ML Predictive Model On Wheter Like/Dislike Music From A Spotify Playlist

Marcelo Argotti

6/20/2020

Contents

1 INTRODUCTION

In this report, an attempt to build machine learning algorithms that can predict the musical taste (like/dislike) of a user based on the tracks present in his Spotify playlist will be made. The following analysis also explores the audio features of the songs and extracted from the Spotify Web API. These available features include attributes such as: acousticness, danceability, duration_ms, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time signature and valence. For more details see **Spotify Audio Features**.

This project is relevant to music lovers. Not only will it help to determine how accurate is the musical preference of a listener, it could also help understand which audio features are likely to become more pleasant than others. By the end of the project, an ensemble will be built from the best models which can predict a target (like or dislike). The most successful algorithms were LDA and Logistic Regression.

2 DATA EXPLORATION

Data Source: The data for this study was acquired from Kaggle's repository, maintained by GeorgeMcIntire. This dataset collects 2017 songs with audio attributes from the Spotify's API as well as complementary information such as artist name, song title and target. Each song is labeled "1" meaning 'like it' and "0" meaning 'don't like'. For more information click **here**

The dataset was splited into a training set and a validation set. The training set was used to develop the algorithm, while the validation set was used to evaluate the predictions. Confusion Matrix was used to evaluate the results.

2.1 Packages and libraries

```r
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(ggcorrplot)) install.packages("ggcorrplot",
                                          repos = "http://cran.us.r-project.org")
if(!require(rattle)) install.packages("rattle",
                                      repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart",
                                     repos = "http://cran.us.r-project.org")
if(!require(rpart.plot)) install.packages("rpart.plot",
                                          repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr",
                                     repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",
                                     repos = "http://cran.us.r-project.org")
if(!require(class)) install.packages("class",
                                     repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
                                            repos = "http://cran.us.r-project.org")
if(!require(MASS)) install.packages("MASS",
                                    repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos="http://cran.us.r-project.org")
```

2.2 Preparing data

For practical issues I made a copy of the original dataset into my GitHub repo. The data set and the attached info file will both be downloaded.

```
# SpotifyClassification 2017 dataset:
  # https://github.com/margottig/spotyATR/archive/master.zip

temp <- tempfile()
download.file("https://github.com/margottig/spotyATR/archive/master.zip", temp)

#read downloaded data
data <- read.csv(unz(temp, "spotyATR-master/data.csv"))
unlink(temp)
```

3 EXPLORATORY ANALYSIS

In order to get insights into the dataset, some exploratory analysis was made.

```
str(data)
```

```
## 'data.frame':    2017 obs. of  17 variables:
##  $ X               : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ acousticness    : num  0.0102 0.199 0.0344 0.604 0.18 0.00479 0.0145 0.0202 0.0481 0.00208 ...
##  $ danceability    : num  0.833 0.743 0.838 0.494 0.678 0.804 0.739 0.266 0.603 0.836 ...
##  $ duration_ms     : int  204600 326933 185707 199413 392893 251333 241400 349667 202853 226840 ...
##  $ energy          : num  0.434 0.359 0.412 0.338 0.561 0.56 0.472 0.348 0.944 0.603 ...
##  $ instrumentalness: num  2.19e-02 6.11e-03 2.34e-04 5.10e-01 5.12e-01 0.00 7.27e-06 6.64e-01 0.00 0
##  $ key             : int  2 1 2 5 5 8 1 10 11 7 ...
##  $ liveness        : num  0.165 0.137 0.159 0.0922 0.439 0.164 0.207 0.16 0.342 0.571 ...
##  $ loudness        : num  -8.79 -10.4 -7.15 -15.24 -11.65 ...
##  $ mode            : int  1 1 1 1 0 1 1 0 0 1 ...
##  $ speechiness     : num  0.431 0.0794 0.289 0.0261 0.0694 0.185 0.156 0.0371 0.347 0.237 ...
##  $ tempo           : num  150.1 160.1 75 86.5 174 ...
##  $ time_signature  : num  4 4 4 4 4 4 4 4 4 4 ...
##  $ valence         : num  0.286 0.588 0.173 0.23 0.904 0.264 0.308 0.393 0.398 0.386 ...
##  $ target          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ song_title      : chr  "Mask Off" "Redbone" "Xanny Family" "Master Of None" ...
##  $ artist          : chr  "Future" "Childish Gambino" "Future" "Beach House" ...
```

