparison between them. We will use evaluation metrics such as accuracy, precision, recall and F1 score.

Loading and Preprocessing Data

Loading Libraries

We start with the loading of R packages required beside base R. They will be useful for the data analysis and data visualization.

```
library(dplyr)
library(ggplot2)
library(ggrepel)
library(readr)
library(corrplot)
library(plotly)
library(caret)
library(rpart)
library(rpart.plot)
library(nortest)
library(tidyverse)
library(gridExtra)
library(tidymodels)
library(leaps)
library(glmnet)
library(pROC)
library(rsample)
library(correlation)
library(DataExplorer)
library(knitr)
library(outliers)
library(class)
library(grid)
library(cowplot)
library(car)
library(ISLR2)
library(MASS)
library(e1071)
```

Dataset Presentation

The dataset we are using consists on data of Spotify tracks over a range of more than 100 different genres. Spotify uses an automated system to analyze the audio characteristics of the songs and each track of the dataset present these audio features with mostly numerical values.

We downloaded the dataset from Kaggle: Spotify Tracks Dataset

Let's proceed on the loading:

```
spotify_data <- read.csv("./dataset_spotify.csv")
str(spotify_data)
```

```
## 'data.frame':    114000 obs. of  21 variables:
##  $ X               : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ track_id        : chr  "5SuOikwiRyPMVoIQDJUgSV" "4qPNDBW1i3p13qLCt0Ki3A" "1iJBSr7s7jYXzM8EGcbK5b"
##  $ artists         : chr  "Gen Hoshino" "Ben Woodward" "Ingrid Michaelson;ZAYN" "Kina Grannis" ...
```

```
##  $ album_name      : chr  "Comedy" "Ghost (Acoustic)" "To Begin Again" "Crazy Rich Asians (Original
##  $ track_name      : chr  "Comedy" "Ghost - Acoustic" "To Begin Again" "Can't Help Falling In Love"
##  $ popularity      : int  73 55 57 71 82 58 74 80 74 56 ...
##  $ duration_ms     : int  230666 149610 210826 201933 198853 214240 229400 242946 189613 205594 ...
##  $ explicit        : chr  "False" "False" "False" "False" ...
##  $ danceability    : num  0.676 0.42 0.438 0.266 0.618 0.688 0.407 0.703 0.625 0.442 ...
##  $ energy          : num  0.461 0.166 0.359 0.0596 0.443 0.481 0.147 0.444 0.414 0.632 ...
##  $ key             : int  1 1 0 0 2 6 2 11 0 1 ...
##  $ loudness        : num  -6.75 -17.23 -9.73 -18.52 -9.68 ...
##  $ mode            : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ speechiness     : num  0.143 0.0763 0.0557 0.0363 0.0526 0.105 0.0355 0.0417 0.0369 0.0295 ...
##  $ acousticness    : num  0.0322 0.924 0.21 0.905 0.469 0.289 0.857 0.559 0.294 0.426 ...
##  $ instrumentalness: num  1.01e-06 5.56e-06 0.00 7.07e-05 0.00 0.00 2.89e-06 0.00 0.00 4.19e-03 ...
##  $ liveness        : num  0.358 0.101 0.117 0.132 0.0829 0.189 0.0913 0.0973 0.151 0.0735 ...
##  $ valence         : num  0.715 0.267 0.12 0.143 0.167 0.666 0.0765 0.712 0.669 0.196 ...
##  $ tempo           : num  87.9 77.5 76.3 181.7 119.9 ...
##  $ time_signature  : int  4 4 4 3 4 4 3 4 4 4 ...
##  $ track_genre     : chr  "acoustic" "acoustic" "acoustic" "acoustic" ...
```

Our dataset is made of 114000 lines and 21 columns. Here follows a brief description of the features:

- **track_id**: The Spotify ID for the track.
- **artists**: The artists names who performed the track. If there is more than one artist, they are separated by a ';'.
- **album_name**: The album name in which the track appears.
- **track_name**: Name of the track.
- **popularity**: The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity.
- **duration_ms**: The track length in milliseconds.
- **explicit**: Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown).
- **danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
- **energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.
- **key**: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C /D , 2 = D, and so on. If no key was detected, the value is -1.
- **loudness**: The overall loudness of a track in decibels (dB).
- **mode**: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
- **speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
- **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.

- **instrumentalness**: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
- **liveness**: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
- **valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
- **tempo**: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
- **time_signature**: An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.
- **track_genre**: The genre in which the track belongs.

Dataset Preprocessing

We decided to study a subset of the original dataset, in particular we are focusing on only a few track genres, that is `classical`, `hip-hop`, `rock-n-roll`, `reggaeton` and `techno`. We made this choice to make our analysis clearer and more understandable.

```
spotify <- subset(spotify_data, track_genre %in%
                  c("classical", "hip-hop", "rock-n-roll",
                    "reggaeton", "techno"))
```

First we check if there are any missing values.

```
anyNA(spotify)
```

```
## [1] FALSE
```

The first columns of the dataset contain general information about the audio tracks such as their IDs, names and the artists name. We are not gonna use them in future exploration and analysis of the data so we removed them by extracting a new subset of our dataset, keeping only the useful attributes.

Let's print now the first lines of the dataset to see what it looks like.

```
head(spotify_num,5)
```

```
##       popularity duration_ms explicit danceability energy key loudness mode
## 16001         58      298266    False        0.643  0.268  11  -15.073    0
## 16002         59      482586    False        0.484  0.898   0   -4.132    1
## 16003         54      219437    False        0.608  0.638  11   -6.008    0
## 16004         68      299146    False        0.695  0.293  11  -16.278    0
## 16005         59      387716    False        0.583  0.308   7  -18.303    0
##       speechiness acousticness instrumentalness liveness valence   tempo
## 16001      0.0900        0.593         2.06e-06    0.316   0.620 143.813
## 16002      0.1640        0.365         0.00e+00    0.091   0.680  91.975
## 16003      0.0292        0.581         1.72e-02    0.448   0.439 140.109
## 16004      0.0431        0.596         1.58e-02    0.132   0.637 143.804
## 16005      0.0465        0.581         1.06e-02    0.257   0.241 118.226
```

```
##       time_signature track_genre
## 16001              4   classical
## 16002              4   classical
## 16003              4   classical
## 16004              4   classical
## 16005              4   classical
```

We notice that there are some attributes that are classified as numerical but that represents some categorical feature. for example the `key` column, whose numerical variables each points to a musical key. We want to transform these variables and all character variables into factors:

```
#we transform the numerical and character variables into
#factors
spotify_num$track_genre = as.factor(spotify_num$track_genre)
spotify_num$explicit = as.factor(spotify_num$explicit)
spotify_num$key = as.factor(spotify_num$key)
spotify_num$mode = as.factor(spotify_num$mode)
spotify_num$time_signature = as.factor(spotify_num$time_signature)
```

Finally, we check if the proportion of each genre is balanced in our dataframe. This is necessary to be sure there are not particular differences between track genres when we are going to train our models in predicting the track genre.

```
table(spotify_num$track_genre)
```

```
##
##   classical     hip-hop   reggaeton rock-n-roll      techno
##        1000        1000        1000        1000        1000
```

```
prop.table(table(spotify_num$track_genre))
```

```
##
##   classical     hip-hop   reggaeton rock-n-roll      techno
##         0.2         0.2         0.2         0.2         0.2
```

There are exactly 1000 sample for each music genre, hence the dataset is perfectly balanced.

Exploratory Data Analysis

Overview

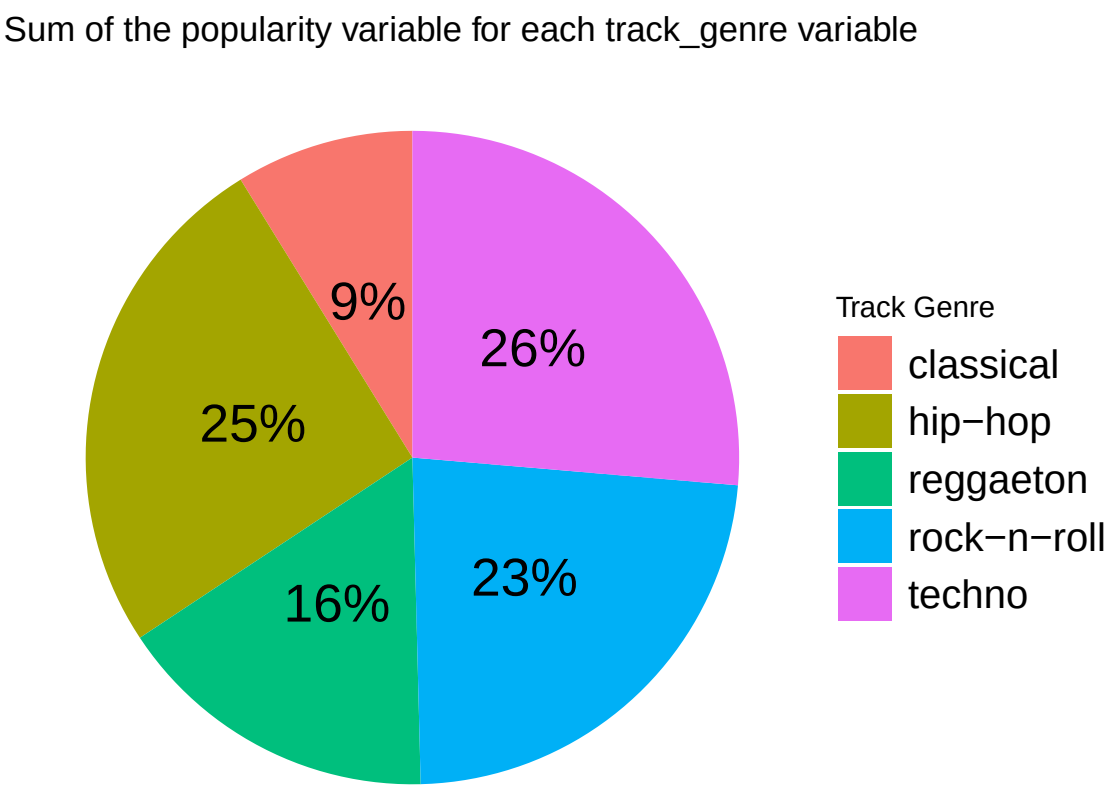
We are going now to explore the features in order to understand their main characteristics and uncover underlying patterns if present. Firstly we are going to see how the features behave in all music tracks, then we will focus on the relations of the features with the different track genres. In fact, later on `track_genres` will be used as target variable for our classification problem.

Let's have a general look to all features.

```
summary(spotify_num)
```

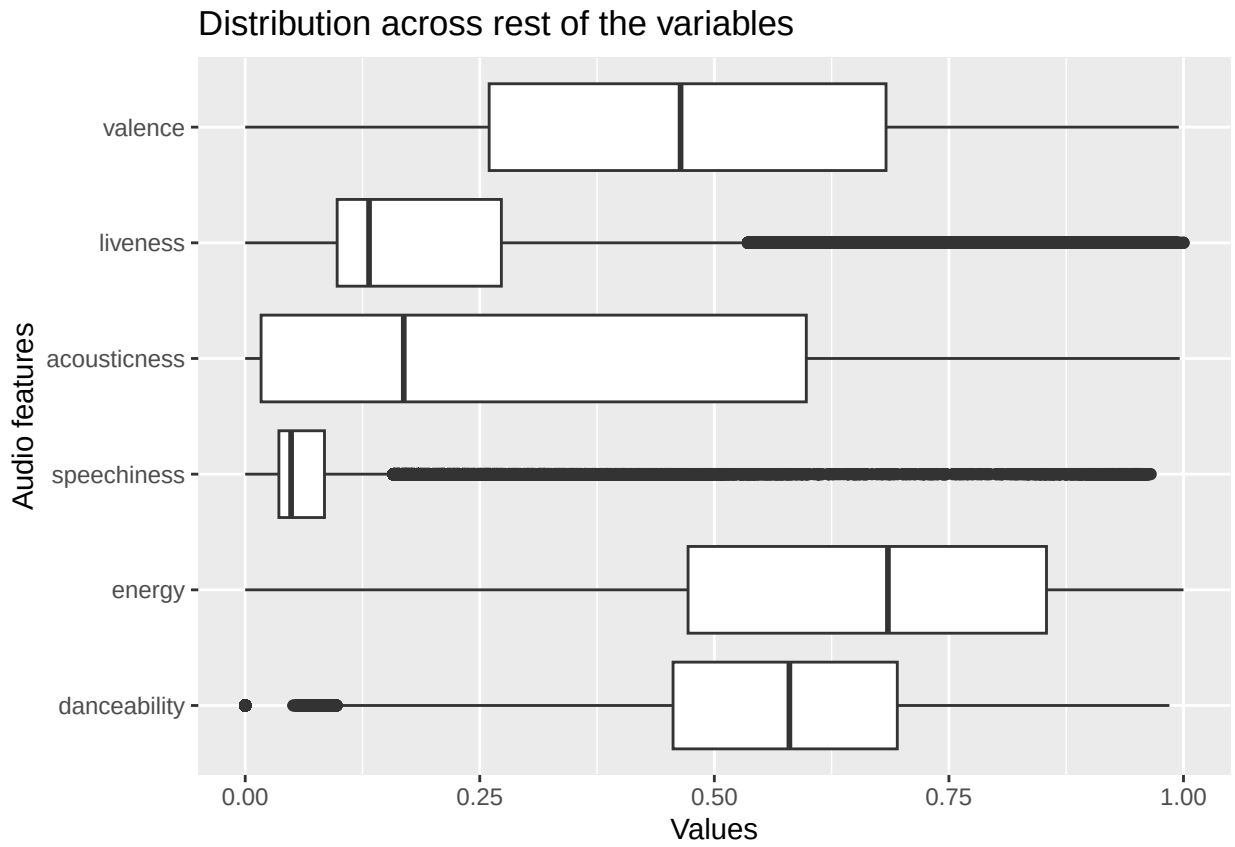
```
##    popularity     duration_ms       explicit     danceability
##  Min.   : 0.00   Min.   :  17453   False:4441   Min.   :0.0713
##  1st Qu.: 1.00   1st Qu.: 162897   True : 559   1st Qu.:0.4980
##  Median :30.00   Median : 202825                Median :0.6610
##  Mean   :29.63   Mean   : 227789                Mean   :0.6222
##  3rd Qu.:53.00   3rd Qu.: 256933                3rd Qu.:0.7660
##  Max.   :99.00   Max.   :2646866                Max.   :0.9640
##
##      energy              key          loudness       mode      speechiness
##  Min.   :0.000756   7      : 624   Min.   :-41.531   0:1879   Min.   :0.02410
##  1st Qu.:0.412000   0      : 532   1st Qu.:-11.455   1:3121   1st Qu.:0.03830
##  Median :0.631000   2      : 485   Median : -7.499            Median :0.05150
##  Mean   :0.576823   1      : 462   Mean   : -9.715            Mean   :0.08355
##  3rd Qu.:0.789000   9      : 449   3rd Qu.: -5.194            3rd Qu.:0.09110
##  Max.   :0.999000   4      : 406   Max.   :  0.550            Max.   :0.76600
##                     (Other):2042
##   acousticness       instrumentalness      liveness         valence
##  Min.   :0.0000083   Min.   :0.000000   Min.   :0.0150   Min.   :0.0000
##  1st Qu.:0.0493000   1st Qu.:0.000000   1st Qu.:0.0953   1st Qu.:0.2940
##  Median :0.2280000   Median :0.000064   Median :0.1230   Median :0.5330
##  Mean   :0.3827493   Mean   :0.237432   Mean   :0.1833   Mean   :0.5135
##  3rd Qu.:0.7440000   3rd Qu.:0.598250   3rd Qu.:0.2320   3rd Qu.:0.7350
##  Max.   :0.9960000   Max.   :0.982000   Max.   :1.0000   Max.   :0.9880
##
##      tempo        time_signature      track_genre
##  Min.   : 35.93   1:  61         classical  :1000
##  1st Qu.: 95.00   3: 418         hip-hop    :1000
##  Median :117.99   4:4452         reggaeton  :1000
##  Mean   :118.45   5:  69         rock-n-roll:1000
##  3rd Qu.:135.00                  techno     :1000
##  Max.   :214.02
##
```

As a first curiosity fact, we start by seeing which genre is more listened to by summing up the `popularity` of the different genres:



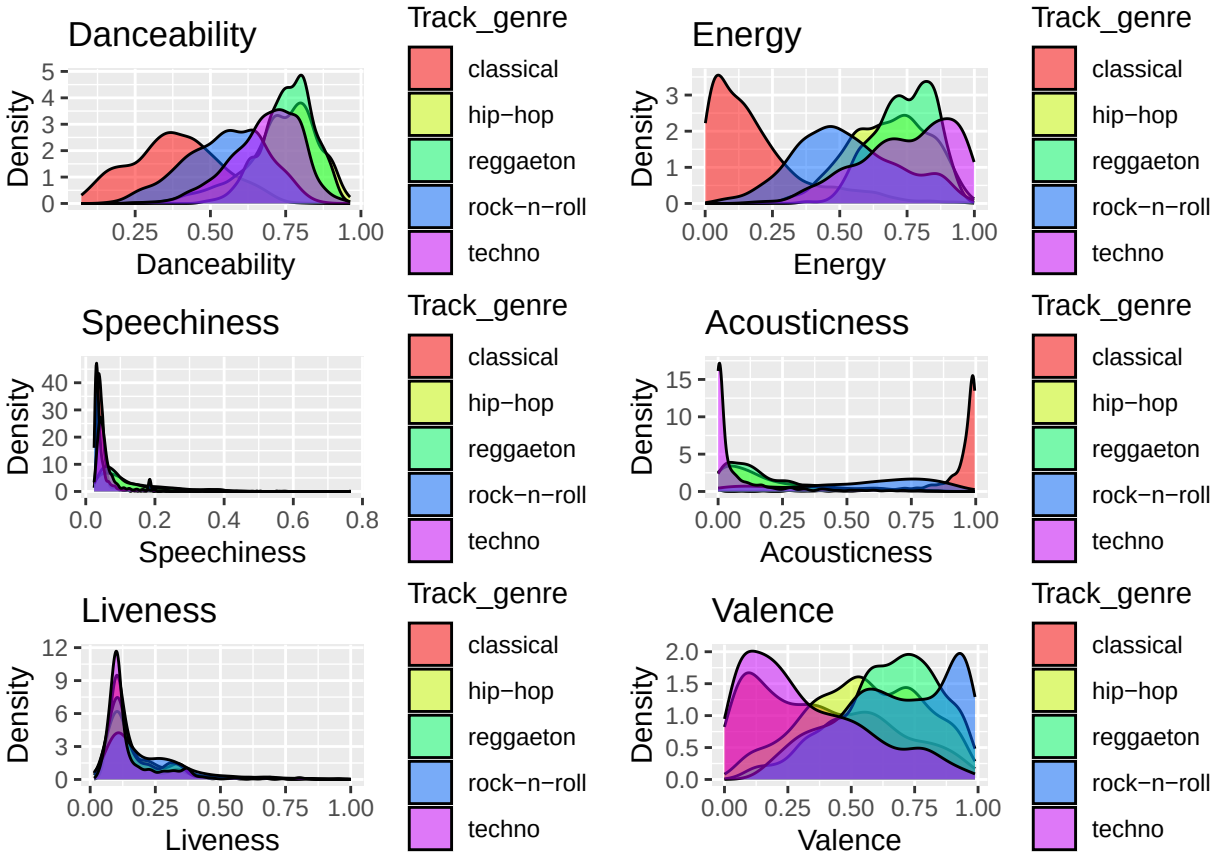
As we might expect, `classical` is the least popular genre whereas `techno` and `hip-hop` tracks are pretty popular respect to the other music genres.

