Final_report

Nguyen Phuc Thai

01/05/2022

Contents

1 Executive summary

This report outlines the analysis and results for the projects of building a model for categorizing songs. The purpose of the project is to help Spotify categorizing songs into the correct genre, so that it can improve the song suggestion services to the end user.

Furthermore, Spotify also specified that it wants to know how speechiness, tempo, release year and danceability of songs can be used to identify a genre of a song, which we will explore in data analysis. The model will be built based on all the available data, where we will choose which predictor is appropriate to be used.

The report includes:

- Methods of analysis
- Data cleaning and exploratory analysis, which shows how songs in different genres has different popularity, speechiness and how song release in different years has different level of popularity and each period has different dominating genre.
- Preprocessing data before inputting them into the model
- Model building for three models: Linear discriminant analysis, K nearest neighbor and Random forest, where the latter two have hyper-parameters to be tuned

The outcome of the report is that Random forest is the best model out of the suggested three based on specificity, sensitivity and area under the curve. Yet with the focus of providing better service to customer by suggesting songs by their genre, our model could not handle the task at the possibly required level. Consequently, the model still requires considerable improvement before it can be put into practice.

2 Methods

The data used for the analysis is extracted from the "spotify_songs.csv" file available on Github. Specifically, we are interested in names of songs, their release date (from which the year was extracted), song popularity, danceability, speechiness, tempo and the variable that we are trying to predict, genres of songs.

The steps in our analysis includes:

- Cleaning the data so that it is fit to be used, including removing variables that are not desirable for the model.
- Exploratory data analysis, including both what the founders asked for specifically and some further exploration between predictors and response.
- Preprocessing the data and building model. Three types of models were considered, including linear discriminant analysis, random forest and k nearest neighbor (also referred to as KNN). The latter two had to be tuned, or refining their parameters to get the best possible models.
- Comparing models and choose the best one. Then the best model will be evaluated on a test dataset to test its predictive capability.

All analyses were done using R software.

3 Results

3.1 Data cleaning

From the original dataset, the data of interest was extracted and cleaned. This includes converting the genre to a factor column, or a column that the models could work on, extracting the year from the release date of the songs.

There are some variables that should be removed from the data, with the corresponding rationale provided as follows:

- Track ID, Album ID and Playlist ID are all removed, because they are unique ID of tracks, album and playlist, which will coincides with their respective names.
- Track name will not be used for analysis, since we are categorizing songs into category, and names are unique identifiers of songs.
- Album name will also be redundant for analysis. This is because if the model is put into practice, we are likely to use the models on new songs, which should come from a new album, and they are new level predictors to the model. And a model cannot work with new levels, except in the cases of numerical predictors (even with quantitative predictors, the new data cannot have values out of the predefined range).
- Playlists cannot be used, as genre is one factor used for updating playlists, or creating new ones according to Spotify's playlist editors (Spotify, 2020). Consequently, as editors needs to know the genre to put songs into playlists, new songs will come with unknown playlist, that our model cannot work with should playlist be a predictors.
- Playlist subgenre cannot be used even though it is a perfect predictors for genre as each subgenre belongs to only one genre. But if we do not know a genre of a song, we should not know its subgenre as well. But if we do, then it is possible to identify a genre without the needs of a complex model, hence, made this project a lot easier.
- Lastly, track artist is not sufficient to be used in this project as the majority of artists available in the have fewer than 10 songs. Filtering artists with more than 10 songs will trim the dataset down to only about 13,000 tracks, or remove more than half the available data. However, if this data is sufficient, then it could be a very strong predictor for the model, as artist do not usually specialized in all six music genres, and many specialized in one. Furthermore, when a new song is added to Spotify database not all new songs are performed by a debutant, and numerous are likely to be performed by a well-known singer. Yet, for a statistical model to work well, if it uses categorical predictors, like artist in our cases, each level of those predictors must have a good number of observations. Moreover, the data will be further sliced later on to create a training data and testing data to validate the model, which adds to the issue of having artists with too few songs in our dataset. Consequently, removing artist variable is preferable in this case, as the model is built better without it, and the resulting model will be able to classify songs performed by any artist. (But if the objective was to build model to classify genre of songs performed by artists that have a lot of tracks with Spotify, then using this variable is recommended.)