The dataset has 17 variables and 2017 observations. Although some are categorical in nature, all variables are stored as numeric/integers except the *song_title* and *artist* variables which are stored as characters. As we can observe from the information displayed above, there are multiple variables that are presented in different magnitudes or scales. Further on, in order to get better consistency in the data analysis we will scale these values.

```
#Some summary statistics
summary(data)
```

```
##        X          acousticness        danceability     duration_ms
##  Min.   :   0   Min.   :0.0000028   Min.   :0.1220   Min.   :  16042
```

```
##  1st Qu.: 504   1st Qu.:0.0096300   1st Qu.:0.5140   1st Qu.: 200015
##  Median :1008   Median :0.0633000   Median :0.6310   Median : 229261
##  Mean   :1008   Mean   :0.1875900   Mean   :0.6184   Mean   : 246306
##  3rd Qu.:1512   3rd Qu.:0.2650000   3rd Qu.:0.7380   3rd Qu.: 270333
##  Max.   :2016   Max.   :0.9950000   Max.   :0.9840   Max.   :1004627
##      energy       instrumentalness        key            liveness
##  Min.   :0.0148   Min.   :0.0000000   Min.   : 0.000   Min.   :0.0188
##  1st Qu.:0.5630   1st Qu.:0.0000000   1st Qu.: 2.000   1st Qu.:0.0923
##  Median :0.7150   Median :0.0000762   Median : 6.000   Median :0.1270
##  Mean   :0.6816   Mean   :0.1332855   Mean   : 5.343   Mean   :0.1908
##  3rd Qu.:0.8460   3rd Qu.:0.0540000   3rd Qu.: 9.000   3rd Qu.:0.2470
##  Max.   :0.9980   Max.   :0.9760000   Max.   :11.000   Max.   :0.9690
##     loudness            mode         speechiness          tempo
##  Min.   :-33.097   Min.   :0.0000   Min.   :0.02310   Min.   : 47.86
##  1st Qu.: -8.394   1st Qu.:0.0000   1st Qu.:0.03750   1st Qu.:100.19
##  Median : -6.248   Median :1.0000   Median :0.05490   Median :121.43
##  Mean   : -7.086   Mean   :0.6123   Mean   :0.09266   Mean   :121.60
##  3rd Qu.: -4.746   3rd Qu.:1.0000   3rd Qu.:0.10800   3rd Qu.:137.85
##  Max.   : -0.307   Max.   :1.0000   Max.   :0.81600   Max.   :219.33
##  time_signature     valence           target        song_title
##  Min.   :1.000   Min.   :0.0348   Min.   :0.0000   Length:2017
##  1st Qu.:4.000   1st Qu.:0.2950   1st Qu.:0.0000   Class :character
##  Median :4.000   Median :0.4920   Median :1.0000   Mode  :character
##  Mean   :3.968   Mean   :0.4968   Mean   :0.5057
##  3rd Qu.:4.000   3rd Qu.:0.6910   3rd Qu.:1.0000
##  Max.   :5.000   Max.   :0.9920   Max.   :1.0000
##     artist
##  Length:2017
##  Class :character
##  Mode  :character
##
##
##
```

```r
#check missing values
colSums(is.na(data))
```

```
##                X     acousticness     danceability      duration_ms
##                0                0                0                0
##           energy instrumentalness              key         liveness
##                0                0                0                0
##         loudness             mode      speechiness            tempo
##                0                0                0                0
##   time_signature          valence           target       song_title
##                0                0                0                0
##           artist
##                0
```

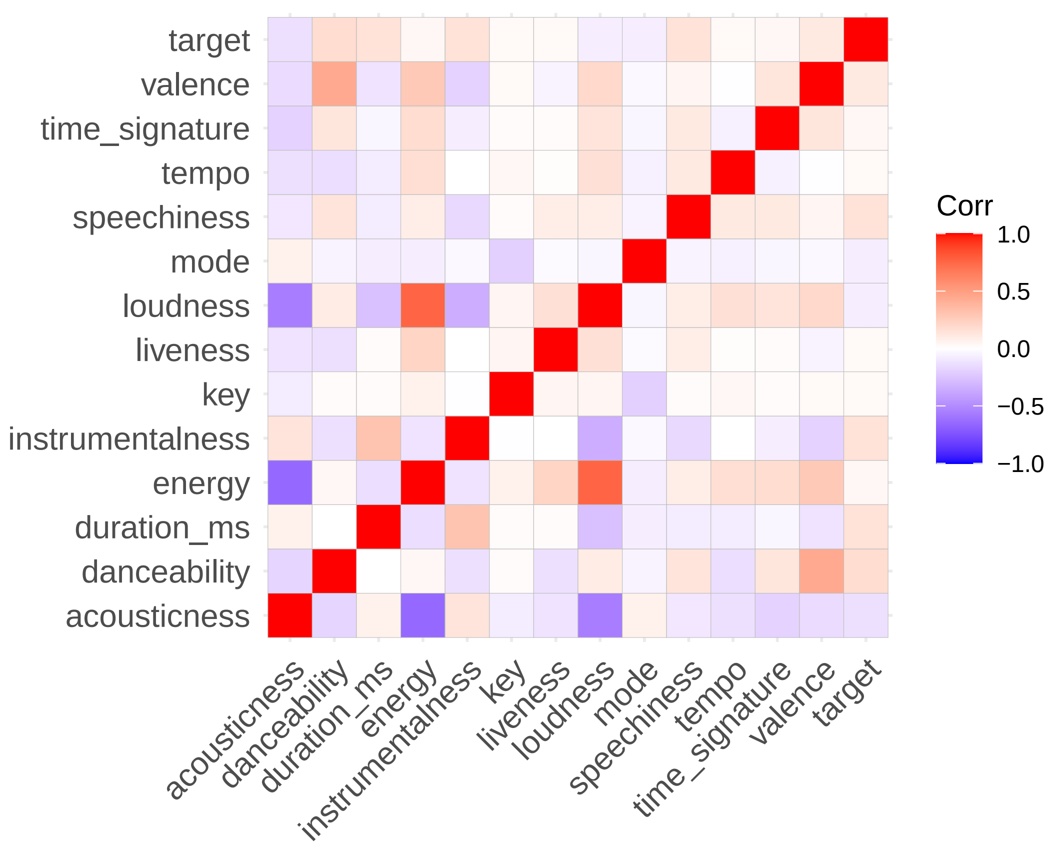
No NAs were found

3.1 Explore target variable

```
#Let see if there is repeated data. Count unique values in the X variable (song_id)
length(unique(data$X))
```

```
## [1] 2017
```

```
#Let explore how audio features correlate between them or else see which of the features
#correlate the best with the target variable (liked/disliked song)
corr <- round(cor(data[,2:15]),6)
ggcorrplot(corr)
```



It can be observed that the variables **loudness-energy** and **valence-danceability** are correlated to some extent compared to the other variables.