For a general overview of audio attributes in all tracks:



We can make some observation from these boxplot. The `valence` seems to have a good gaussian shape in $[0, 1]$ with center near 0.5. `Liveness` and `speechiness` instead are highly skewed, with the majority of points near to the median. We can notice a large number of outliers in more than one features, taking into account the context we are studying this is no surprising as music does not follow strict rules.

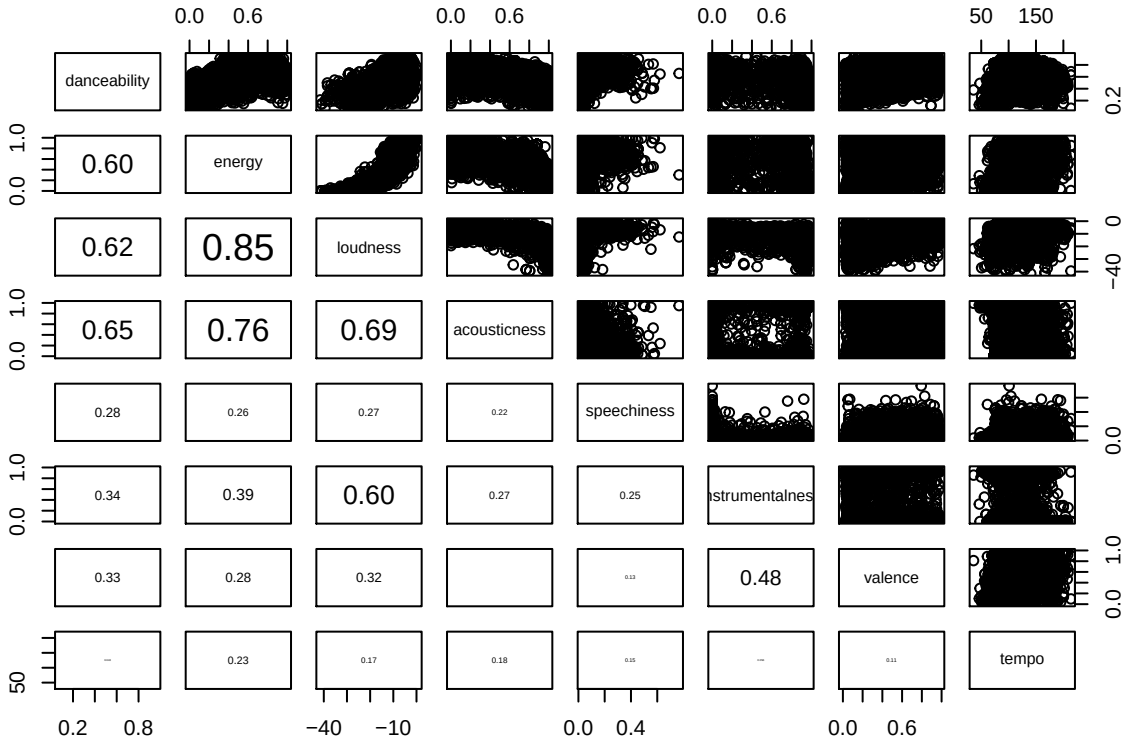
Let's compare now the density plot of each `track_genre` for some relevant features:



From the plots above we can make some useful observation. Note that different features distribute in different ways between genres, in particular `acousticness` emphasis the different instruments used in `classical` and `techno` music, while the `energy` and `valence` features present more 'colorful' distributions as they change between one genre to another. `speechnes` and `liveness` instead doesn't seem to have any important impact for the distinction of the `track_genre`.

Correlations

We focus now on studying the correlations between numerical features. Correlations are statistical measures used to quantify the strength of the linear relationship between two variables, so how they are influenced by each other. We start to investigate them through the Pearson coefficient and the scatter plot.

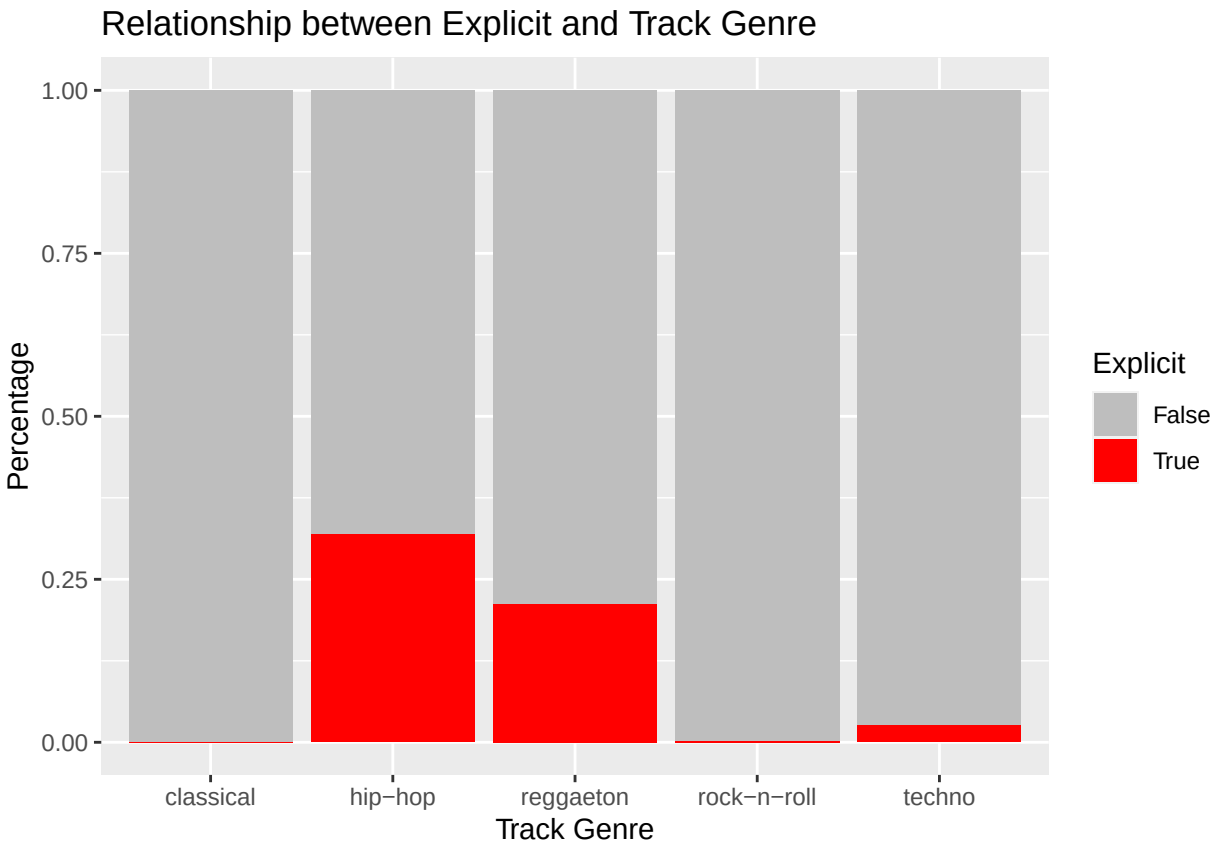


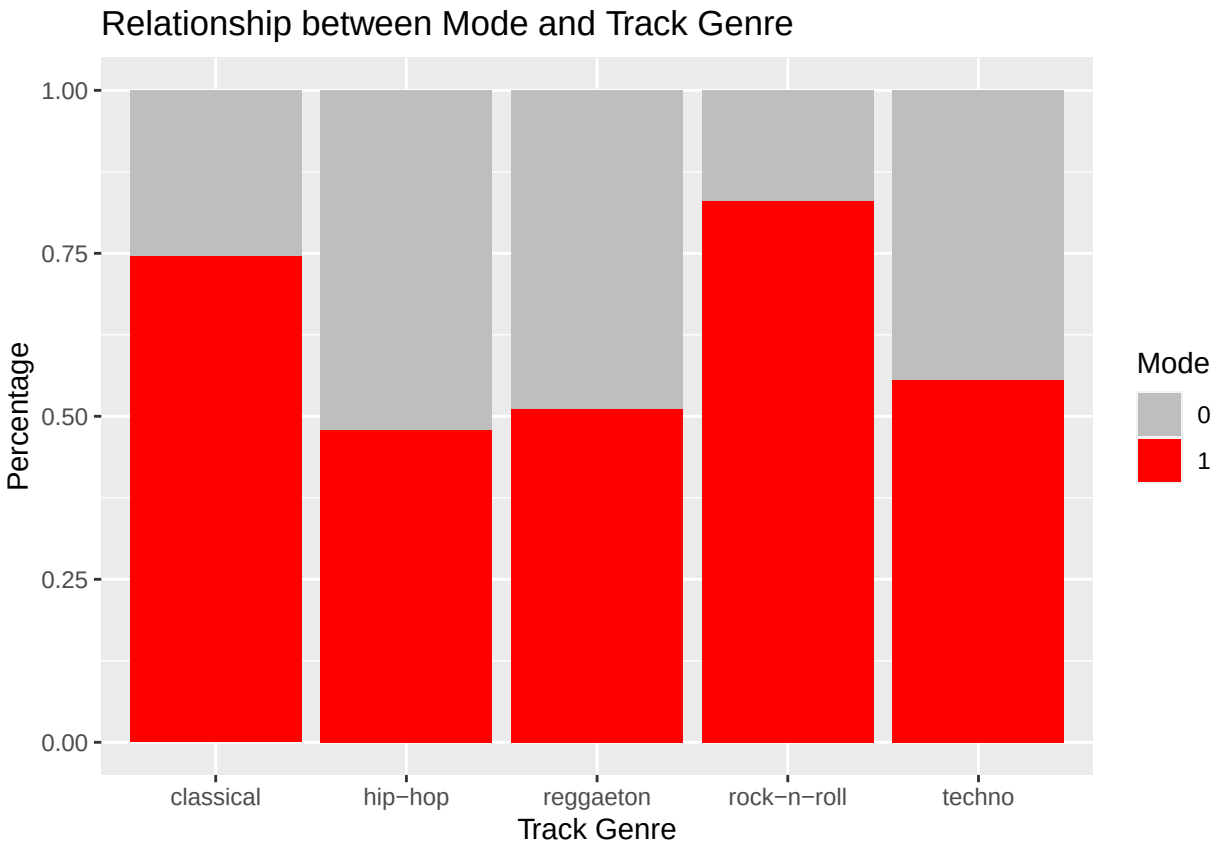
In general there are very weak correlations between most variables with some exceptions. In particular `danceability`, `energy`, `loudness` and `acousticness` shows some significant relations between each others. At the same time features like `speechiness`, `tempo` and `valence` shows almost zero correlations.

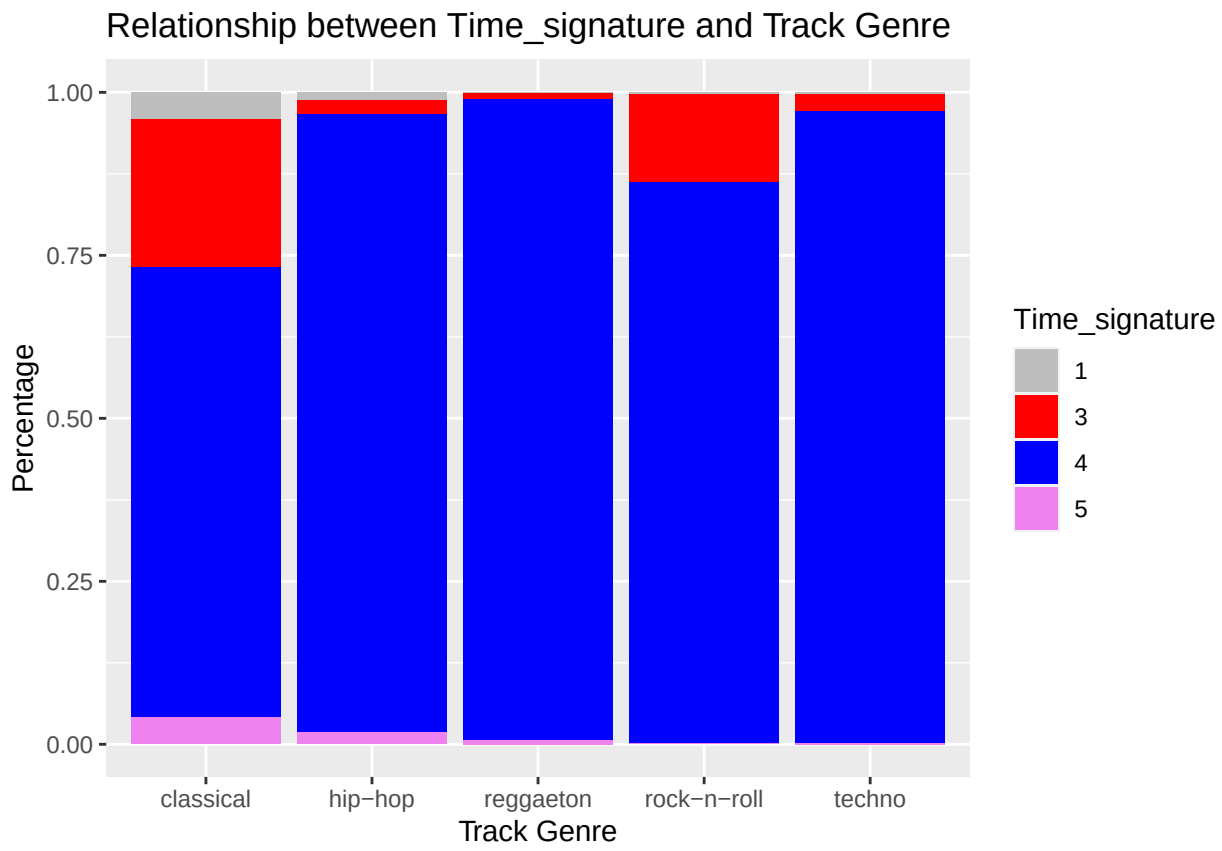
Categorical features

We proceed now to explore the categorical features and their relation with `track_genre`.

We plot the distribution of `explicit`, `mode` and `time_signature` among all the genres.

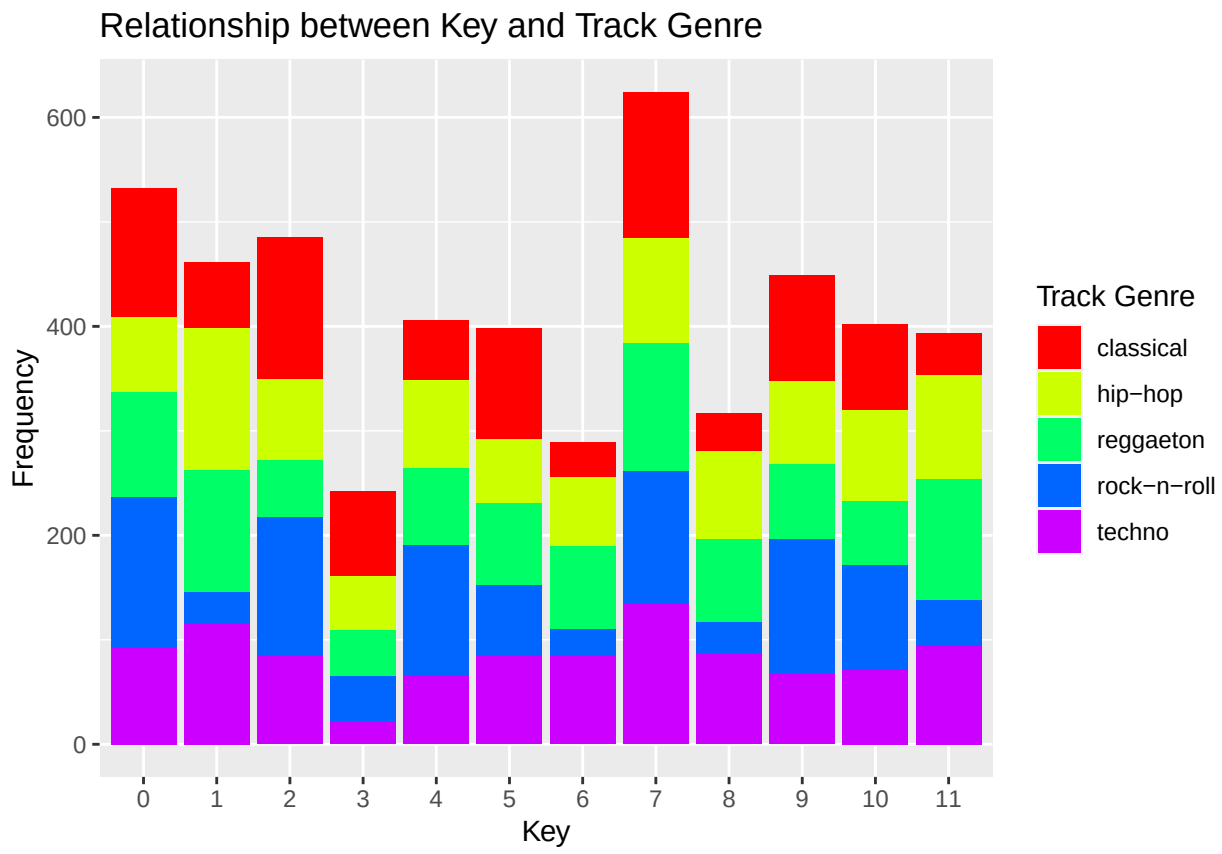

Relationship between Explicit and Track Genre
1.00
0.75
0.50
0.25
0.00
Percentage
classical
hip−hop
reggaeton
rock−n−roll
techno
Track Genre
Explicit
False
True


Relationship between Mode and Track Genre
1.00
0.75
0.50
0.25
0.00
Percentage
classical
hip−hop
reggaeton
rock−n−roll
techno
Track Genre
Mode
0
1



For what concern the `explicit` feature it is possible to see that it occurs significantly in hip hop and reggaeton tracks, while it is almost absent in other genres. The `mode` variable is distribute quite uniformly with an higher presence of major tonalities in rock and roll and classical. `time_signature` shows a prevalence of 4/4 in all the genres besides in classical where there is a soft diversity.