The code for data import and cleaning can be found in the importing and cleaning data section of the appendix.

3.2 Exploratory data analysis

Popularity and genre

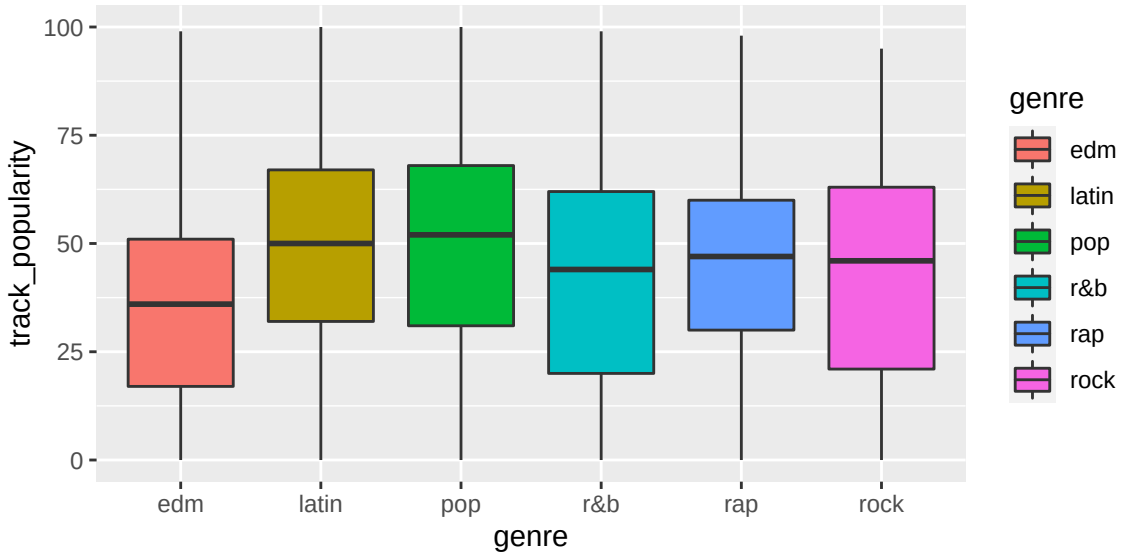

Figure 1: Popularity and genre

Figure 1 shows that pop songs are generally more popular than songs of other genres. Latin, rap and rock songs are as trendy as each other on average, while the genre R&B is in slightly less demand overall, and EDM songs are the least popular. However, all genres has songs that score close to either 100 or 0 on the popularity scale.