3.2 Data Wrangling

```
#Transform categorical variables into factors in order to represent data more
# efficiently. Before converting this variables, lets make a copy of the data.
playlist <- data
playlist$target <- factor(playlist$target, levels = c(0,1), labels = c("dislike", "like"))

data$target <- factor(data$target, levels = c(0,1), labels = c("dislike", "like"))
```

```r
data$mode <- factor(data$mode, levels=c(0,1), labels=c("minor", "major"))
data$key <- factor(playlist$key)
data$duration_ms <- playlist$duration_ms/60000
#It is know from pitch class integer notation that each number from the key variable
# are relative to a specific letter in elementary music theory. So now we convert
#the numerical keys to the actual musical keys

levels(data$key)[1] <- "C"
levels(data$key)[2] <- "C#"
levels(data$key)[3] <- "D"
levels(data$key)[4] <- "D#"
levels(data$key)[5] <- "E"
levels(data$key)[6] <- "F"
levels(data$key)[7] <- "F#"
levels(data$key)[8] <- "G"
levels(data$key)[9] <- "G#"
levels(data$key)[10] <- "A"
levels(data$key)[11] <- "A#"
levels(data$key)[12] <- "B"

#convert the duration_ms units to minutes
playlist$duration_ms <- playlist$duration_ms/60000

# identify which class applies for each variable
sapply(playlist, class)
```

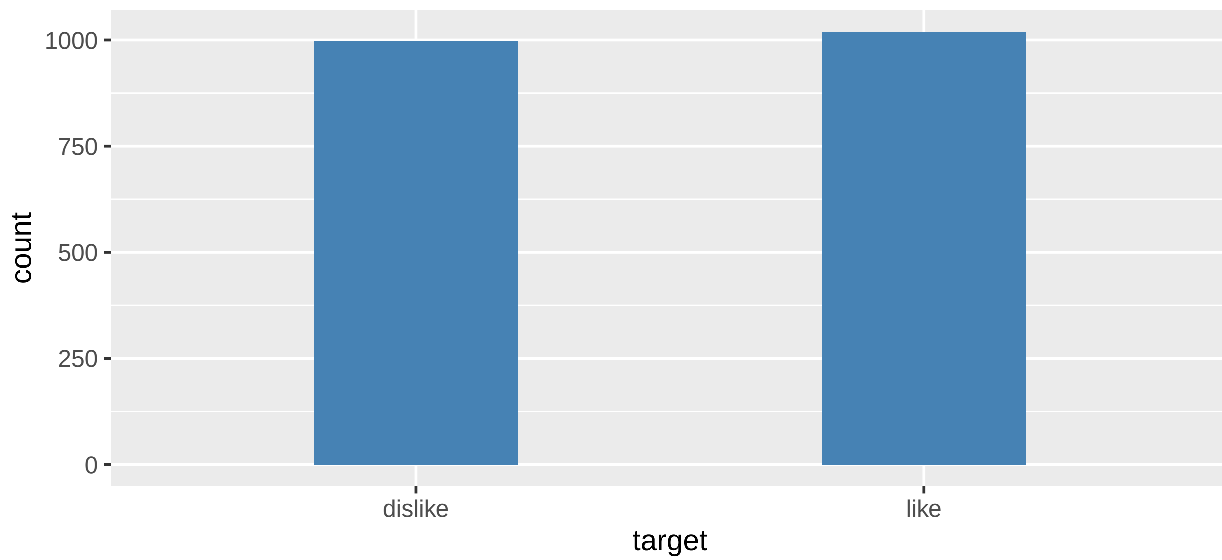
```
##                X     acousticness     danceability      duration_ms
##        "integer"        "numeric"        "numeric"        "numeric"
##           energy instrumentalness              key         liveness
##        "numeric"        "numeric"        "integer"        "numeric"
##         loudness             mode      speechiness            tempo
##        "numeric"        "integer"        "numeric"        "numeric"
##   time_signature          valence           target       song_title
##        "numeric"        "numeric"         "factor"      "character"
##           artist
##      "character"
```

```r
# At the moment we will not be using character variables, so lets skipped them
#from our dataset including the song_id variable 'X'

playlist$X <- NULL
playlist$song_title <- NULL
playlist$artist <- NULL
```