We now a take a look to the absolute frequencies of different keys.



The `key` 7 is the most popular but without a neat advantage. we notice there isn't any clear correlation between `key`and `track_genre`

Building models

We now try different techniques used in classification problems to find which performs better with our data. The approaches we are going to try are *Logistic Regression* performed with variable selection through *backward stepwise selection*, *Ridge* and *Lasso Logistic Regression*, *Linear Discriminant Analysis*, *Quadratic Discriminant Analysis* and *Naive Bayes Classifier*.

Logistic Regression

Since Logistic Regression is used to build binary classification models and we are facing a multiclass problem our idea consists in training a logistic model for each genres in our dataset to determine whether a track belongs to it or not. In a second moment we will compare the predictions made by each model to choose which one is more confident. This is basically what is done in the one-vs-all approach for classification in machine learning.

Splitting into train and test set

First thing to do is to create some dummy variables for the `track_genre` so that we can consider them as separated features:

```
dummies <- model.matrix(~ track_genre + 0)
spotify_dummies <- data.frame(spotify_num, dummies)
names(spotify_dummies)[names(spotify_dummies) ==
                    "track_genreclassical" ] <- "classical"
names(spotify_dummies)[names(spotify_dummies) ==
                    "track_genrehip.hop" ] <- "hiphop"
names(spotify_dummies)[names(spotify_dummies) ==
                    "track_genrereggaeton" ] <- "reggaeton"
names(spotify_dummies)[names(spotify_dummies) ==
                    "track_genrerock.n.roll" ] <- "rock.n.roll"
names(spotify_dummies)[names(spotify_dummies) ==
                    "track_genretechno" ] <- "techno"
```

We now have 5 more column in our dataset, each one represent a genre and tell us if a track is labeled with that genre ('1') or not ('0'). In this way it will be possible to train a model for each genre to discriminate whether a track belongs to it or not.

We proceed with the split of the dataset into train and test set. A validation set is not necessary because no hyperparameter needs to be chosen in this section.

```
#splitting in training e test set
set.seed(2222)

split <- initial_split(spotify_dummies, prop = 0.80)
train <- training(split)
test <- testing(split)
```

What is the proportion now between genres in the two subsets?

```
#proportion of the variable "track_genre" in
#training and test set
prop.table(table(train$track_genre))
```

```
##
##   classical     hip-hop   reggaeton rock-n-roll      techno
##     0.19675     0.20275     0.20075     0.19925     0.20050
```

```
prop.table(table(test$track_genre))
```

```
##
##   classical     hip-hop   reggaeton rock-n-roll      techno
##       0.213       0.189       0.197       0.203       0.198
```

Training Logistic Regression Models (variable selection)

We are ready to start the training. As anticipated we will deal with 5 different binary classification problems, that we want to solve with a logistic regression model. In this way, for each model we will find the probability of a track to belong to the correspondent genre.

In order to find the best subset of predictors for each genre we use the *greedy algorithm* of *Backward Stepwise Selection*. We start every time by fitting the full model and then we proceed to remove the predictors with the highest p-value following a backward elimination approach. The final aim is to find that subset of variables

in the dataset that can train a model with low AIC quantity: the *Akaike Information Criterion (AIC)* is an estimator of prediction error and of the relative quality of a statistical model for the chosen set of features. In particular it combines the model's complexity (number of parameters) and how well data fit (expressed as loglikelihood).

$$AIC = 2 * k - \log(L)$$

```
###Classical
glm.classical <- glm(classical~ popularity + duration_ms
                     + explicit + danceability
                     + energy + key + loudness + mode
                     + speechiness + acousticness
                     + instrumentalness + valence+ tempo
                     + liveness + time_signature,
                     data = train, family = binomial)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + explicit +
##     danceability + energy + key + loudness + mode + speechiness +
##     acousticness + instrumentalness + valence + tempo + liveness +
##     time_signature, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3584  -0.0923  -0.0271   0.0000   3.6285
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.388e+00  1.290e+00  -1.852 0.064031 .
## popularity       -3.373e-02  5.739e-03  -5.877 4.18e-09 ***
## duration_ms       8.505e-06  1.311e-06   6.488 8.71e-11 ***
## explicitTrue     -1.520e+01  6.345e+02  -0.024 0.980885
## danceability     -8.579e+00  9.665e-01  -8.877  < 2e-16 ***
## energy           -1.354e+00  9.340e-01  -1.450 0.147168
## key1             -4.295e-02  6.234e-01  -0.069 0.945071
## key2              3.000e-01  4.189e-01   0.716 0.473815
## key3              3.919e-01  5.405e-01   0.725 0.468360
## key4             -2.533e-01  5.205e-01  -0.487 0.626489
## key5              3.350e-01  4.856e-01   0.690 0.490294
## key6              4.776e-01  6.446e-01   0.741 0.458697
## key7              3.315e-01  4.318e-01   0.768 0.442650
## key8              3.541e-01  5.288e-01   0.670 0.503080
## key9              4.013e-01  4.106e-01   0.977 0.328451
## key10            -1.607e-01  4.512e-01  -0.356 0.721675
## key11             1.573e+00  4.829e-01   3.257 0.001125 **
## loudness         -1.183e-01  3.436e-02  -3.443 0.000575 ***
## mode1            -3.481e-01  2.491e-01  -1.398 0.162239
## speechiness       2.132e+00  2.113e+00   1.009 0.313202
## acousticness      6.751e+00  5.975e-01  11.300  < 2e-16 ***
## instrumentalness  3.147e+00  4.008e-01   7.853 4.07e-15 ***
## valence           2.004e+00  5.949e-01   3.368 0.000756 ***
## tempo            -2.431e-03  3.643e-03  -0.667 0.504606
```

```
## liveness         -1.034e+00  7.588e-01  -1.362 0.173169
## time_signature3  -2.219e+00  7.731e-01  -2.870 0.004110 **
## time_signature4  -2.008e+00  7.390e-01  -2.718 0.006576 **
## time_signature5  -1.846e-01  1.080e+00  -0.171 0.864262
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  679.54  on 3972  degrees of freedom
## AIC: 735.54
##
## Number of Fisher Scoring iterations: 19
```

```
#removing explicit
glm.classical <- update(glm.classical, .~.-explicit)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
##     energy + key + loudness + mode + speechiness + acousticness +
##     instrumentalness + valence + tempo + liveness + time_signature,
##     family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2545  -0.0940  -0.0295  -0.0082   3.6232
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.517e+00  1.288e+00  -1.954 0.050646 .
## popularity       -3.273e-02  5.710e-03  -5.733 9.89e-09 ***
## duration_ms       8.438e-06  1.302e-06   6.482 9.08e-11 ***
## danceability     -8.690e+00  9.651e-01  -9.004  < 2e-16 ***
## energy           -1.136e+00  9.301e-01  -1.221 0.222050
## key1             -5.969e-02  6.199e-01  -0.096 0.923292
## key2              2.973e-01  4.187e-01   0.710 0.477751
## key3              3.845e-01  5.392e-01   0.713 0.475839
## key4             -2.580e-01  5.188e-01  -0.497 0.619021
## key5              3.483e-01  4.847e-01   0.719 0.472364
## key6              4.804e-01  6.426e-01   0.747 0.454764
## key7              3.616e-01  4.309e-01   0.839 0.401453
## key8              3.681e-01  5.282e-01   0.697 0.485861
## key9              4.255e-01  4.093e-01   1.040 0.298519
## key10            -1.102e-01  4.512e-01  -0.244 0.806947
## key11             1.431e+00  4.793e-01   2.985 0.002837 **
## loudness         -1.220e-01  3.435e-02  -3.551 0.000384 ***
## mode1            -3.866e-01  2.475e-01  -1.562 0.118271
## speechiness       3.967e-01  1.944e+00   0.204 0.838353
## acousticness      6.842e+00  6.011e-01  11.383  < 2e-16 ***
## instrumentalness  3.208e+00  4.012e-01   7.996 1.28e-15 ***
## valence           2.106e+00  5.949e-01   3.540 0.000401 ***
```

```
## tempo            -2.014e-03  3.634e-03  -0.554 0.579449
## liveness         -9.793e-01  7.529e-01  -1.301 0.193366
## time_signature3  -2.266e+00  7.702e-01  -2.942 0.003258 **
## time_signature4  -2.057e+00  7.362e-01  -2.794 0.005214 **
## time_signature5  -1.321e-01  1.087e+00  -0.122 0.903244
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  684.06  on 3973  degrees of freedom
## AIC: 738.06
##
## Number of Fisher Scoring iterations: 8
```

```
#removing key
glm.classical <- update(glm.classical, .~.-key)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
##     energy + loudness + mode + speechiness + acousticness + instrumentalness +
##     valence + tempo + liveness + time_signature, family = binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1196  -0.1013  -0.0319  -0.0095   3.7867
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -2.016e+00  1.235e+00  -1.633 0.102570
## popularity      -3.103e-02  5.582e-03  -5.559 2.72e-08 ***
## duration_ms      8.472e-06  1.275e-06   6.645 3.03e-11 ***
## danceability    -8.267e+00  9.374e-01  -8.818  < 2e-16 ***
## energy          -1.538e+00  9.038e-01  -1.702 0.088804 .
## loudness        -1.107e-01  3.393e-02  -3.264 0.001098 **
## mode1           -4.781e-01  2.437e-01  -1.962 0.049724 *
## speechiness      1.224e+00  1.994e+00   0.614 0.539463
## acousticness     6.638e+00  5.714e-01  11.617  < 2e-16 ***
## instrumentalness 3.128e+00  3.932e-01   7.956 1.78e-15 ***
## valence          2.004e+00  5.852e-01   3.424 0.000617 ***
## tempo           -2.388e-03  3.572e-03  -0.669 0.503761
## liveness        -8.968e-01  7.348e-01  -1.221 0.222241
## time_signature3 -2.155e+00  7.605e-01  -2.834 0.004592 **
## time_signature4 -1.972e+00  7.278e-01  -2.709 0.006746 **
## time_signature5 -2.354e-01  1.073e+00  -0.219 0.826374
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  697.85  on 3984  degrees of freedom
## AIC: 729.85
##
## Number of Fisher Scoring iterations: 8
```

```
#removing time_signature
glm.classical <- update(glm.classical, .~.-time_signature)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
##     energy + loudness + mode + speechiness + acousticness + instrumentalness +
##     valence + tempo + liveness, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1382  -0.1024  -0.0321  -0.0100   3.7819
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -3.850e+00  1.039e+00  -3.707 0.000209 ***
## popularity       -2.972e-02  5.511e-03  -5.394 6.89e-08 ***
## duration_ms       8.628e-06  1.271e-06   6.787 1.14e-11 ***
## danceability     -8.171e+00  9.244e-01  -8.839  < 2e-16 ***
## energy           -1.720e+00  8.938e-01  -1.924 0.054305 .
## loudness         -1.008e-01  3.334e-02  -3.025 0.002490 **
## mode1            -4.912e-01  2.411e-01  -2.037 0.041622 *
## speechiness       1.776e+00  1.976e+00   0.899 0.368845
## acousticness      6.673e+00  5.732e-01  11.643  < 2e-16 ***
## instrumentalness  3.109e+00  3.863e-01   8.048 8.44e-16 ***
## valence           1.951e+00  5.795e-01   3.366 0.000762 ***
## tempo            -2.650e-03  3.519e-03  -0.753 0.451394
## liveness         -9.976e-01  7.266e-01  -1.373 0.169773
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  709.96  on 3987  degrees of freedom
## AIC: 735.96
##
## Number of Fisher Scoring iterations: 8
```

```
#removing tempo
glm.classical <- update(glm.classical, .~.-tempo)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
```

```
##     energy + loudness + mode + speechiness + acousticness + instrumentalness +
##     valence + liveness, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1000  -0.1014  -0.0319  -0.0098   3.7748
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.120e+00  9.751e-01  -4.226 2.38e-05 ***
## popularity       -2.968e-02  5.507e-03  -5.389 7.08e-08 ***
## duration_ms       8.593e-06  1.267e-06   6.785 1.16e-11 ***
## danceability     -8.085e+00  9.132e-01  -8.853  < 2e-16 ***
## energy           -1.798e+00  8.854e-01  -2.031 0.042279 *
## loudness         -9.946e-02  3.311e-02  -3.004 0.002666 **
## mode1            -5.061e-01  2.403e-01  -2.106 0.035189 *
## speechiness       1.659e+00  1.980e+00   0.838 0.402001
## acousticness      6.714e+00  5.711e-01  11.756  < 2e-16 ***
## instrumentalness  3.095e+00  3.837e-01   8.066 7.24e-16 ***
## valence           1.866e+00  5.657e-01   3.298 0.000974 ***
## liveness         -9.815e-01  7.243e-01  -1.355 0.175403
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  710.53  on 3988  degrees of freedom
## AIC: 734.53
##
## Number of Fisher Scoring iterations: 8
```

```r
#removing speechiness
glm.classical <- update(glm.classical, .~.-speechiness)
summary(glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
##     energy + loudness + mode + acousticness + instrumentalness +
##     valence + liveness, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1091  -0.1020  -0.0318  -0.0094   3.7685
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.239e+00  9.678e-01  -4.380 1.19e-05 ***
## popularity       -3.048e-02  5.444e-03  -5.599 2.16e-08 ***
## duration_ms       8.623e-06  1.265e-06   6.815 9.41e-12 ***
## danceability     -7.959e+00  8.970e-01  -8.872  < 2e-16 ***
## energy           -1.572e+00  8.436e-01  -1.863 0.062460 .
## loudness         -1.031e-01  3.284e-02  -3.138 0.001702 **
```

```
## mode1            -5.098e-01  2.401e-01  -2.123 0.033764 *
## acousticness      6.771e+00  5.698e-01  11.881  < 2e-16 ***
## instrumentalness  3.060e+00  3.798e-01   8.058 7.76e-16 ***
## valence           1.860e+00  5.651e-01   3.292 0.000996 ***
## liveness         -9.734e-01  7.209e-01  -1.350 0.176931
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.9  on 3999  degrees of freedom
## Residual deviance:  711.2  on 3989  degrees of freedom
## AIC: 733.2
##
## Number of Fisher Scoring iterations: 8
```

```
#removing liveness
best.glm.classical <- update(glm.classical, .~.-liveness)
summary(best.glm.classical)
```

```
##
## Call:
## glm(formula = classical ~ popularity + duration_ms + danceability +
##     energy + loudness + mode + acousticness + instrumentalness +
##     valence, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0588  -0.1019  -0.0319  -0.0097   3.7936
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -4.450e+00  9.575e-01  -4.648 3.35e-06 ***
## popularity       -3.063e-02  5.446e-03  -5.624 1.86e-08 ***
## duration_ms       8.588e-06  1.271e-06   6.759 1.39e-11 ***
## danceability     -7.729e+00  8.768e-01  -8.815  < 2e-16 ***
## energy           -1.674e+00  8.408e-01  -1.991  0.04645 *
## loudness         -1.048e-01  3.292e-02  -3.183  0.00146 **
## mode1            -5.035e-01  2.399e-01  -2.099  0.03584 *
## acousticness      6.714e+00  5.666e-01  11.850  < 2e-16 ***
## instrumentalness  3.097e+00  3.785e-01   8.181 2.82e-16 ***
## valence           1.793e+00  5.641e-01   3.179  0.00148 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3966.91  on 3999  degrees of freedom
## Residual deviance:  713.09  on 3990  degrees of freedom
## AIC: 733.09
##
## Number of Fisher Scoring iterations: 8
```

Here we just report the steps taken for the `classical` logistic regression model. We followed the same

procedure also to find the best models for `hip-hop`, `reggaeton`, `rock-n-roll` and `techno`. Focusing on this process, it can be seen that at each step we removed the features with highest *p-value*. From model to model we haven't kept the same variables because they may act differently for different `track_genres`. For example in the first step on the feature selection for the `classical` model we removed the `explicit` feature, while for the `hip-hop` classification we kept it in the final model.