Genre and speechiness

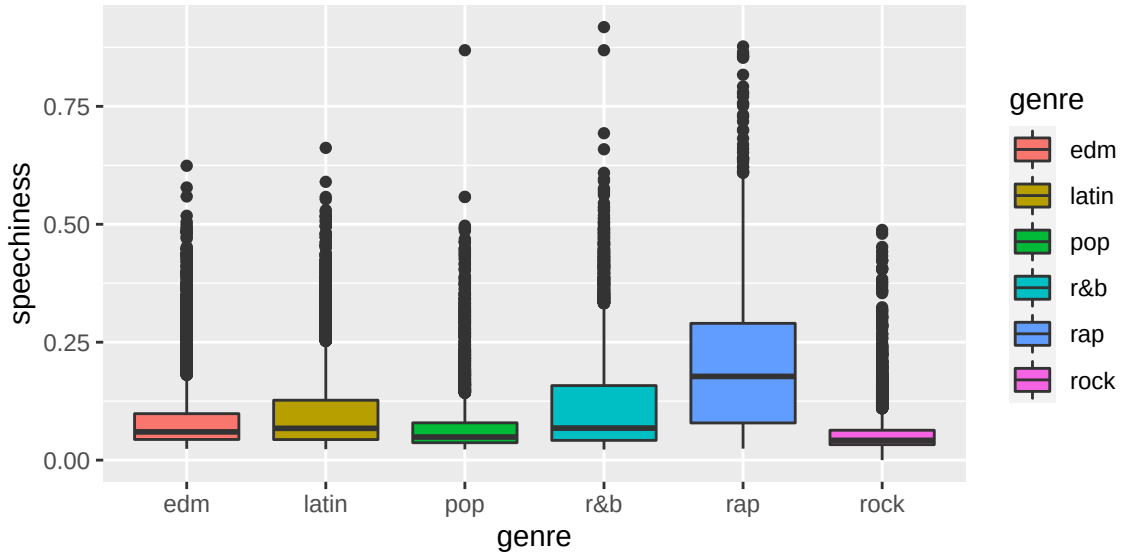

Figure 2: Speechiness and genre

The above figure shows that pop and rock songs are closed to not speechy at all,while edm, latin and R&B songs are slightly speechier than the former two genres. However, Rap stands out in this category as the only category that consists of more songs that score above around 0.2 on the speechiness scale than not, while nearly all songs from other genres score less than that level. This indicates that speechiness can be a very good indicator to distinguish rap from the rest

Track popularity over time

With songs attributes recorded between 1957 and 2020, there are years with very few tracks. Hence, to best visualize this, years with fewer than 150 songs were filtered out.

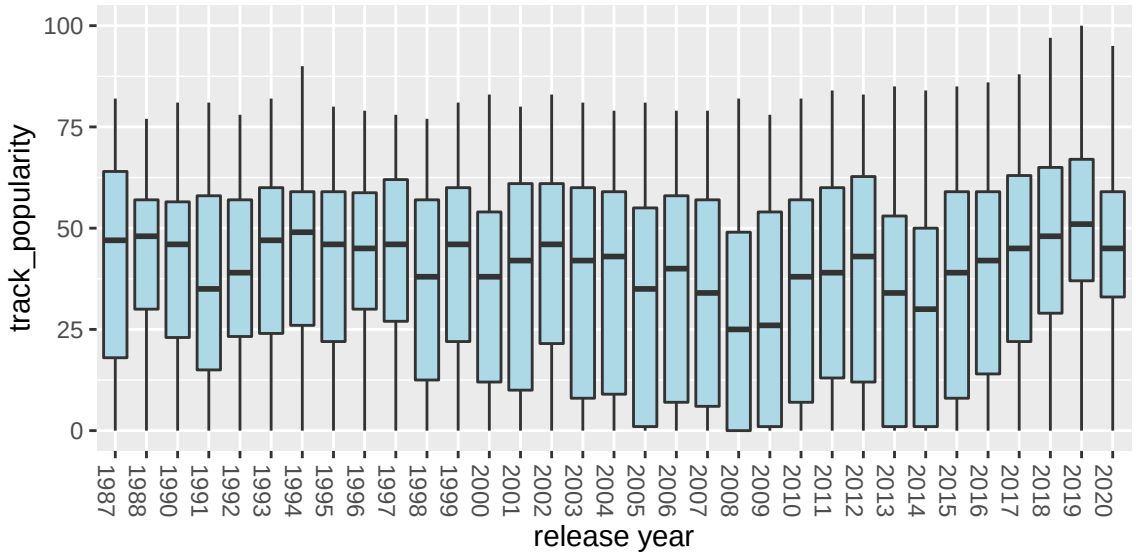

Figure 3: Track popularity over time

The popularity of songs seems to oscillate over time, which means periods with many songs in high demand were followed by periods that did not achieve the that. For instance, looking from the left of the graph, songs between 1987 and 1990 were better-received than those in 1991 and 1992, tracks overall popularity then rise again stay high for four or five years then follow by a period of ups and downs. Looking at the most recent years, tracks are seemingly gaining popularity, except for 2020.