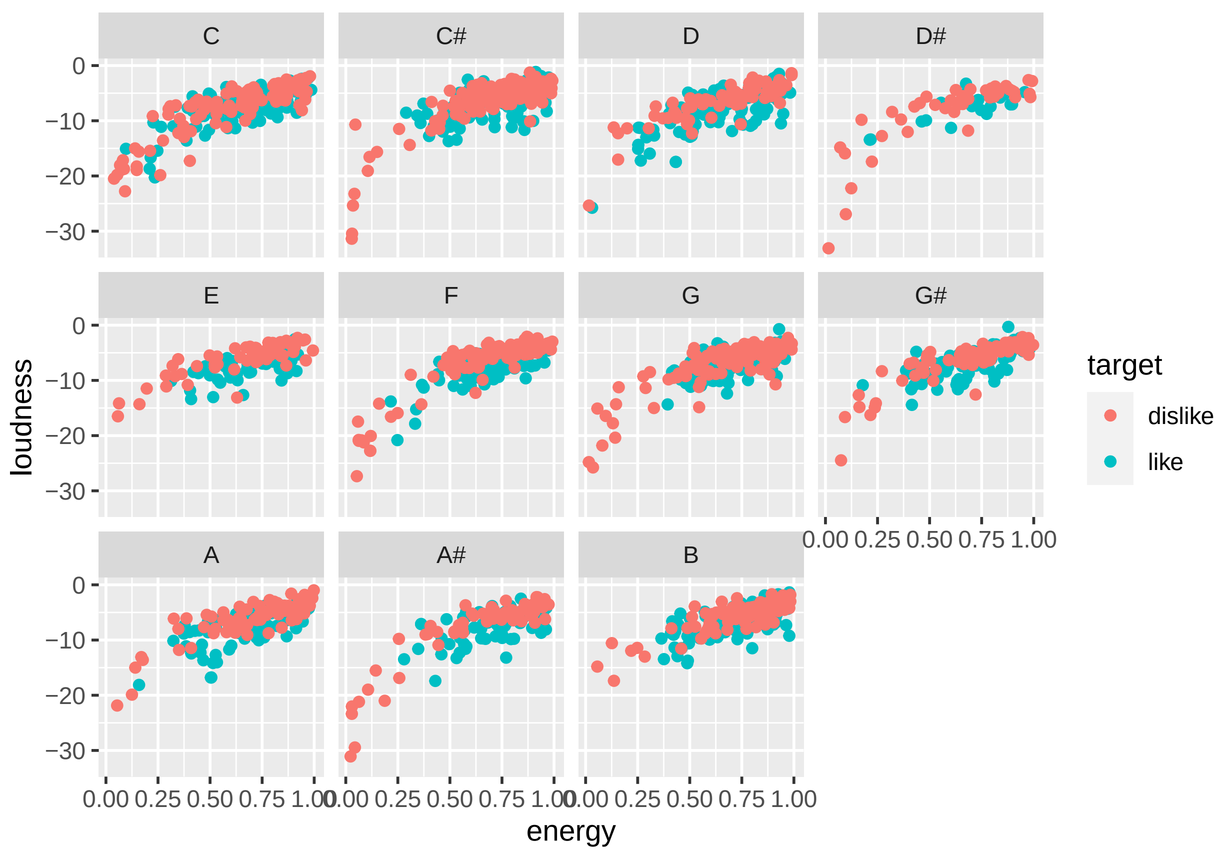
4 VISUALIZATION

The distribution of **like** and **dislike** target values is as follows:

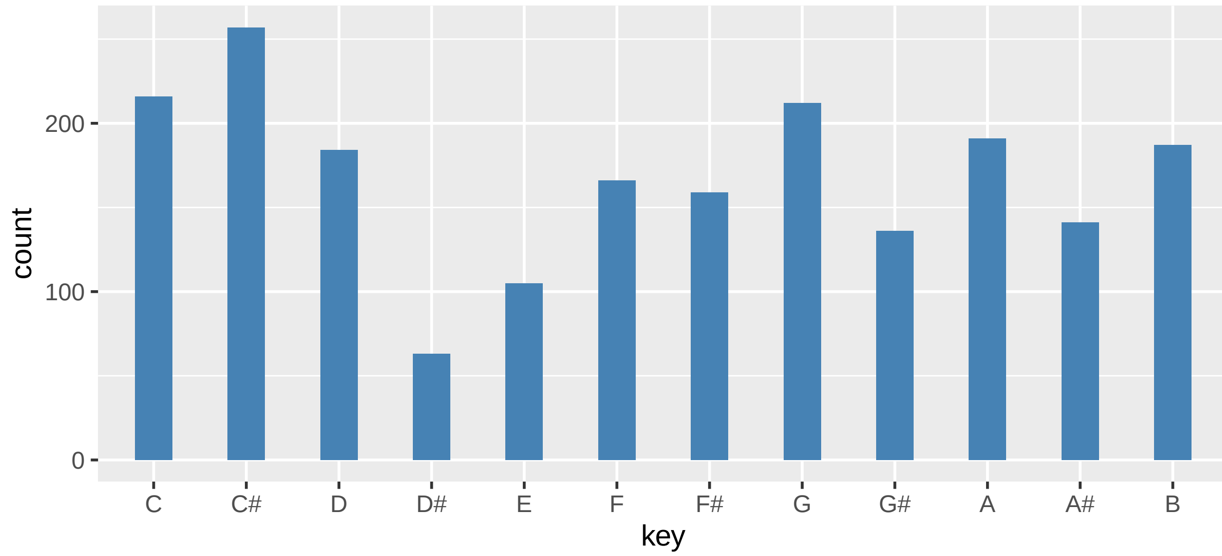
```r
playlist %>% ggplot(aes(target)) + geom_bar(width = 0.4,fill="steelblue")
```




```r
# Let's see how likely is the musical taste among key notation
KeyNotation <- c("C","C#","D", "D#", "E", "F", "G", "G#", "A","A#", "B")
Taste <- c("like", "dislike")
data %>% filter(key%in%KeyNotation & target%in%Taste) %>%
  ggplot(aes(energy, loudness, col = target)) +
  geom_point() + facet_wrap(~key)
```