We have continued until all variables left had a low *p-value*, and we can notice how the AIC index remained more or less the same with a slow decreasing. The best we got for the model of `classical` classification was $AIC = 729$. Nonetheless we keep the last model we found. It has a slightly higher AIC index (733) but less parameters.

Diagnostic and Results

We now try to understand if there are some problems in the models we found and how they perform on the test set.

First we check for collinearity in the models by using the *VIF()* function:

```
vif(best.glm.classical)
```

```
##       popularity      duration_ms     danceability           energy
##         1.094314         1.763901         1.465845         3.046093
##         loudness             mode     acousticness instrumentalness
##         2.394715         1.091042         1.653401         1.521424
##          valence
##         2.050718
```

```
vif(best.glm.techno)
```

```
##       popularity      duration_ms         explicit     danceability
##         1.033744         1.085635         1.076483         2.063719
##           energy         loudness      speechiness     acousticness
##         4.467195         5.030994         1.193155         2.598165
## instrumentalness          valence            tempo
##         2.262015         1.609608         1.255641
```

```
vif(best.glm.rock_n.roll)
```

```
##       popularity      duration_ms         explicit     danceability
##         1.086059         1.244284         1.015403         2.026770
##           energy         loudness             mode      speechiness
##         4.344284         3.305080         1.030233         1.263234
##     acousticness instrumentalness          valence         liveness
##         1.936629         1.364589         1.974688         1.076955
```

```
vif(best.glm.reggaeton)
```

```
##                      GVIF Df GVIF^(1/(2*Df))
## popularity       1.019923  1        1.009912
## duration_ms      1.027775  1        1.013792
## explicit         1.220543  1        1.104782
```

```
## danceability     1.214570  1        1.102075
## loudness         1.086745  1        1.042470
## mode             1.036084  1        1.017882
## speechiness      1.070929  1        1.034857
## acousticness     1.205796  1        1.098087
## instrumentalness 1.023032  1        1.011450
## valence          1.166373  1        1.079987
## liveness         1.029960  1        1.014870
## time_signature   1.075287  3        1.012171
```

```
vif(best.glm.hiphop)
```

```
##       popularity         explicit     danceability           energy
##         1.061265         1.194941         1.532234         2.456156
##         loudness             mode      speechiness     acousticness
##         2.238567         1.027546         1.090170         1.618717
## instrumentalness          valence            tempo         liveness
##         1.145397         1.355747         1.092449         1.063511
```

It is possible to see that none of the models present particular problems of collinearity between the predictors. Precisely, the *Variance Inflation Factor* computed on the predictors of each model never exceed the value 5. We only have a vif equals to 5.03 in the `loudness` variable of `techno` track genre.

Let's try the best models, one by one, on the test set.

```
## [1] "CLASSICAL"
```

```
##
## logistic_pred_classical   1   0
##                       1 191  11
##                       0  22 776
```

```
## [1] "Accuracy: 0.967"
```

```
## [1] "TPR:  0.897"
```

```
## [1] "FPR:  0.014"
```

```
## [1] "TECHNO"
```

```
##
## logistic_pred_techno   1   0
##                    1 151  13
##                    0  47 789
```

```
## [1] "Accuracy: 0.94"
```

```
## [1] "TPR:  0.763"
```

```
## [1] "FPR:  0.016"
```

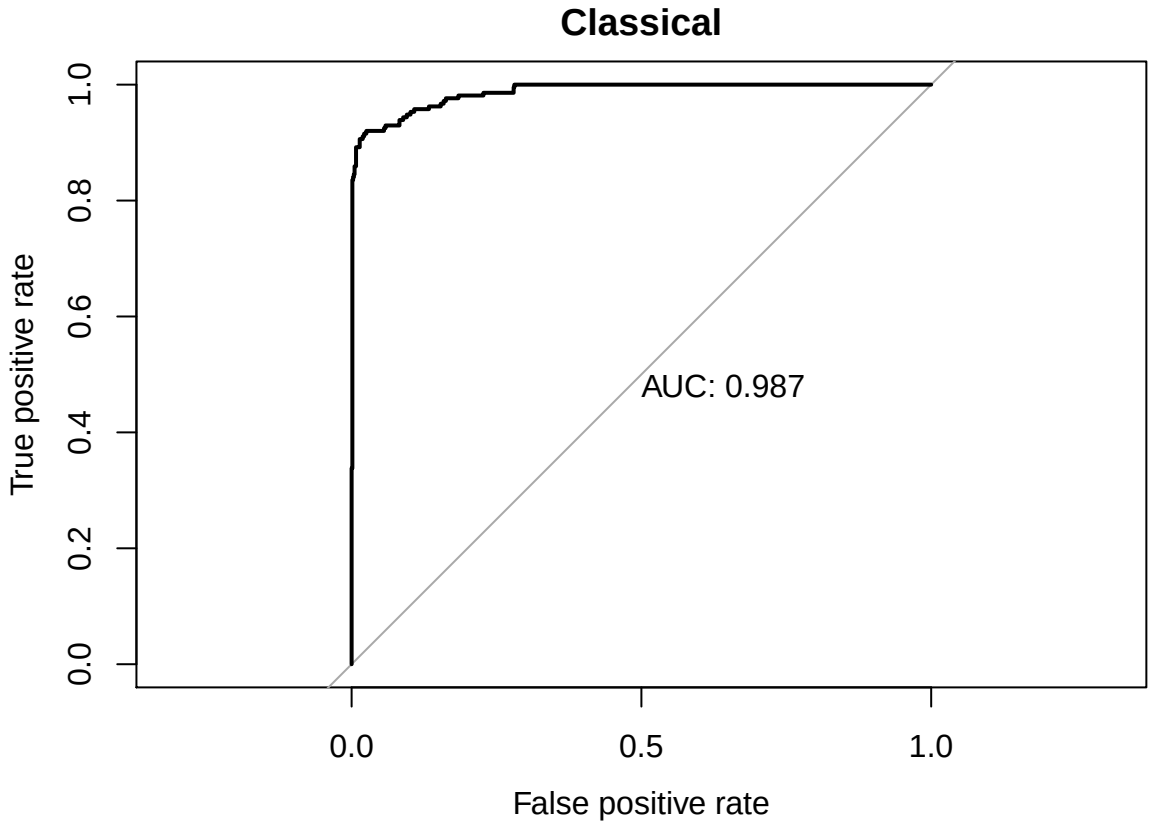
```
## [1] "HIP-HOP"

##
## logistic_pred_hiphop   1   0
##                    1  66  57
##                    0 123 754

## [1] "Accuracy: 0.82"

## [1] "TPR:  0.349"

## [1] "FPR:  0.07"

## [1] "REGGAETON"

##
## logistic_pred_reggaeton   1   0
##                       1 112  59
##                       0  85 744

## [1] "Accuracy: 0.856"

## [1] "TPR:  0.569"

## [1] "FPR:  0.073"

## [1] "ROCK-N-ROLL"

##
## logistic_pred_rock.n.roll   1   0
##                         1 155  35
##                         0  48 762

## [1] "Accuracy: 0.917"

## [1] "TPR:  0.764"

## [1] "FPR:  0.044"
```

For `classical`, `techno` and `rock-n-roll` the *accuracy* is above 90% and for `reggaeton` and `hip-hop` above 80%. At first this seems like a good result but in fact it is misleading. In fact, we train the classifiers to recognize one target genre at time, comparing it with all the others genres in the dataset. In this way, even if all the genres are equally represented in the training set, every time the target class (1) we are trying to identify is the minority class (approx 0.2 of the examples) while the "other" (0) class is the majority class (approx 0.8 of the examples).

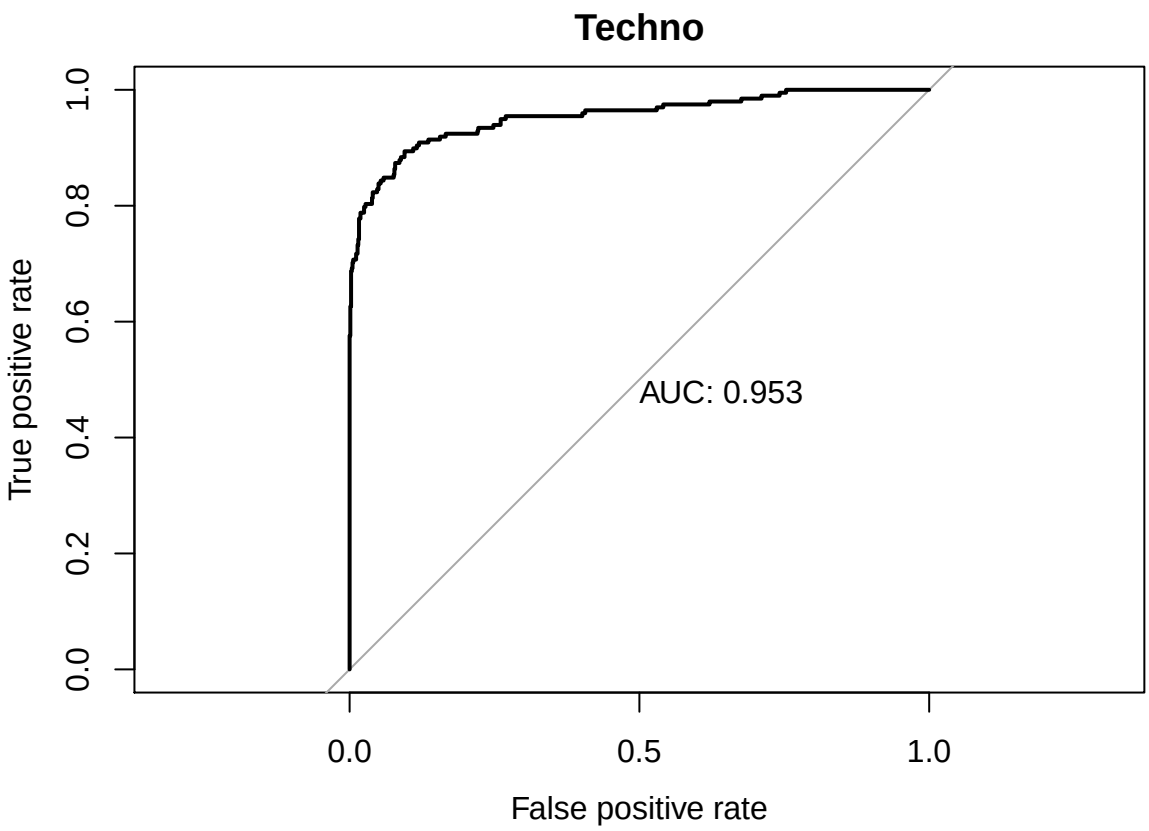
In this context hence the accuracy is not a reliable metric. The *True and False Positive rates* of the classifiers may be more explanatory. While the FPR is below 10% for all the classifiers the TPR does not behave very well. In fact, despite for `classical` music which is around 90%, it shows mediocre performances for `techno` and `rock-n-roll` (around 77%) and poor results for `reggaeton` and `hip-hop`, respectively around 56% and 35%.

ROC curves

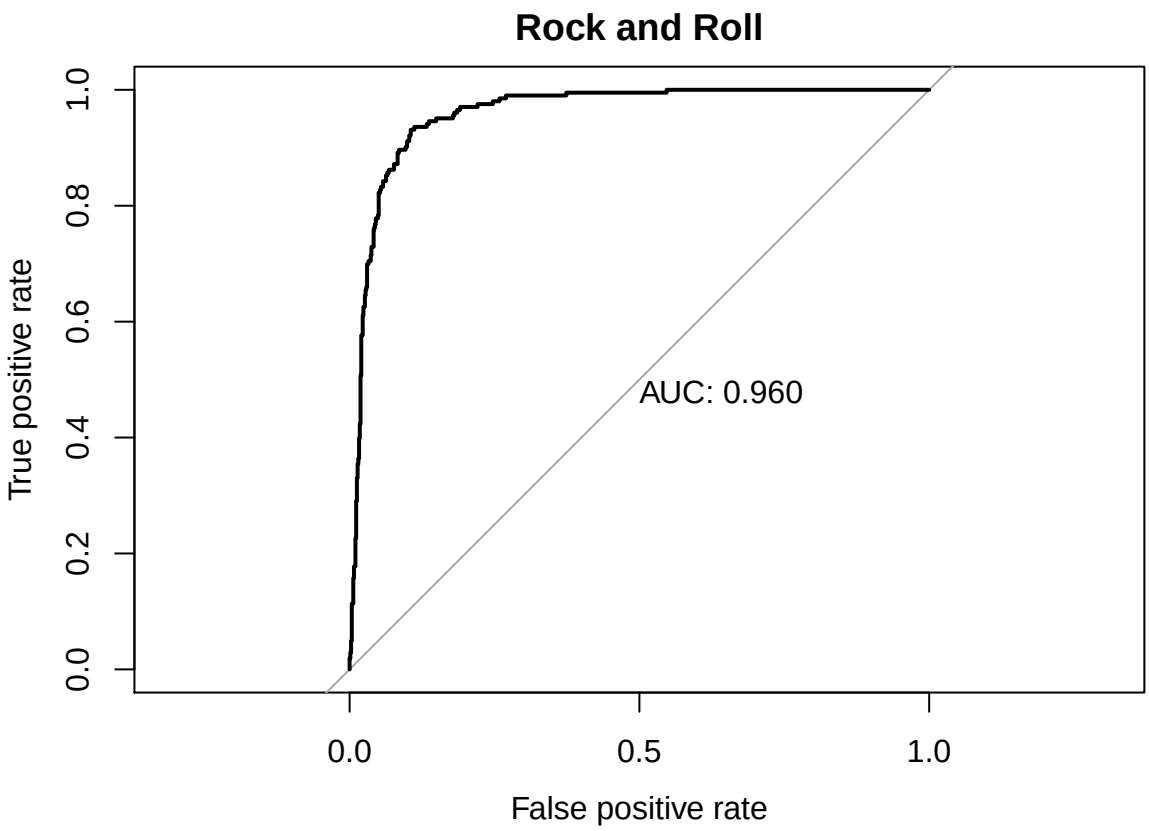
We introduce now the *Receiver Operating Characteristic (ROC)* curves used to visualize the performance of ours classification models. They are obtained by plotting the *False Positive Rate* against the *True Positive Rate* on the cartesian reference system. They will be also used on the last paragraph to compare the models and understand which is the best by seeing the area under the curve: greater the area, better the model's performance.



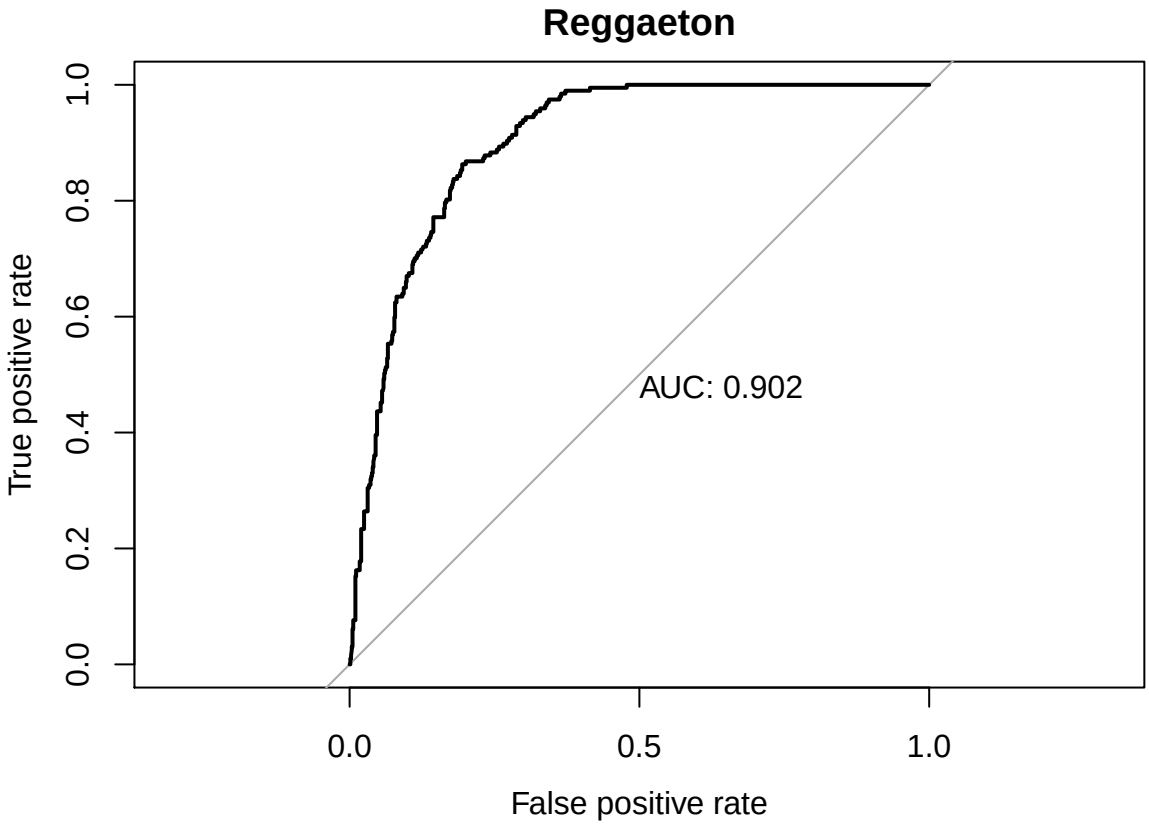

```
## Area under the curve: 0.9866
```