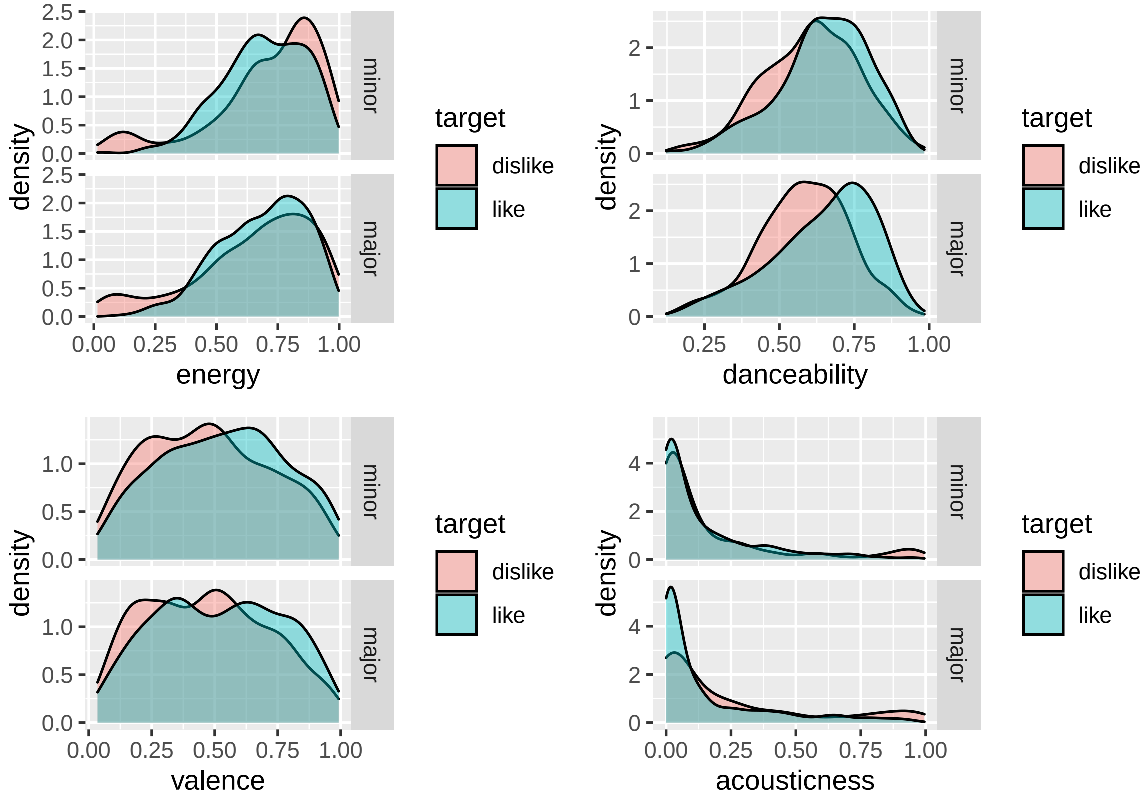


```r
#Let see which key notes are most predominant in data
ggplot(data) + geom_bar(aes(key),width = 0.4,fill="steelblue")
```

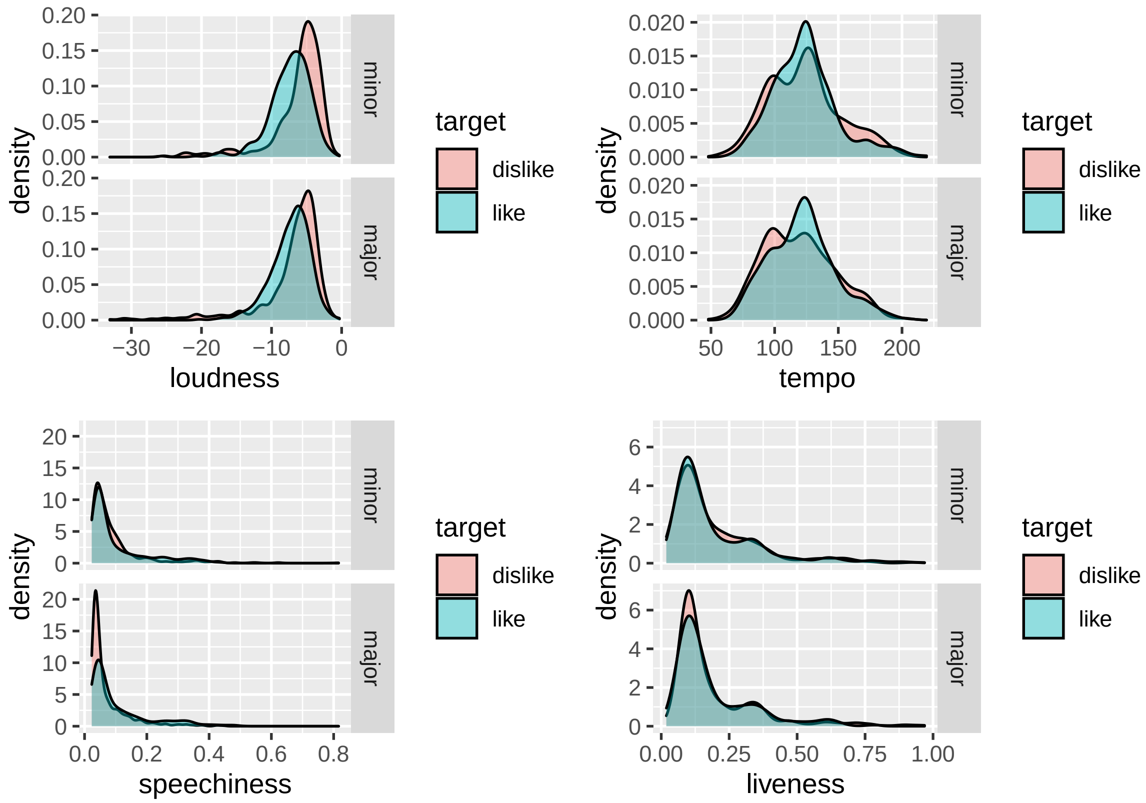


From the previous scatterplot it is difficult to reveal important characteristics of the data distribution.It will be useful to visualize some variables through density plots.

```r
# energy density
plot_energy <- data %>% ggplot(aes(energy, fill = target)) +
  geom_density(alpha=0.4) + facet_grid(mode~.)
# loudness density
plot_loudness <- data %>% ggplot(aes(loudness, fill = target)) +
  geom_density(alpha=0.4) + facet_grid(mode~.)
# danceability density
plot_danceability <- data %>% ggplot(aes(danceability, fill = target)) +
  geom_density(alpha=0.4) + facet_grid(mode~.)
# acousticness density
plot_acousticness <- data %>% ggplot(aes(acousticness, fill = target)) +
  geom_density(alpha=0.4) +  facet_grid(mode~.)
#valence density
plot_valence <- data %>% ggplot(aes(valence, fill = target)) +
  geom_density(alpha=0.4) +  facet_grid(mode~.)
# speechiness density
plot_speechiness <- data %>% ggplot(aes(speechiness, fill = target)) +
  geom_density(alpha=0.4) +  facet_grid(mode~.)
#liveness density
plot_liveness <- data %>% ggplot(aes(liveness, fill = target)) +
  geom_density(alpha=0.4) +  facet_grid(mode~.)
# tempo density
plot_tempo <- data %>% ggplot(aes(tempo, fill = target)) +
  geom_density(alpha=0.4) +  facet_grid(mode~.)
#combine plots
grid.arrange(plot_energy, plot_danceability, plot_valence, plot_acousticness)
```




```r
grid.arrange(plot_loudness, plot_tempo, plot_speechiness, plot_liveness)
```



5 MODELING APPROACH

In this section, we will be applying different Machine Learning algorithms on the training set and then validate against the test set. We'll work with seven different methods to compare predictions:

- Decision Tree Model
- Generalized Linear Model GLM
- K-nearest neighbors KNN
- Support Vector Machine SVM
- Naive Bayes Classifier
- Random Forest Classifier
- Linear Discriminant Analysis LDA

5.1 Creating partitions of training and test dataset

The data set was divided into two parts: train (80%) and test (20%) dataset.

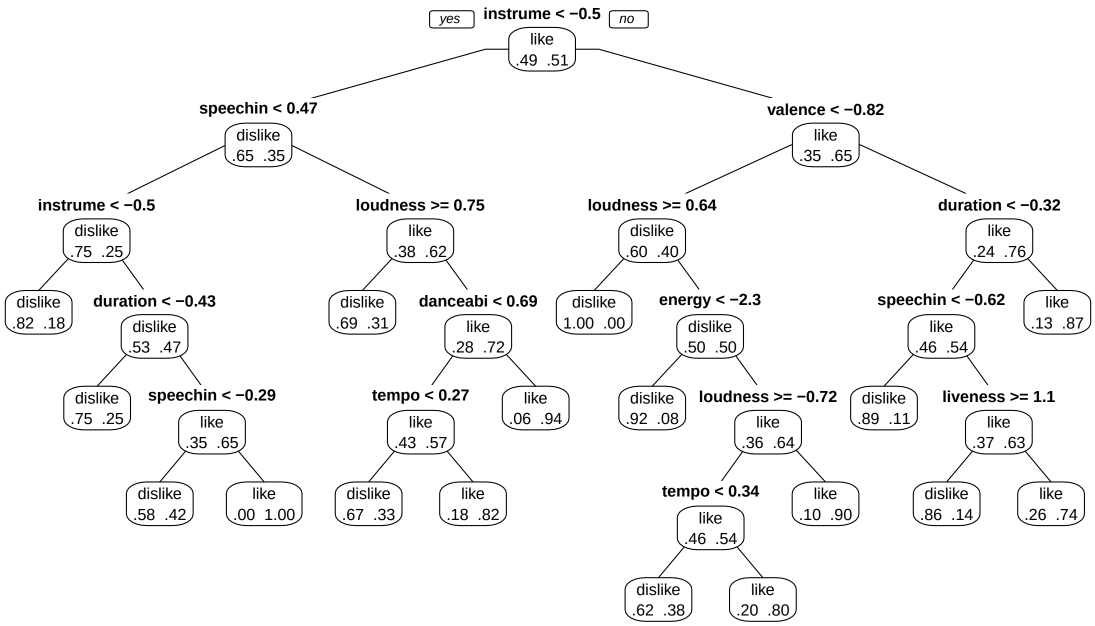
```
#Create Data partition into test set and training set 80% and 20%

set.seed(1)
test_index <- createDataPartition(y = playlist$target, times = 1, p = 0.80,
                                  list = FALSE)
Train <- playlist[-test_index,]
Test <- playlist[test_index,]

#scale data
ScaledTrain <- Train
ScaledTest <- Test

ScaledTrain[, -14]=scale(ScaledTrain[, -14])
ScaledTest[, -14]=scale(ScaledTest[, -14])
```

5.2 Decision Tree Model

Applying the Decision Tree model:

```
DecisionT <- rpart(target~., data=ScaledTrain)
prp(DecisionT, type=1,extra=4, main= "Probabilities per class")
```