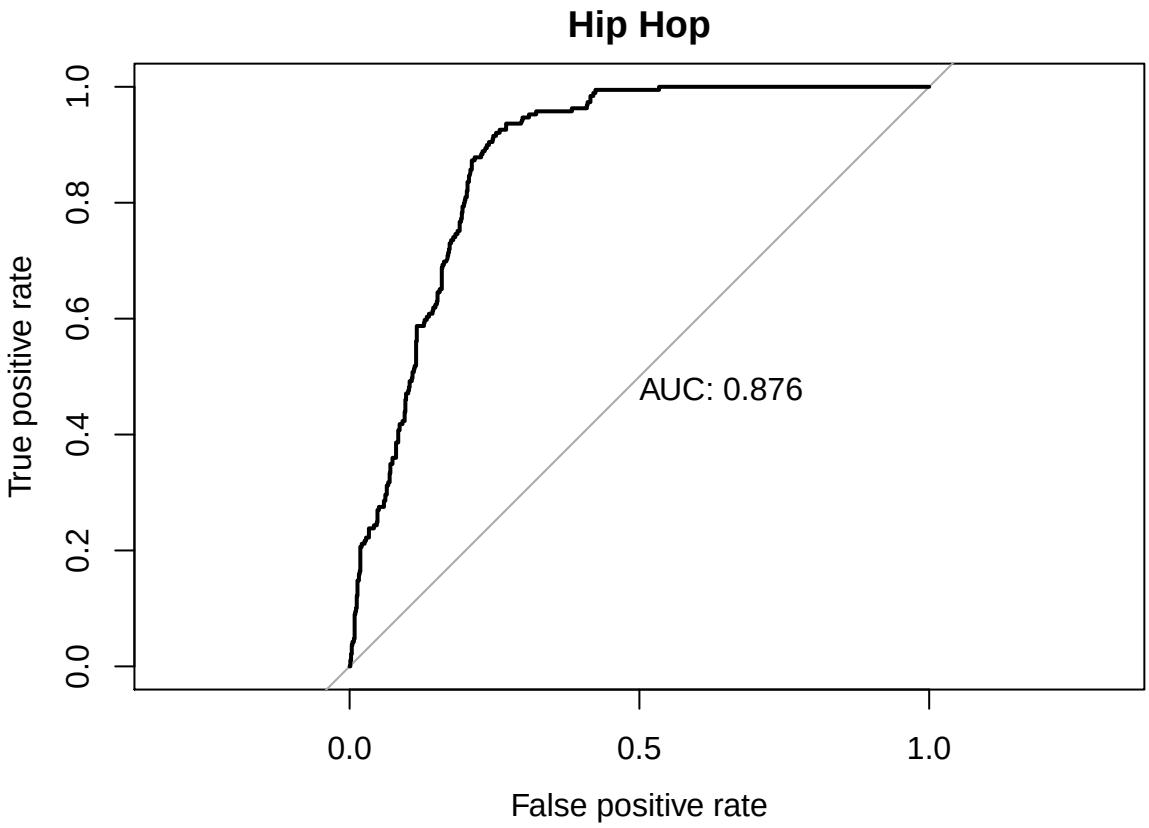

```
## Area under the curve: 0.9534
```




```
## Area under the curve: 0.9601
```




```
## Area under the curve: 0.9022
```




```
## Area under the curve: 0.8761
```

By comparing the area under the curves we can immediately see that the logistic model built to classify `classical` genre works really well, while the one for `hip-hop` has the worse performance. Let's see if we can improve it by adjusting the unbalance in the training set.

Dealing with unbalance of the training set

In order to overcome the unbalancing in the data we decided to compare three approaches: *undersampling*, *oversampling* and a *mixed strategy*. As target music genre for the comparison we used `hip-hop`, whose classifier had the worst TPR.

Undersampling We first perform undersampling by balancing the number of `hip-hop` tracks (class 1) with the number of tracks of other music genres (class 0), by taking a sample of the latter of the same length of class 1.

```
# Undersampling with a perfect balance between
# hiphop class (1) and "other" class (0)
other_class <- train[train$hiphop == 0, ]
# size of the 1 class
subsample_size <- nrow(train[train$hiphop == 1, ])
set.seed(123) # set a seed for reproducibility
other_class_subsample <- other_class[sample(nrow(other_class),
                                            subsample_size), ]
```

```
undersample.balanced.data.hiphop <- rbind(other_class_subsample,
                                          train[train$hiphop == 1, ])

best.glm.hiphop.undersampling <- glm(hiphop ~ popularity
                                          + explicit + danceability
                                          + energy + +mode
                                          + speechiness
                                          + acousticness
                                          + instrumentalness
                                          + valence
                                          +tempo
                                          +liveness,
                                          data=undersample.balanced.data.hiphop,
                                          family = binomial)

#Here we check if the performance on the test set improved
logistic_prop_hiphop_under <- predict(
  best.glm.hiphop.undersampling, test, type = "response")
logistic_pred_hiphop_under <- rep(0, 1000)
logistic_pred_hiphop_under[logistic_prop_hiphop_under > 0.5] <-1
hiphop_table_under <- table(logistic_pred_hiphop_under,
                            test$hiphop)
hiphop_table_under <- hiphop_table_under[2:1, 2:1]
tpr_for_hiphop_under <- hiphop_table_under[1,1]/
  sum(hiphop_table_under[,1])
fpr_for_hiphop_under <- hiphop_table_under[1,2]/
  sum(hiphop_table_under[,2])
print("HIP-HOP with undersampling")
```

```
## [1] "HIP-HOP with undersampling"
```

```
print(paste("TPR: ", round(tpr_for_hiphop_under,3)))
```

```
## [1] "TPR:  0.905"
```

```
print(paste("FPR: ", round(fpr_for_hiphop_under,3)))
```

```
## [1] "FPR:  0.244"
```

The use of an undersampling approach to train a new model improves a lot the TPR for `hip-hop`: it goes from approx 0.35 to approx. 0.90, while affects a bit the fpr that increases from approx. 0.02 to approx 0.25. In this case in the training set we used around 2000 samples, 1000 for each class.

Oversampling We now try oversampling by taking two times the set of the minority class, i.e. class 1, the `hip-hop` tracks.

```
###Now we try the oversample of the minority class just by
#duplicating the elemnts belonging to the hiphop class.
other_class <- train[train$hiphop == 0, ]
balanced_data <- rbind(other_class, train[train$hiphop == 1, ],
```

```
                        train[train$hiphop == 1, ])

best.glm.hiphop_with_oversampling <- glm(hiphop ~ popularity +
                                           explicit +
                                           danceability +
                                           energy +
                                           loudness + mode +
                                           speechiness +
                                           acousticness +
                                           instrumentalness +
                                           tempo + liveness,
                                           data=balanced_data,
                                           family = binomial)

#Checking if perfromances improved
logistic_prop_hiphop_over <- predict(best.glm.hiphop_with_oversampling,
                                     test, type = "response")
logistic_pred_hiphop_over <- rep(0, 1000)
logistic_pred_hiphop_over[logistic_prop_hiphop_over > 0.5] <- 1
hiphop_table_over <- table(logistic_pred_hiphop_over,
                           test$hiphop)
hiphop_table_over <- hiphop_table_over[2:1, 2:1]
tpr_for_hiphop_over <- hiphop_table_over[1,1]/
  sum(hiphop_table_over[,1])
fpr_for_hiphop_over <- hiphop_table_over[1,2]/
  sum(hiphop_table_over[,2])
print("HIP-HOP with oversampling")
```

```
## [1] "HIP-HOP with oversampling"
```

```
print(paste("TPR: ", round(tpr_for_hiphop_over,3)))
```

```
## [1] "TPR:  0.651"
```

```
print(paste("FPR: ", round(fpr_for_hiphop_over,3)))
```

```
## [1] "FPR:  0.15"
```

Oversampling the minority class improved slightly the TPR, which became around 65%, but the improvement was not comparable with the undersampling. The proportion of elements of the two class is 2000 samples for class 1 and 4000 for class 0.

Mixed approach We now perform an oversampling of the minority class and an undersampling of the majority class contemporanearly to have a balanced dataset between the 2 class.

```
#Mixed approach: oversampling class 1 and undersampling class 0
other_class <- train[train$hiphop == 0, ]
subsample_size <- nrow(train[train$hiphop == 1, ])
set.seed(123)
other_class_subsample <- other_class[sample(nrow(other_class),
```

```
                                            subsample_size*2), ]
balanced_data <- rbind(other_class_subsample,
                       train[train$hiphop == 1, ],
                       train[train$hiphop == 1, ])

best.glm.hiphop_with_mixedsampling <- glm(hiphop ~ popularity +
                                          explicit +
                                          danceability +
                                          energy +
                                          loudness + mode +
                                          speechiness +
                                          acousticness +
                                          instrumentalness +
                                          tempo + liveness,
                                          data=balanced_data,
                                          family = binomial)

#Checking if perfromances improved
logistic_prop_hiphop_mixed <- predict(
  best.glm.hiphop_with_mixedsampling, test, type = "response")
logistic_pred_hiphop_mixed <- rep(0, 1000)
logistic_pred_hiphop_mixed[logistic_prop_hiphop_mixed  > 0.5] <- 1
hiphop_table_mixed <- table(logistic_pred_hiphop_mixed,
                            test$hiphop)
hiphop_table_mixed <- hiphop_table_mixed[2:1, 2:1]
tpr_for_hiphop_mixed <- hiphop_table_mixed[1,1]/
  sum(hiphop_table_mixed[,1])
fpr_for_hiphop_mixed <- hiphop_table_mixed[1,2]/
  sum(hiphop_table_mixed[,2])
print("HIP-HOP with mixed approach")
```

```
## [1] "HIP-HOP with mixed approach"
```

```
print(paste("TPR: ", round(tpr_for_hiphop_mixed,3)))
```

```
## [1] "TPR:  0.931"
```

```
print(paste("FPR: ", round(fpr_for_hiphop_mixed,3)))
```

```
## [1] "FPR:  0.249"
```

We can see that this mixed approach shows performance pretty similar to the undersampling, with a slightly higher TPR but also a slighty higher FPR. Here we had 4000 samples in total, 2000 for each class.

Undersampling for other genres We now use the undersampling approach, being it the most straightforward, to build new models for each genres, despite for `classical` that already had an high TPR.

```
## [1] "REGGAETON with undersampling"
```

```
## [1] "TPR:  0.904"
```

```
## [1] "FPR:  0.217"

## [1] "ROCK N ROLL with undersampling"

## [1] "TPR:  0.876"

## [1] "FPR:  0.054"

## [1] "TECHNO with undersampling"

## [1] "TPR:  0.874"

## [1] "FPR:  0.076"
```

As can be seen from the values of TPR and FPR the models have improved their performance on the test by undersampling the majority class.

Classification

Now we want to get for each track in the test set a unique prediction for its genre. As mentioned before, to do that we compare the probabilities given by the classifiers and pick the larger one. In this way we classify the track on the basis of the classifier that has the strongest confidence for it.

```
compare <- function(){
  classical.prob <- predict(best.glm.classical, test,
                            type = "response")
  techno.prob <- predict(best.glm.techno.undersampling,
                         test, type = "response")
  rnr.prob <- predict(best.glm.rock.undersampling, test,
                      type = "response")
  reggaeton.prob <- predict(best.glm.reggaeton.undersampling,
                            test, type = "response")
  hiphop.prob <-predict(best.glm.hiphop.undersampling,
                        test, type = "response")

  # Create a matrix with the five vectors as
  #levels(spotify_dummies$track_genre)
  #classical =  1
  #hiphop =     2
  #reggaeton =  3
  #rocknroll =  4
  #techno =     5
  prob.matrix <- cbind(classical.prob, hiphop.prob,
                       reggaeton.prob, rnr.prob, techno.prob)
  genres <- levels(test$track_genre)

  track_genre.pred <- rep(0, length(test))

  # Loop over the indices of the vectors
  for (i in 1:length(classical.prob)) {
    # Find the maximum value at this index across all vectors
```

```
    max_value <- max(prob.matrix[i, ])

    for (j in 1:ncol(prob.matrix)) {
      if (prob.matrix[i, j] == max_value) {
        track_genre.pred[i] <- genres[j]
      }
    }
  }

  return(track_genre.pred)
}

track_genre.pred <- compare()

cm <- table(track_genre.pred, test$track_genre)
print(cm)
```

```
##
## track_genre.pred classical hip-hop reggaeton rock-n-roll techno
##      classical         175       0         1           4      0
##      hip-hop             4     117        64           5     21
##      reggaeton           0      53       123           4     13
##      rock-n-roll        31       7         7         189      8
##      techno              3      12         2           1    156
```

Results interpretation

We proceed to analyze the performance of our one-vs-all approach to this classification problem.

First let's compute the overall accuracy:

```
cm.accuracy <- sum(diag(cm))/sum(cm)
print(paste('Overall Accuracy =', cm.accuracy))
```

```
## [1] "Overall Accuracy = 0.76"
```

Then we compute precision, recall (TPR) and F1-score for each class:

```
#Classical
classical.precision <- cm[1,1]/sum(cm[1,])
classical.recall <- cm[1,1]/sum(cm[,1])
classical.f1 <- 2*((classical.precision * classical.recall)/
                     (classical.precision + classical.recall))

#Hip-hop
hiphop.precision <- cm[2,2]/sum(cm[2,])
hiphop.recall <- cm[2,2]/sum(cm[,2])
hiphop.f1 <- 2*((hiphop.precision * hiphop.recall)/
                  (hiphop.precision + hiphop.recall))

#Reggaeton
reggaeton.precision <- cm[3,3]/sum(cm[3,])
```

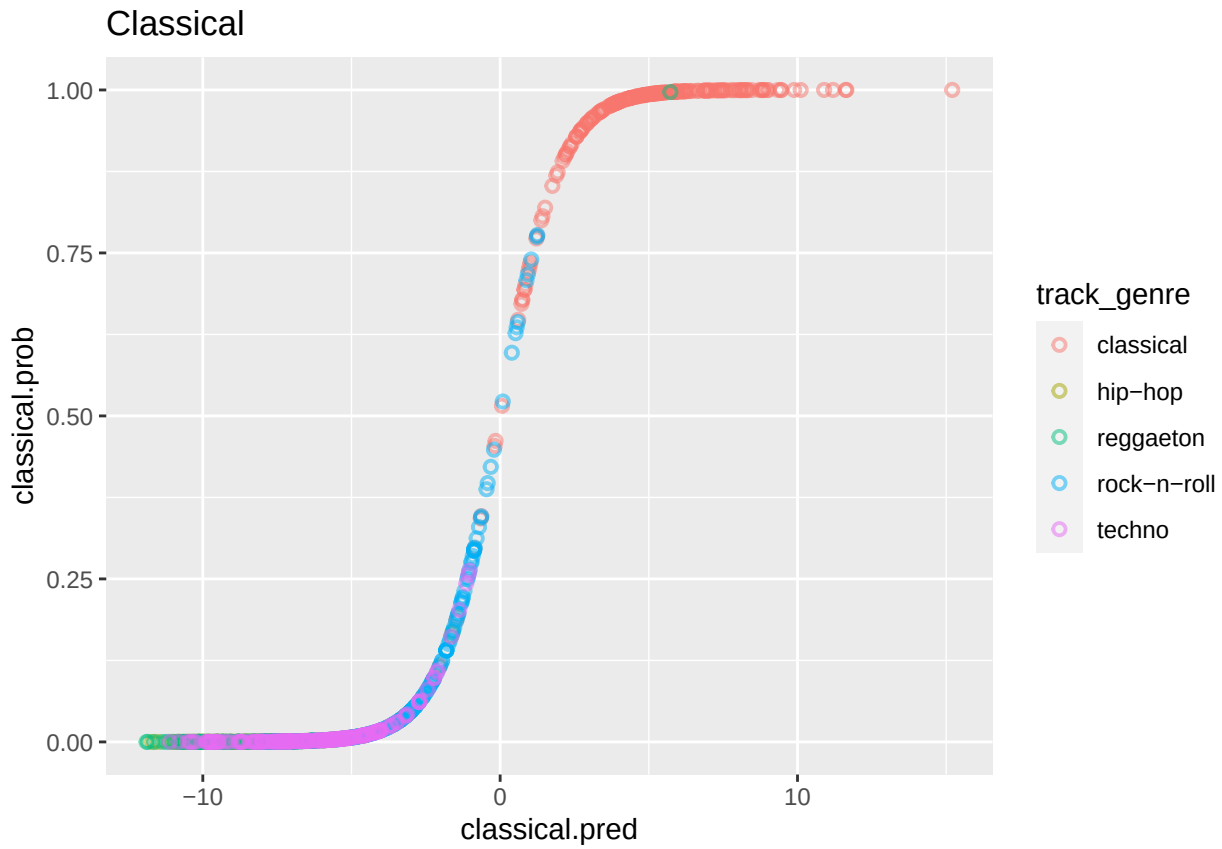
```
reggaeton.recall <- cm[3,3]/sum(cm[,3])
reggaeton.f1 <- 2*((reggaeton.precision * reggaeton.recall)/
                     (reggaeton.precision + reggaeton.recall))

#Rock and roll
rnr.precision <- cm[4,4]/sum(cm[4,])
rnr.recall <- cm[4,4]/sum(cm[,4])
rnr.f1 <- 2*((rnr.precision * rnr.recall)/(rnr.precision +
                                             rnr.recall))

#Techno
techno.precision <- cm[5,5]/sum(cm[5,])
techno.recall <- cm[5,5]/sum(cm[,5])
techno.f1 <- 2*((techno.precision * techno.recall)/
                  (techno.precision + techno.recall))

metrics <- data.frame(
  class <- c(levels(test$track_genre)),
  precision <- round(c(classical.precision,
                       hiphop.precision, reggaeton.precision,
                       rnr.precision, techno.precision), 2),
  recall <- round(c(classical.recall, hiphop.recall,
                    reggaeton.recall, rnr.recall, techno.recall),
                  2),
  f1 <- round(c(classical.f1, hiphop.f1, reggaeton.f1, rnr.f1,
                techno.f1), 2)
)
colnames(metrics) <- c("Genre", "Precision", "Recall(TPR)", "F1-score")

#Here are the performance
metrics
```

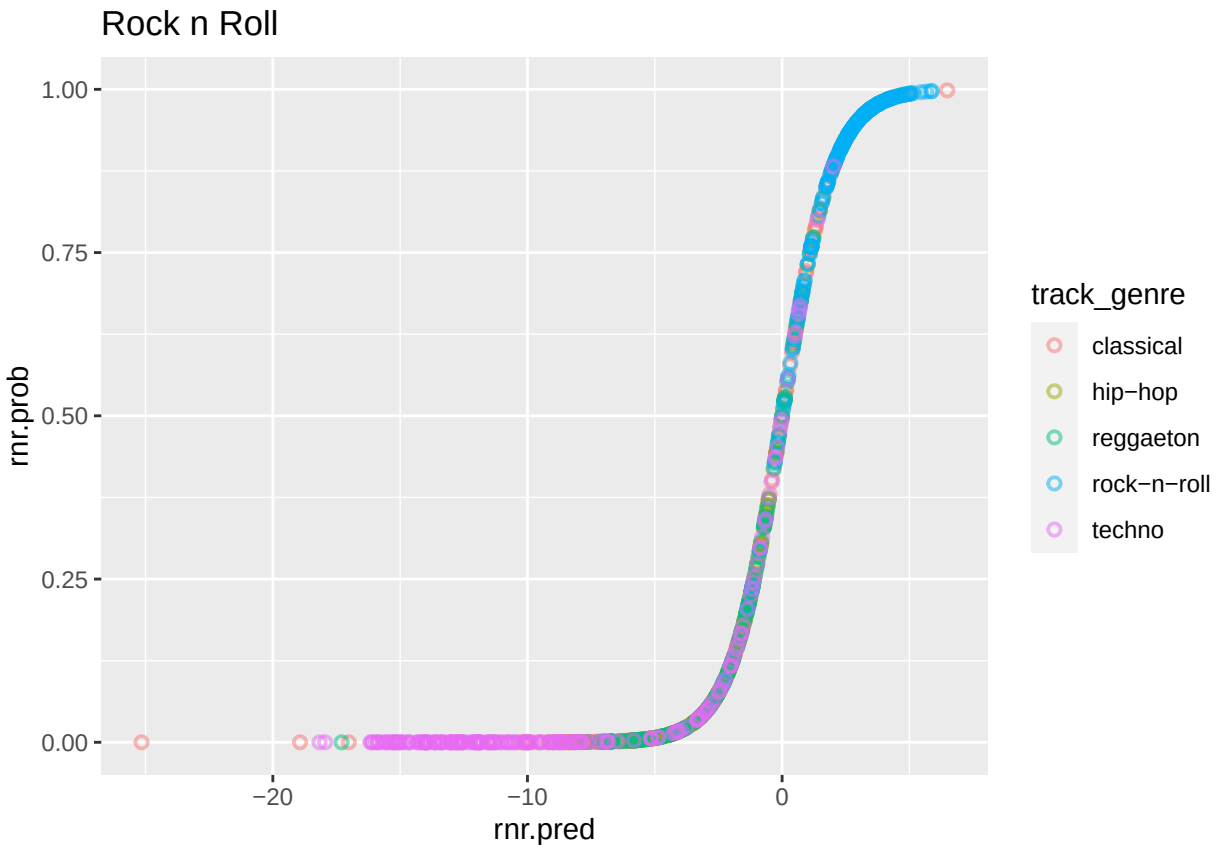
```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.97        0.82     0.89
## 2     hip-hop      0.55        0.62     0.58
## 3   reggaeton      0.64        0.62     0.63
## 4 rock-n-roll      0.78        0.93     0.85
## 5      techno      0.90        0.79     0.84
```

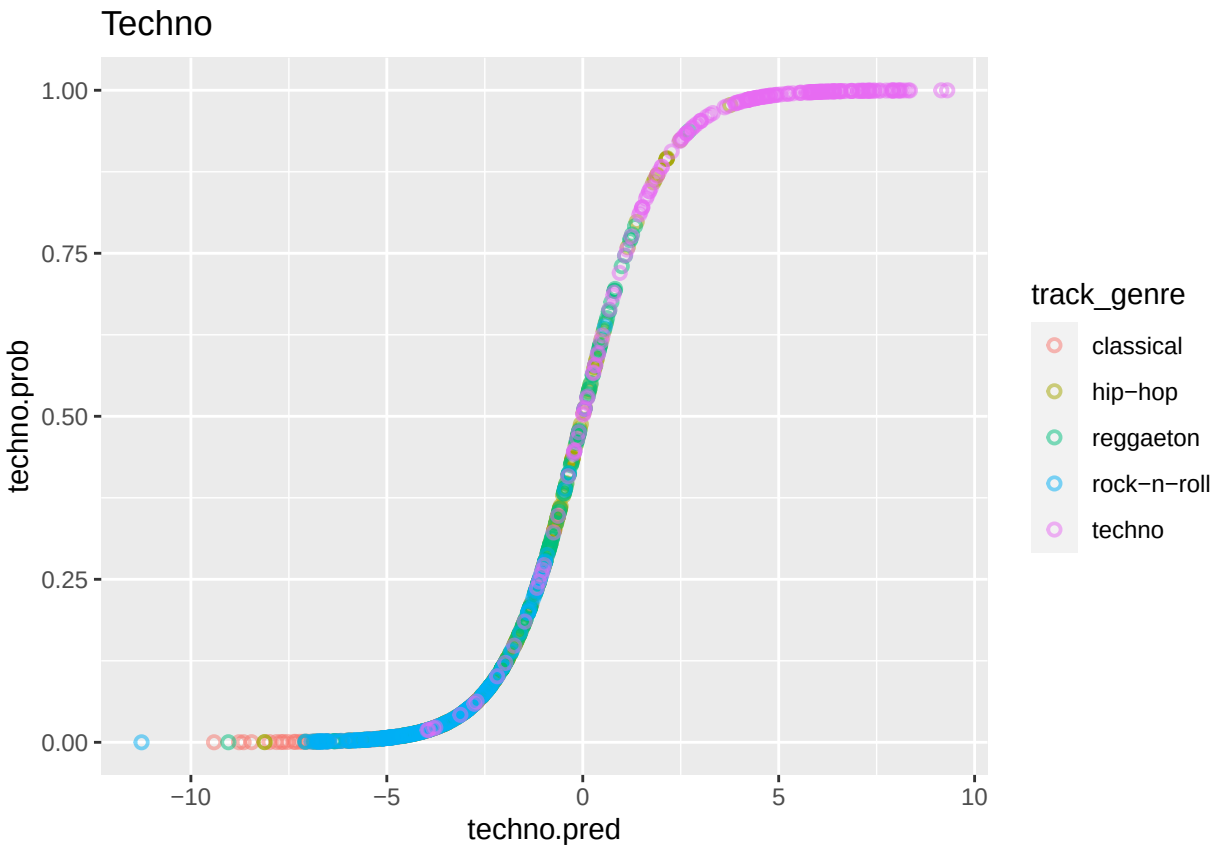
The *F1 score* provides and overview of the quality of the classification for each class. As we can see the highest scores are obtained for `classical`, `rock-n-roll` and `techno` while `reggaeton` and `hip-hop` obtain quite poor results. Looking at precision and recall allow to grasp more details on the behavior of our method which changes between different genres.

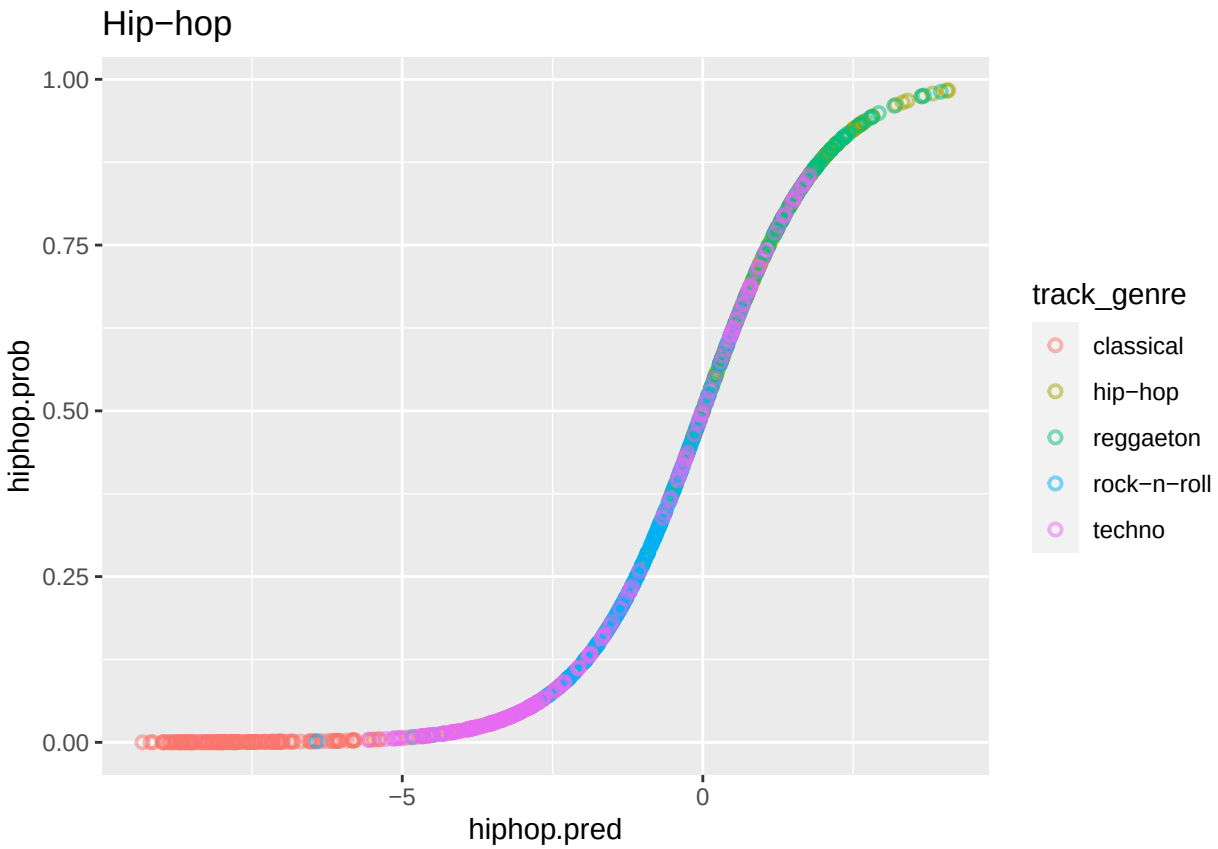
Classic music has the highest score for *precision* meaning that almost every time the methods classify a track as `classical` it does the right choice. *Recall* is a bit lower (0.82) and this indicate that our method is missing some `classical` tracks. The consideration are pretty similar for `techno` music. For `Rock-n-Roll` instead the behavior is the opposite: *recall* is high (0.94) meaning that our method is classifying correctly most rock tracks but at the same time, being precision low, it can be seen that it is overestimating the presence of rock in the test set. In other words, it is classifying more tracks as rock than what is present in our data. `Hip-hop` and `reggaeton` shows very similar score in each metrics moreover, by looking at the confusion matrix, it is possible to see that between those genres there is the greatest misclassification. There are a consistent number of `hip-hop` track that are classified as `reggaeton` and viceversa.

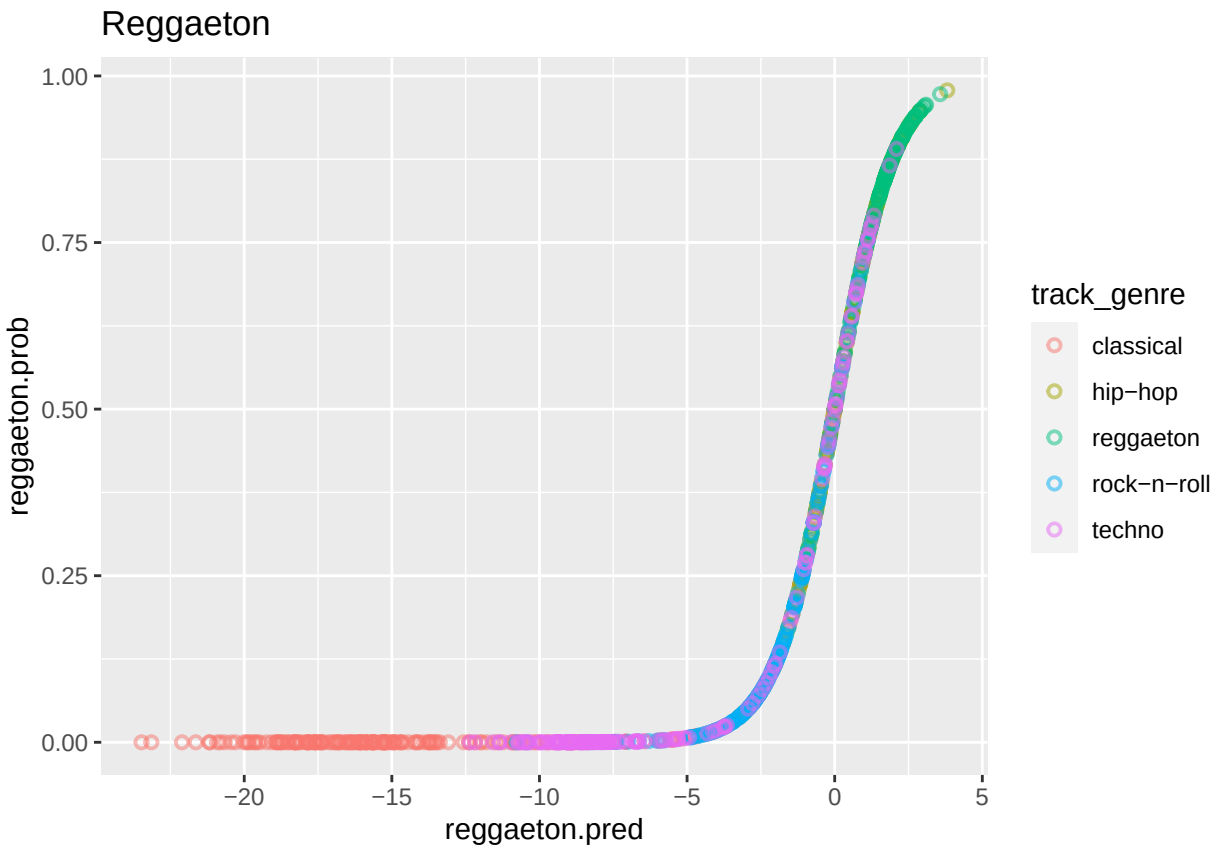
Let's try to understand better our results.




Rock n Roll
1.00
0.75
0.50
0.25
0.00
rnr.prob
−20
−10
0
rnr.pred
track_genre
classical
hip−hop
reggaeton
rock−n−roll
techno


Techno
1.00
0.75
0.50
0.25
0.00
techno.prob
−10
−5
0
5
10
techno.pred
track_genre
classical
hip−hop
reggaeton
rock−n−roll
techno


Hip−hop
1.00
0.75
0.50
0.25
0.00
hiphop.prob
−5
0
hiphop.pred
track_genre
classical
hip−hop
reggaeton
rock−n−roll
techno



We proceeded to plot the probabilities of the predictions given by each classifiers to try to understand the results we got from our method. The graphs show that the genres with the higher F1-scores have lot of points (predictions) associated with high probabilities and also a few points concentrated around the decision boundary. This is the case of `Classical` and `Techno` for example for which it can be seen that the classifiers assign the highest probabilities to points actually belonging to the correct classes. This could be interpreted as a strong confidence of the classifier, and thus of the ensemble method, when making predictions. For `hip-hop` and `reggaeton`, which have low values for the F1-score, the graphs are clearly different. It can be seen how there are fewer points associated with high probabilities and there is a great concentration of points on the middle of the curve. There is also a strong overlapping of points belonging to the two different classes in the top of the curve. This can be seen somehow as a greater uncertainty of the classifiers with respect to the ones for `Classical`, `Techno` and `Rock-n-roll`. Thus, even if the classifiers by them self shows good results when they are compared with the others to decide which genre to pick among all it is more likely that there is a misclassification, especially between `hip-hop` and `reggaeton`.

Ridge and Lasso Logistic Regression

We now want to compare whether *ridge* or *lasso* regularization could yield to better models than the ones we obtained through variables selection. To do so we will train a lasso and ridge regularized version of the logistic models we built previously. For each genres we will use the same training data we used for logistic regression, in the sense that, besides for `classical`, we will use the balanced data used in the undersampling approach. This is done to allow for a fair comparison of the logistic models.

Lasso and Ridge are two different statistical regularization methods that allow us to compute an automatic selection of variables, operating shrinkage on the coefficients of the predictors in such a way that they assume values very close to zero (Ridge) or even zero (Lasso).