Probabilities per class




```r
dt_pred <- predict(DecisionT, ScaledTest, type= "class")
cm_decisiontree = confusionMatrix(dt_pred, ScaledTest$target, positive= "like")
cm_decisiontree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     271  185
##    like        527  631
##
##                Accuracy : 0.5589
##                  95% CI : (0.5342, 0.5833)
##     No Information Rate : 0.5056
##     P-Value [Acc > NIR] : 1.013e-05
##
##                   Kappa : 0.1134
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7733
##             Specificity : 0.3396
##          Pos Pred Value : 0.5449
##          Neg Pred Value : 0.5943
##              Prevalence : 0.5056
```

```
##          Detection Rate : 0.3910
##    Detection Prevalence : 0.7175
##       Balanced Accuracy : 0.5564
##
##        'Positive' Class : like
##
```

5.3 Generalized Linear Model GLM

Applying the Logistic Regression model:

```r
LogicRegression = glm(formula = target ~ ., family = binomial,data = ScaledTrain)
# Predicting the Test set results
lr_pred = predict(LogicRegression, type = 'response', newdata = ScaledTest)
y_pred = ifelse(lr_pred > 0.5, "like", "dislike") %>% factor
# Making the Confusion Matrix
cm_glm = confusionMatrix(y_pred, ScaledTest$target, positive = "like")
cm_glm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     528  298
##    like        270  518
##
##                Accuracy : 0.6481
##                  95% CI : (0.6242, 0.6714)
##     No Information Rate : 0.5056
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.2963
##
##  Mcnemar's Test P-Value : 0.2573
##
##             Sensitivity : 0.6348
##             Specificity : 0.6617
##          Pos Pred Value : 0.6574
##          Neg Pred Value : 0.6392
##              Prevalence : 0.5056
##          Detection Rate : 0.3209
##    Detection Prevalence : 0.4882
##       Balanced Accuracy : 0.6482
##
##        'Positive' Class : like
##
```

5.4 K-nearest neighbors KNN

Applying the kNN model with k = 5:

```r
# Fitting K-NN to the Training set and Predicting the Test set results
Knearest = knn(train = ScaledTrain[, -14],
             test = ScaledTest[, -14],
             cl = ScaledTrain[, 14],
             k = 5,
             prob = TRUE)

cm_knn = confusionMatrix(Knearest, ScaledTest$target, positive = "like")
cm_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     645  367
##    like        153  449
##
##                Accuracy : 0.6778
##                  95% CI : (0.6544, 0.7006)
##     No Information Rate : 0.5056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3575
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5502
##             Specificity : 0.8083
##          Pos Pred Value : 0.7458
##          Neg Pred Value : 0.6374
##              Prevalence : 0.5056
##          Detection Rate : 0.2782
##    Detection Prevalence : 0.3730
##       Balanced Accuracy : 0.6793
##
##        'Positive' Class : like
##
```

5.5 Support Vector Machine SVM

Applying SVM model:

```r
# Fitting SVM to the Training set
SVM = svm(formula = target ~ .,
                 data = ScaledTrain,
                 type = 'C-classification',
                 kernel = 'linear')

# Predicting the Test set results
svm_pred = predict(SVM, newdata = ScaledTest[-14])

# Making the Confusion Matrix
```

```r
cm_svm = confusionMatrix(svm_pred, ScaledTest$target, positive = "like")
cm_svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     564  324
##    like        234  492
##
##                Accuracy : 0.6543
##                  95% CI : (0.6305, 0.6775)
##     No Information Rate : 0.5056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3093
##
##  Mcnemar's Test P-Value : 0.0001648
##
##             Sensitivity : 0.6029
##             Specificity : 0.7068
##          Pos Pred Value : 0.6777
##          Neg Pred Value : 0.6351
##              Prevalence : 0.5056
##          Detection Rate : 0.3048
##    Detection Prevalence : 0.4498
##       Balanced Accuracy : 0.6549
##
##        'Positive' Class : like
##
```

5.6 Naive Bayes Classifier

Applying the Naive Bayes model:

```r
NBay = naiveBayes(x = ScaledTrain[-14], y = ScaledTrain$target)
# Predicting the Test set results
nbay_pred = predict(NBay, newdata = ScaledTest[-14])
# Making the Confusion Matrix
cm_naivebayes = confusionMatrix(nbay_pred, ScaledTest$target, positive = "like")
cm_naivebayes
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     521  282
##    like        277  534
##
##                Accuracy : 0.6537
##                  95% CI : (0.6299, 0.6769)
##     No Information Rate : 0.5056
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.3073
##
##  Mcnemar's Test P-Value : 0.8657
##
##             Sensitivity : 0.6544
##             Specificity : 0.6529
##          Pos Pred Value : 0.6584
##          Neg Pred Value : 0.6488
##              Prevalence : 0.5056
##          Detection Rate : 0.3309
##    Detection Prevalence : 0.5025
##       Balanced Accuracy : 0.6536
##
##        'Positive' Class : like
##
```