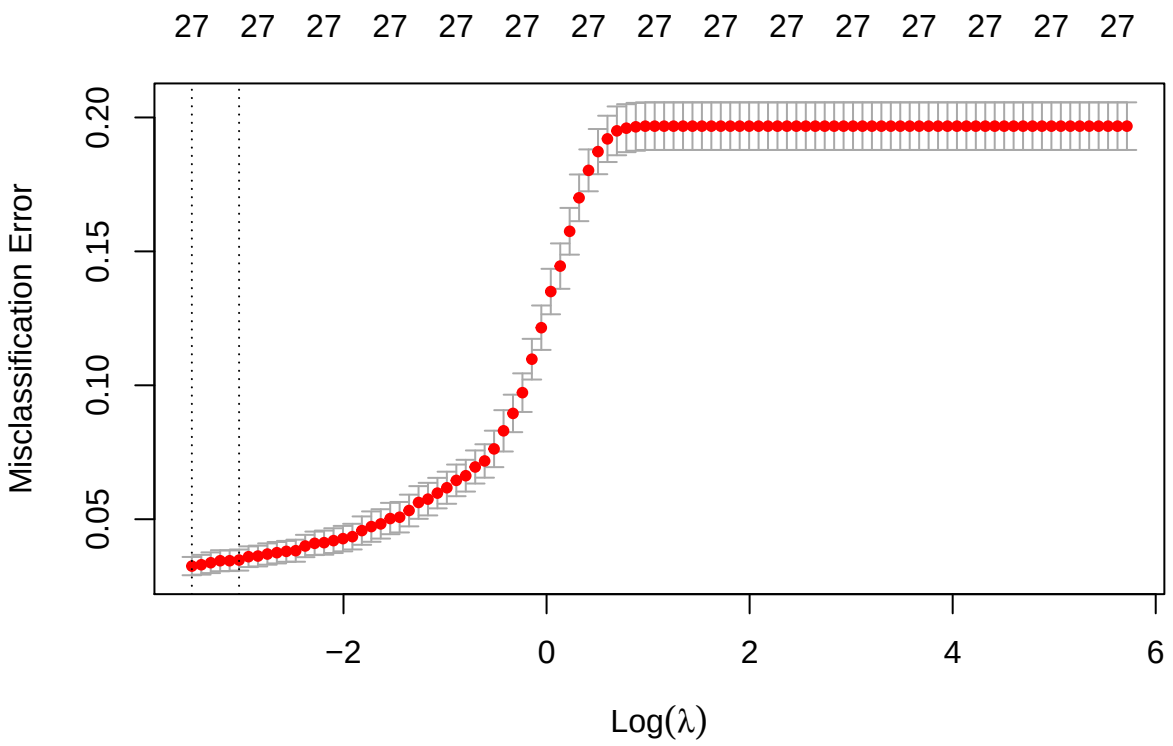
Ridge Regression

First we try to use the ridge regularization.

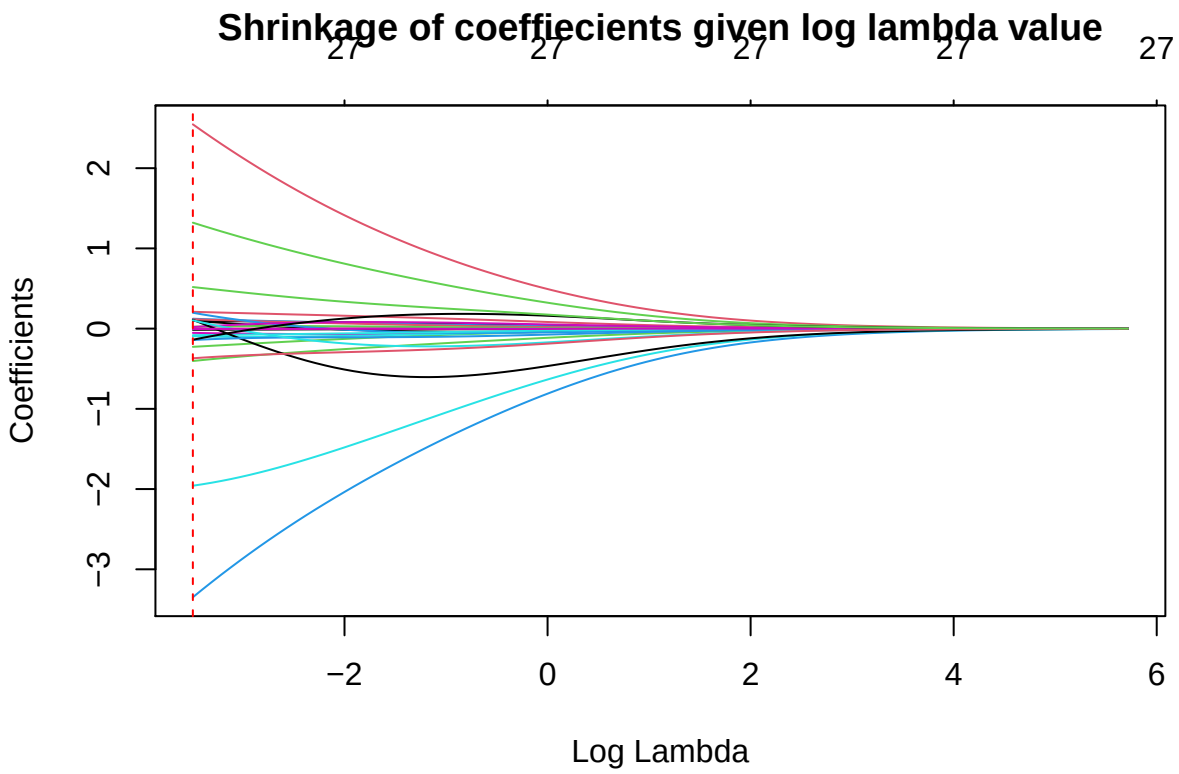
We use the `cv.glment` function from the `MASS` package to perform cross validation on the train set and choose the best value for the regularization parameter.

Here we show the code for the `classical` model, the code is the same for other genres.

```
#Classical

#Preparing the data to be used with glmnet functions
train.classical <- train[,1:17]
test.classical <- test[,1:17]
train.classical <- train.classical[,-16]
test.classical <- test.classical[,-16]
X.classical.train <- model.matrix(classical ~ ., data =
                                    train.classical)
X.classical.test <- model.matrix(classical ~ ., data =
                                   test.classical)
X.classical.train <- X.classical.train[,-1]
X.classical.test <- X.classical.test[,-1]

#performing cv
classical.ridge.cv <- cv.glmnet(X.classical.train,
                                train$classical,
                                alpha = 0,
                                family = "binomial",
                                type.measure = "class")

plot(classical.ridge.cv) #?
```




```
best.lambda.ridge.classical <- classical.ridge.cv$lambda.min
```



We can now use the value of lambda found by the cross-validation procedure to make prediction on the test set.

```
#prediction
ridge.prediction.classical <- predict(classical.ridge.cv,
                                      X.classical.test,
                                      s = best.lambda.ridge.classical,
                                      type = "class")
classical.ridge.table <- table(test$classical,
                               ridge.prediction.classical)
classical.ridge.table <- classical.ridge.table[2:1, 2:1]
classical.ridge.table
```

```
##    ridge.prediction.classical
##       1   0
##   1 184  29
##   0   7 780
```

Recall the performance of the classical logistic classifier obtained with variable selection.

```
classical_table
```

```
##
## logistic_pred_classical   1   0
##                       1 191  11
##                       0  22 776
```

We can see that the results are pretty similar to our previous model, with a shift between the false positive and false negative.

We now build the ridge version of the classifiers also for the other genres and compare them as we did previously. We don't show the code for the sake of readability.

Here we show the confusion matrix for the predictions of the models trained with ridge regularization. We will return on these results later.

```
##
## track_genre.pred.ridge classical hip-hop reggaeton rock-n-roll techno
##             classical         164       0         0           1      0
##             hip-hop             2     117        66           5     21
##             reggaeton           0      53       121           8     14
##             rock-n-roll        43       7         7         189     11
##             techno              4      12         3           0    152
```

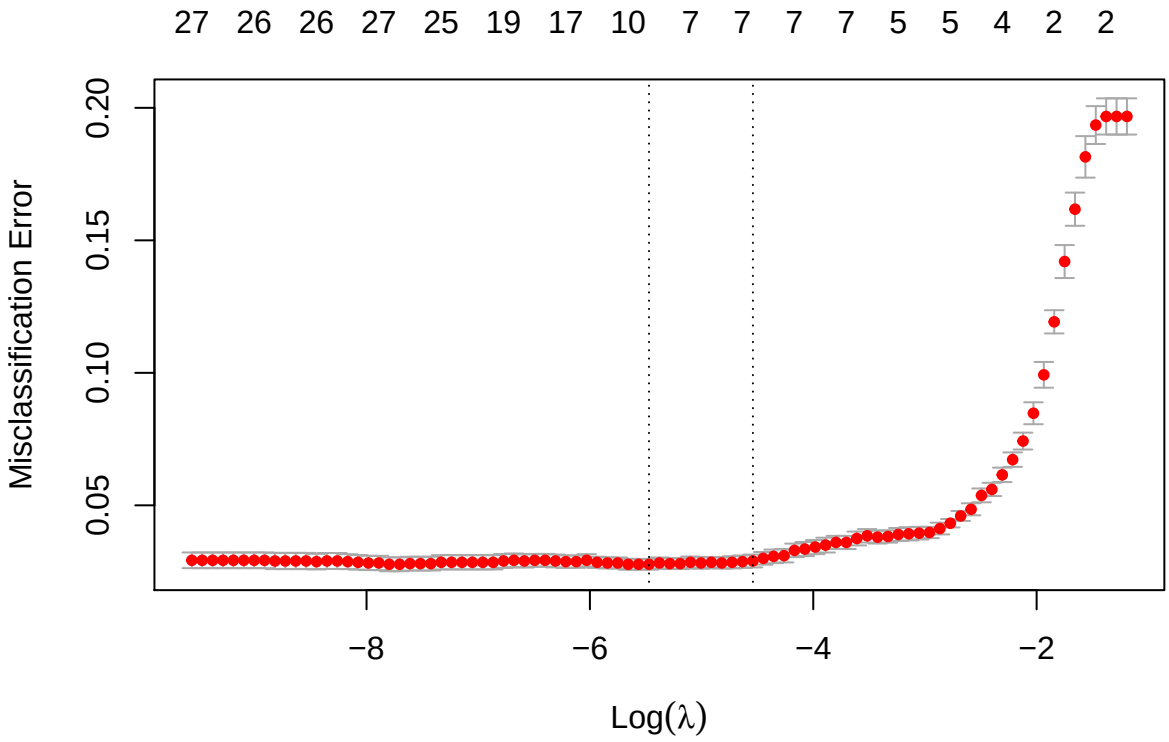
```
## [1] "Overall Accuracy = 0.743"
```

```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.99        0.77     0.87
## 2     hip-hop      0.55        0.62     0.58
## 3   reggaeton      0.62        0.61     0.62
## 4 rock-n-roll      0.74        0.93     0.82
## 5      techno      0.89        0.77     0.82
```

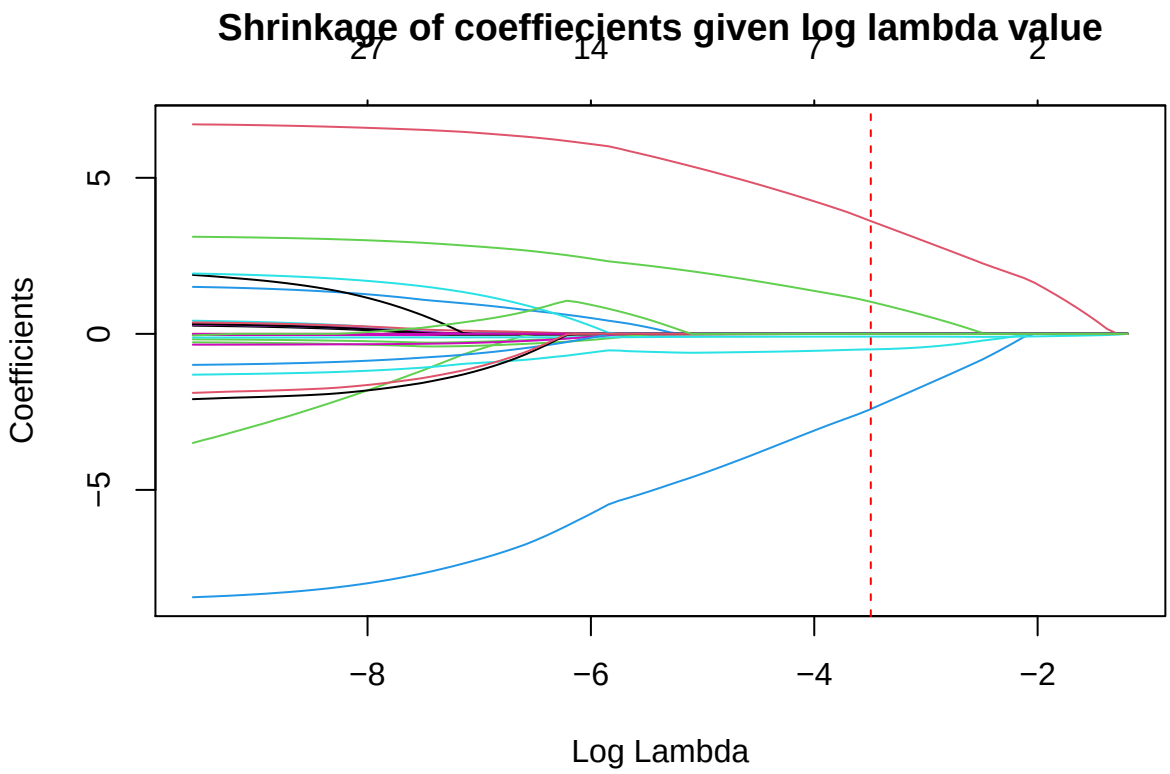
Lasso

We now perform the same task using LASSO regression and cross-validation.

```
#Data has already been pre-processed for ridge regression

#Classical

#performing cv
classical.lasso.cv <- cv.glmnet(X.classical.train,
                                train$classical,
                                alpha = 1,
                                family = "binomial",
                                type.measure = "class")

plot(classical.lasso.cv)#?
```




```
best.lambda.lasso.classical <- classical.lasso.cv$lambda.min
```



As we did before we can use the best lambda selected with cross-validation to make predictions.

```
#prediction
lasso.prediction.classical <- predict(classical.lasso.cv,
                                      X.classical.test,
                                      s = best.lambda.lasso.classical,
                                      type = "class")
lasso.classical.table <- table(test$classical,
                               lasso.prediction.classical)
lasso.classical.table <- lasso.classical.table[2:1, 2:1]
lasso.classical.table
```

```
##    lasso.prediction.classical
##       1   0
##   1 191  22
##   0  10 777
```

As before we do the same process for the other genres without showing the code.

Here we show the confusion matrix for the predictions made by comparing the logistic models built with LASSO regularization. The predictions were obtained as before by choosing the prediction from the classifier with the higher confidence.

```
##
## track_genre.pred.lasso classical hip-hop reggaeton rock-n-roll techno
##            classical         176       0         0           2      0
```

```
##            hip-hop             4     101        52           5     21
##            reggaeton           1      76       137           6     18
##            rock-n-roll        29       4         6         188      7
##            techno              3       8         2           2    152
```

We proceed to compute the metrics for the quality of the predictions.

```
## [1] "Overall Accuracy = 0.754"
```

```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.99        0.83     0.90
## 2     hip-hop      0.55        0.53     0.54
## 3   reggaeton      0.58        0.70     0.63
## 4 rock-n-roll      0.80        0.93     0.86
## 5      techno      0.91        0.77     0.83
```

Discriminant Analysis: LDA and QDA

Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) are statistical techniques used for classification. They both try to approximate the Bayes classifier by using an estimation of the density distribution of each feature in each class to compute the probability of a sample to belong to a certain class. LDA and QDA are two different, yet similar, ways of computing the conditional probabilities of a sample to belong to a certain class. In the end both method consist in computing *discriminant functions* that are used to assign an example to one of the classes.

The assumptions made by the Linear Discriminant Analysis are:

• **Multinormality of the predictors**: LDA assumes that the predictors are distributed as a *Multivariate Gaussian* given the class. The mean is specific is to the class.

• **Common Variance**: the variance is common between all classes.

• **Linearly separated classes**: the discriminant functions computed by LDA is linear and thus the accuracy of the results is better the more the data are linearly separable.

Like LDA, Quadratic Discriminant Analysis assume that the predictors are drawn from a multivariate gaussian but it also assumes that each class has his own covariance matrix. Moreover the discriminant functions used by the QDA are quadratic and this allows the method to be more accurate at the expense of an higher variance.

Shapiro - Wilks test for normality

Both LDA and QDA assume that the features are normally distributed in each target class. In order to check whether this assumption is satisfied by our training data we perform the *Shapiro - Wilks test*.

```
#We define the subset of features we want to test
subset.features <- c("popularity", "duration_ms",
                     "danceability",
                     "energy", "loudness", "speechiness",
                     "acousticness", "instrumentalness",
                     "liveness","valence", "tempo")
#Classical
classical.data <- train[train$track_genre == "classical",
                        subset.features]
```

```r
shapiro.results <- lapply(classical.data, shapiro.test)
p.values <- sapply(shapiro.results, function(x) x$p.value)
print(p.values)
```

```
##       popularity      duration_ms     danceability           energy
##     6.555819e-33     1.197661e-36     8.242480e-05     1.641786e-26
##         loudness      speechiness     acousticness instrumentalness
##     5.626618e-03     2.280619e-47     1.093119e-40     3.183210e-34
##         liveness          valence            tempo
##     2.914501e-35     3.140314e-17     1.282909e-11
```

```r
#Techno
techno.data <- train[train$track_genre == "techno",
                     subset.features]
shapiro.results <- lapply(techno.data, shapiro.test)
p.values <- sapply(shapiro.results, function(x) x$p.value)
print(p.values)
```

```
##       popularity      duration_ms     danceability           energy
##     2.656656e-18     2.526425e-13     3.785981e-10     4.811881e-16
##         loudness      speechiness     acousticness instrumentalness
##     1.391520e-04     2.670890e-42     2.989879e-40     6.453145e-31
##         liveness          valence            tempo
##     8.708819e-37     2.685097e-20     5.346623e-25
```

```r
#Hiphop
hiphop.data <- train[train$track_genre == "hip-hop",
                     subset.features]
shapiro.results <- lapply(hiphop.data, shapiro.test)
p.values <- sapply(shapiro.results, function(x) x$p.value)
print(p.values)
```

```
##       popularity      duration_ms     danceability           energy
##     3.841300e-32     6.402548e-19     4.760587e-15     2.639350e-07
##         loudness      speechiness     acousticness instrumentalness
##     2.400107e-10     1.860696e-27     4.817391e-27     3.465389e-51
##         liveness          valence            tempo
##     7.076680e-32     5.774669e-09     1.133908e-18
```

```r
#Rock
rock.data <- train[train$track_genre == "rock-n-roll",
                   subset.features]
shapiro.results <- lapply(rock.data, shapiro.test)
p.values <- sapply(shapiro.results, function(x) x$p.value)
print(p.values)
```

```
##       popularity      duration_ms     danceability           energy
##     1.887857e-23     9.724026e-25     1.602254e-05     2.662127e-08
##         loudness      speechiness     acousticness instrumentalness
##     1.512670e-05     1.169475e-40     5.895010e-18     1.684096e-49
##         liveness          valence            tempo
##     2.690681e-28     3.965475e-17     2.030462e-12
```

```r
#Reggaeton
reggaeton.data <- train[train$track_genre == "reggaeton",
                        subset.features]
shapiro.results <- lapply(reggaeton.data, shapiro.test)
p.values <- sapply(shapiro.results, function(x) x$p.value)
print(p.values)
```

```
##       popularity      duration_ms     danceability           energy
##     4.275042e-34     8.478803e-16     8.519430e-07     8.356745e-09
##         loudness      speechiness     acousticness instrumentalness
##     3.718124e-18     1.562947e-29     1.483868e-26     2.473546e-51
##         liveness          valence            tempo
##     1.645680e-31     8.474368e-12     2.329162e-29
```

The Shapiro-Wilks test measures the discrepancy between the observed data and the expected values under the assumption of normality. If the p-value associated with the test statistic is greater than a predetermined significance level (often 0.05), we fail to reject the null hypothesis and conclude that the data appears to be normally distributed. On the other hand, if the p-value is less than the significance level, we should reject the null hypothesis and conclude that the data does not follow a normal distribution.

In our case we definitely don't have data that behave like normal distributions but we try to perform the LDA no matter the results. We already know they will not probably be good.

LDA

We now fit a LDA model.

```r
lda.fit <- lda(track_genre ~ popularity + duration_ms +
                 danceability + energy + loudness +
                 speechiness + acousticness +
                 instrumentalness + liveness + valence +
                 tempo, data = train)
lda.fit
```

```
## Call:
## lda(track_genre ~ popularity + duration_ms + danceability + energy +
##     loudness + speechiness + acousticness + instrumentalness +
##     liveness + valence + tempo, data = train)
##
## Prior probabilities of groups:
##   classical     hip-hop   reggaeton rock-n-roll      techno
##     0.19675     0.20275     0.20075     0.19925     0.20050
##
## Group means:
##             popularity duration_ms danceability    energy   loudness
## classical     13.22236    233794.4    0.3839808 0.1911691 -20.077089
## hip-hop       37.23058    208037.9    0.7340148 0.6813687  -5.932275
## reggaeton     23.97634    211395.3    0.7603587 0.7380648  -4.982870
## rock-n-roll   34.64366    174021.5    0.5510703 0.5246854  -9.463541
## techno        38.70075    313586.0    0.6838653 0.7468317  -8.095393
##             speechiness acousticness instrumentalness  liveness   valence
## classical    0.05138933   0.92221202      0.630402955 0.1679009 0.3824497
```

```
## hip-hop      0.12975647   0.19891423      0.011391446 0.1949020 0.5480551
## reggaeton    0.12281357   0.17215141      0.003087005 0.1711833 0.6458487
## rock-n-roll  0.05138545   0.54751905      0.014882970 0.2304536 0.6719084
## techno       0.06490062   0.08005016      0.542824006 0.1580966 0.3238574
##                 tempo
## classical   107.7002
## hip-hop     115.9614
## reggaeton   119.7955
## rock-n-roll 120.2644
## techno      128.2501
##
## Coefficients of linear discriminants:
##                             LD1           LD2           LD3           LD4
## popularity         5.557873e-03  9.387239e-04  1.711799e-02  2.917167e-02
## duration_ms       -5.793535e-07  1.992417e-06  7.788126e-08 -1.303064e-06
## danceability       4.495092e+00  4.461446e-01 -2.711732e+00  5.656827e-01
## energy             1.836355e+00  2.530745e+00  7.506567e-01 -1.826775e+00
## loudness           5.038409e-02 -5.396850e-02  4.691585e-02 -9.922123e-03
## speechiness        1.782676e+00 -1.586108e+00 -1.000236e+01  3.624186e+00
## acousticness      -2.842091e+00 -6.006009e-01 -9.661297e-02 -2.480977e-01
## instrumentalness -5.867679e-01  3.063733e+00  2.171179e-02 -6.171997e-01
## liveness         -2.689527e-01 -2.162292e-01  1.094160e+00  1.836220e+00
## valence          -1.449553e+00 -1.708086e+00  8.743224e-01 -2.133359e+00
## tempo              1.600026e-03  4.550574e-03  9.967578e-03 -4.679443e-03
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4
## 0.7474 0.2116 0.0321 0.0089
```

Here is the confusion matrix for the predictions.

```
lda.predictions <- predict(lda.fit, test)
#ldahist(data = lda.predictions$x[,1], g = train$track_genre)
cm.lda <- table(lda.predictions$class, test$track_genre)
cm.lda
```

```
##
##               classical hip-hop reggaeton rock-n-roll techno
##   classical         181       0         0           4      0
##   hip-hop             2     102        66          11     24
##   reggaeton           0      70       129           8     21
##   rock-n-roll        28      14         1         180      7
##   techno              2       3         1           0    146
```

We now proceed to compute the metrics of the performance of the LDA classifier.

```
## [1] "The accuracy of the LDA is:  0.738"
```

```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.98        0.85     0.91
## 2     hip-hop      0.50        0.54     0.52
## 3   reggaeton      0.57        0.65     0.61
## 4 rock-n-roll      0.78        0.89     0.83
## 5      techno      0.96        0.74     0.83
```

We will make our observations and talk about the results in the later comparison between models.

QDA

Let's use the *qda()* function from the `MASS` package.

```
qda.fit <- qda(track_genre ~ popularity + duration_ms +
                 danceability + energy + loudness +
                 speechiness + acousticness +
                 instrumentalness + liveness +
                 valence + tempo, data = train)
qda.fit
```

```
## Call:
## qda(track_genre ~ popularity + duration_ms + danceability + energy +
##     loudness + speechiness + acousticness + instrumentalness +
##     liveness + valence + tempo, data = train)
##
## Prior probabilities of groups:
##   classical     hip-hop   reggaeton rock-n-roll      techno
##     0.19675     0.20275     0.20075     0.19925     0.20050
##
## Group means:
##             popularity duration_ms danceability    energy   loudness
## classical     13.22236    233794.4    0.3839808 0.1911691 -20.077089
## hip-hop       37.23058    208037.9    0.7340148 0.6813687  -5.932275
## reggaeton     23.97634    211395.3    0.7603587 0.7380648  -4.982870
## rock-n-roll   34.64366    174021.5    0.5510703 0.5246854  -9.463541
## techno        38.70075    313586.0    0.6838653 0.7468317  -8.095393
##             speechiness acousticness instrumentalness  liveness   valence
## classical    0.05138933   0.92221202      0.630402955 0.1679009 0.3824497
## hip-hop      0.12975647   0.19891423      0.011391446 0.1949020 0.5480551
## reggaeton    0.12281357   0.17215141      0.003087005 0.1711833 0.6458487
## rock-n-roll  0.05138545   0.54751905      0.014882970 0.2304536 0.6719084
## techno       0.06490062   0.08005016      0.542824006 0.1580966 0.3238574
##                tempo
## classical   107.7002
## hip-hop     115.9614
## reggaeton   119.7955
## rock-n-roll 120.2644
## techno      128.2501
```

Here we show the confusion matrix.

```
qda.predictions <- predict(qda.fit, test)
cm.qda <- table(qda.predictions$class, test$track_genre)
cm.qda
```

```
##
##               classical hip-hop reggaeton rock-n-roll techno
##   classical         192       0         1           7      1
##   hip-hop             5      68        20           5      7
```

```
##   reggaeton           0     111       175           9     31
##   rock-n-roll        14       5         1         181      5
##   techno              2       5         0           1    154
```

And here are the metrics, as before, we will talk later about them.

```
#Here are the performance
print(paste("The accuracy of the QDA is: ", round(cm.qda.accuracy, 2)))
```

```
## [1] "The accuracy of the QDA is:  0.77"
```

```
qda.metrics
```

```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.96        0.90     0.93
## 2     hip-hop      0.65        0.36     0.46
## 3   reggaeton      0.54        0.89     0.67
## 4 rock-n-roll      0.88        0.89     0.89
## 5      techno      0.95        0.78     0.86
```

Marginal considerations As we could see from the Shapiro-Wilkis test the normality assumption of the methods was not satisfied. This suggests that no matter how the result may be good compared to the other methods we tried, other techniques should be preferred in the scenario we are considering.

Naive Bayes

The Naive Bayes classifier is another techniques that approximates a Bayes classifier. As we discussed earlier LDA and QDA assume that the predictors are distributed normally within a class. The Naive Bayes relaxes this assumption by making no hypothesis on the family on the distribution. At the same time, being the computation of a joint probability intractable in most cases, it assumes that the simplifying assumption of independence of the the predictors within a class. Even if this simplification is not realistic in most cases, the Naive Bayes is known to work well in practice.

Let's now see how it performs on our data.

```
#Naive bayes classifier
nb.fit <- naiveBayes(track_genre ~ ., data = train[,1:16])
nb.class <- predict(nb.fit, test[,1:16])
cm.nb <- table(nb.class, test$track_genre)
```

Let's take a look at the confusion matrix for our predictions.

```
##
## nb.class      classical hip-hop reggaeton rock-n-roll techno
##   classical         188       0         1           7      2
##   hip-hop             1      64        30           4     13
##   reggaeton           0     114       165          11     32
##   rock-n-roll        19       6         1         180      8
##   techno              5       5         0           1    143
```

And to the performance.

```
## [1] "The accuracy of the Naive Bayes is:  0.74"
```

```
##         Genre Precision Recall(TPR) F1-score
## 1   classical      0.95        0.88     0.91
## 2     hip-hop      0.57        0.34     0.43
## 3   reggaeton      0.51        0.84     0.64
## 4 rock-n-roll      0.84        0.89     0.86
## 5      techno      0.93        0.72     0.81
```

Models Comparison

We want to make a comparison between the models we built so we summarize in the following tables the results obtained for our genre prediction task.

First we take a look at the accuracy achieved by each model.

Model	**Accuracy**
Logistic Regression models with variable selection	0.76
Logistic Regression models with ridge regularization	0.74
Logistic Regression models with lasso regularization	0.75
LDA	0.74
QDA	0.77
Naive Bayes	0.74

And then to the performance with respect to each class:

- *Classical*

Model	**Precision**	**Recall(TPR)**	**F1-score**
Logistic Regression with variable selection	0.97	0.82	0.89
Logistic Regression models with ridge regularization	0.99	0.77	0.87
Logistic Regression models with lasso regularization	0.99	0.83	0.90
LDA	0.98	0.85	0.91
QDA	0.96	0.90	0.93
Naive Bayes	0.95	0.88	0.91

- *Hip-hop*

Model	**Precision**	**Recall(TPR)**	**F1-score**
Logistic Regression with variable selection	0.55	0.62	0.58
Logistic Regression models with ridge regularization	0.55	0.62	0.58
Logistic Regression models with lasso regularization	0.55	0.53	0.54
LDA	0.50	0.54	0.52
QDA	0.65	0.36	0.46
Naive Bayes	0.57	0.34	0.43

- *Reggaeton*

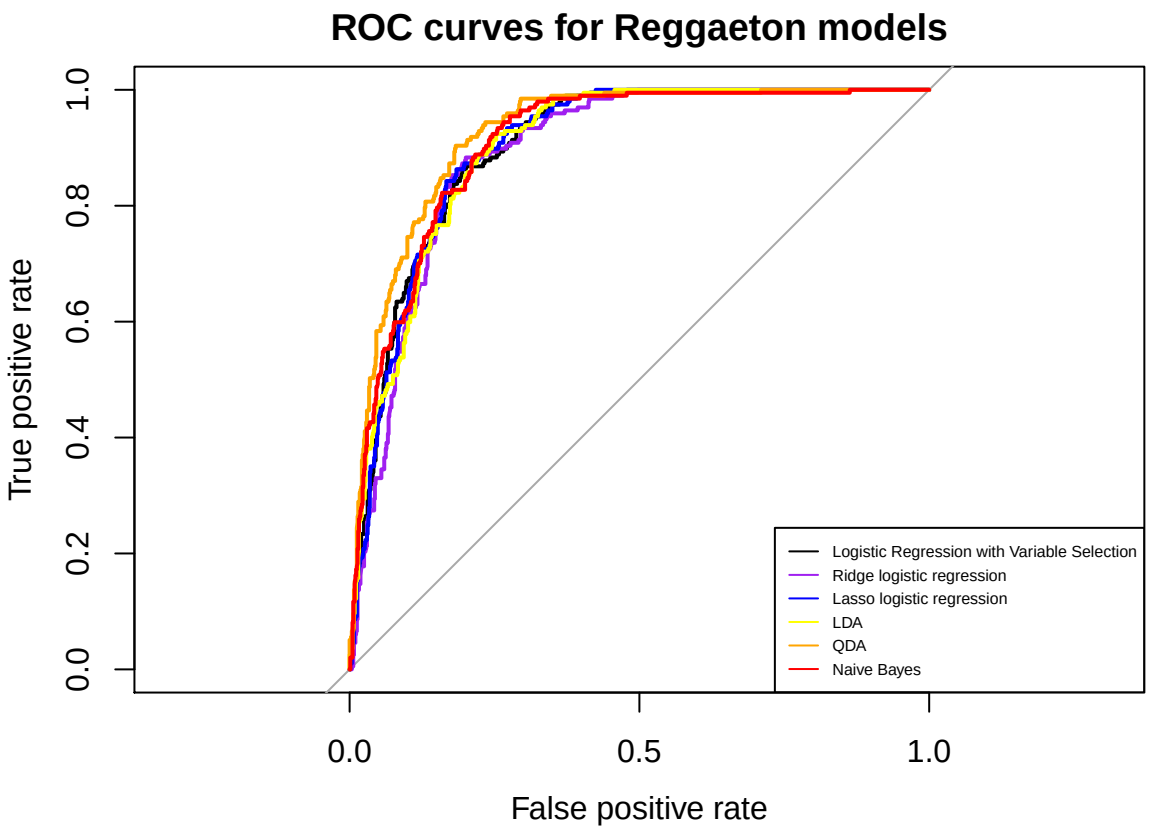
Model	Precision	Recall(TPR)	F1-score
Logistic Regression with variable selection	0.64	0.62	0.63
Logistic Regression models with ridge regularization	0.62	0.61	0.62
Logistic Regression models with lasso regularization	0.58	0.70	0.63
LDA	0.57	0.65	0.61
QDA	0.54	0.89	0.67
Naive Bayes	0.51	0.84	0.64

- *Rock and Roll*

Model	Precision	Recall(TPR)	F1-score
Logistic Regression with variable selection	0.78	0.93	0.85
Logistic Regression models with ridge regularization	0.74	0.93	0.82
Logistic Regression models with lasso regularization	0.80	0.93	0.86
LDA	0.78	0.89	0.83
QDA	0.88	0.89	0.89
Naive Bayes	0.84	0.89	0.86

- *Techno*

Model	Precision	Recall(TPR)	F1-score
Logistic Regression with variable selection	0.90	0.79	0.84
Logistic Regression models with ridge regularization	0.89	0.77	0.82
Logistic Regression models with lasso regularization	0.91	0.77	0.83
LDA	0.96	0.74	0.83
QDA	0.95	0.78	0.86
Naive Bayes	0.93	0.72	0.81




```
## [1] "Auc for Reggaeton classification models"
```

```
## [1] "Logistic regression with model selection: AUC = 0.902"

## [1] "Ridge logistic regression:                AUC = 0.891"

## [1] "Lasso logistic regression:                AUC = 0.902"

## [1] "LDA model:                                AUC = 0.901"

## [1] "QDA model:                                AUC = 0.925"

## [1] "Naive Bayes model:                        AUC = 0.907"
```

Note how the results for the *overall accuracy*, the *F1-score* and the *AUC* of the models brings to different leader board:

- The *accuracy* table shows us a general result of how each model perform in the classification of each genre contemporaneously. None of the model reach outstanding results, they all wander around 75% of accuracy, with QDA that reaches the top of 77%.
- When we see at the table with *precision, recall, and F1-score* we must take one genre at a time. In this case, different `track_genre` leads to different best models. As general observations we see that the ridge

model is the one that perform worse, with an exception in `hip-hop` genre. QDA still seems the one with better performances, again besides `hip-hop`.

• We then wanted to see how the models behave by comparing the *Roc Curves* and AUC. As an example we focused on one single genre, `reggaeton`. We plotted the ROC curves of all models and we can see that all curves are similar. In particular the predictor build with QDA is the one that performed better, as also noticed earlier, with the AUC of 0.925. The worse AUC instead is given by the *Logistic regression with Ridge regularization*, with an AUC of 0.891.

Conclusions

In this report we explored the task of `track_genre` classification, with the final objective of building a model that could accurately predict the genre of a give track based on its audio features.

Through our analysis, we studied features correlations and we experimented with various machine learning algorithms. We found out that `track_genre` is partially correlated with some audio features of the music track. It might be that a feature is correlated with each of the genres or that it is highly correlated with only one or two genres. This is the case for example of `instrumentalness` and his strict correlation with `classical` and `techno` and plays a significant role in the classification.

The model that gained the best accuracy score in this task is the QDA model with a 77%, even if the scores are all pretty close to each others. When we see instead the single genre classification QDA is still the best model for most of the genres except for `hip-hop` where the logistic regressions with variable selection and ridge regularization got the best scores.

Our results are not particularly relevant seen the poor accuracy our best model could reach. In general however, it is important to note that `track_genre` classification is a complex task due to the inherent subjectivity and ambiguity in defining music genres. Some genres may have some particular features or relevant characteristic that better separate them from others, but in general this is not the case for all genres. Moreover, the dataset itself might have certain limitations or biases, impacting the performance and generalization of our models. To further improve the accuracy of genre classification, future research could consider incorporating additional data sources, such as lyrics.