5.7 Random Forest Classifier

Applying the Random Forest model, with number of trees limit set to 777:

```r
# Fitting Random Forest Classification to the Training set
RndForest = randomForest(x = ScaledTrain[-14],
                                       y = ScaledTrain$target,
                                       ntree = 777)

# Predicting the Test set results
rndforest_pred = predict(RndForest, newdata = ScaledTest[-14])

# Making the Confusion Matrix
cm_randomforest = confusionMatrix(rndforest_pred, ScaledTest$target, positive = "like")
cm_randomforest
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     391  137
##    like        407  679
##
##                Accuracy : 0.6629
##                  95% CI : (0.6393, 0.686)
##     No Information Rate : 0.5056
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3233
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8321
##             Specificity : 0.4900
##          Pos Pred Value : 0.6252
```

```
##          Neg Pred Value : 0.7405
##              Prevalence : 0.5056
##          Detection Rate : 0.4207
##    Detection Prevalence : 0.6729
##       Balanced Accuracy : 0.6610
##
##        'Positive' Class : like
##
```

5.8 Linear Discriminant Analysis LDA

Applying the Linear Discriminant Analysis (LDA) model:

```r
train_set = ScaledTrain
test_set = ScaledTest
# Applying LDA
LDA = lda(formula = target ~ ., data = train_set)
train_set = as.data.frame(predict(LDA, train_set))
train_set = train_set[c(4, 1)]
test_set = as.data.frame(predict(LDA, test_set))
test_set = test_set[c(4, 1)]

# Fitting SVM to the Training set
# install.packages('e1071')
classifier_lda = svm(formula = class ~ .,data = train_set,
                     type = 'C-classification', kernel = 'linear')

# Predicting the Test set results
lda_pred = predict(classifier_lda, newdata = test_set[-2])

# Making the Confusion Matrix
cm_lda = confusionMatrix(lda_pred, test_set$class, positive = "like")
cm_lda
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction dislike like
##    dislike     834   29
##    like          0  751
##
##                Accuracy : 0.982
##                  95% CI : (0.9743, 0.9879)
##     No Information Rate : 0.5167
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.964
##
##  Mcnemar's Test P-Value : 1.999e-07
##
##             Sensitivity : 0.9628
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
```

```
##          Neg Pred Value : 0.9664
##              Prevalence : 0.4833
##          Detection Rate : 0.4653
##    Detection Prevalence : 0.4653
##       Balanced Accuracy : 0.9814
##
##        'Positive' Class : like
##
```

6 RESULTS AND DISCUSSION

```r
# Comparison of confusion matrix of used models
confusionmatrix_list <- list(DT = cm_decisiontree,GLM = cm_glm, KNN = cm_knn,
                             SVM = cm_svm, NV = cm_naivebayes,
                             RF= cm_randomforest,LDA = cm_lda)
# confusionmatrix_list

confusionmatrix_list_results <- sapply(confusionmatrix_list, function(x) x$byClass)
confusionmatrix_list_results %>% knitr::kable()
```

	DT	GLM	KNN	SVM	NV	RF	LDA
Sensitivity	0.7732843	0.6348039	0.5502451	0.6029412	0.6544118	0.8321078	0.9628205
Specificity	0.3395990	0.6616541	0.8082707	0.7067669	0.6528822	0.4899749	1.0000000
Pos Pred Value	0.5449050	0.6573604	0.7458472	0.6776860	0.6584464	0.6252302	1.0000000
Neg Pred Value	0.5942982	0.6392252	0.6373518	0.6351351	0.6488169	0.7405303	0.9663963
Precision	0.5449050	0.6573604	0.7458472	0.6776860	0.6584464	0.6252302	1.0000000
Recall	0.7732843	0.6348039	0.5502451	0.6029412	0.6544118	0.8321078	0.9628205
F1	0.6393110	0.6458853	0.6332863	0.6381323	0.6564229	0.7139853	0.9810581
Prevalence	0.5055762	0.5055762	0.5055762	0.5055762	0.5055762	0.5055762	0.4832714
Detection Rate	0.3909542	0.3209418	0.2781908	0.3048327	0.3308550	0.4206939	0.4653036
Detection Prevalence	0.7174721	0.4882280	0.3729864	0.4498141	0.5024783	0.6728625	0.4653036
Balanced Accuracy	0.5564417	0.6482290	0.6792579	0.6548540	0.6536470	0.6610414	0.9814103

6.1 Conclusions

- Based on the Balanced Accuracy of the models used to predict whether the user like or dislike a particular song, the LDA algorithm present better results. Nevertheless, due to too many similar values among variables it could be possible that the model is overestimating the likeability of certain songs.
- Data provide interesting insights on spotify's audio features which can be used for song composition and promotion.
- Data does not reveal a clearly patron or tendency in the musical taste of the user. Many songs have very similar audio features and still they could be target as a liked or disliked song.
- Audio features from the present dataset dont define the audio or sound experience of a track. There is a lot more that creates impression to a particular user such as lyrics, rhythm, current environment, among others.

7 REFERENCES

- Irizarry, Rafael A. (2020, June 06). Introduction to Data Science. Retrieved from https://rafalab.github.io/dsbook/
- GeorgeMcIntire. (2017, August 04). Spotify Song Attributes. Retrieved from https://www.kaggle.com/geomack/spotifyclassification
- Get Audio Features for a track n.d., Retrieved from https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/
- (Pitch Class 2020) Retrieved from https://en.wikipedia.org/wiki/Pitch_class
- Schuller, B. W. (2013). Intelligent audio analysis. Berlin Heidelberg: Springer.
- Trafton, A. (2016, July 13). Why we like the music we do. Retrieved from http://news.mit.edu/2016/music-tastes-cultural-not-hardwired-brain-0713
- Xie Y., Allaire J. & Grolemund G. (2020, April 26) R Markdown: The Definitive Guide. Retrieved from https://bookdown.org/yihui/rmarkdown/