Other findings

Aside from the requested exploration, other predictors were also visualized against music genre.

It was found that edm musics has the highest tempo among all genres, latin and R&B have the slowest tempo in general, while pop, rock and rap songs have similar tempo on average. Consequently, it could be a good predictor for distinguishing music genre, even though the general differences are not too significant. This is shown in figure 5 in the appendix

Regarding danceability of songs, rock tracks are considerably less danceable compared to music from the other five genres. Rap and latin songs are the easiest to dance to on average, followed by R&B, while edm and pop are on par on average. This is seen from graph 6 in the appendix

Lastly, the year when songs were released seems to reveal some very interesting patterns, as shown below:

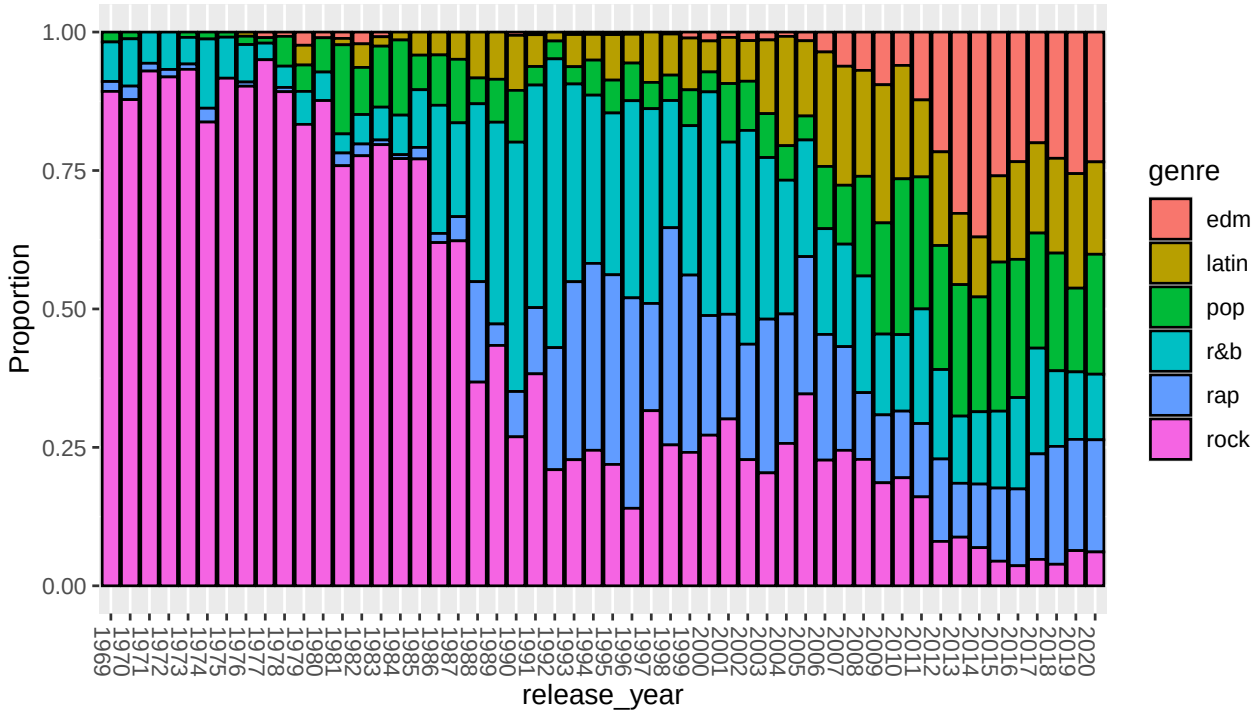

Figure 4: music genre over time

Graph 4 shows the proportion of songs from each category in any given year, and only years with 50 songs recorded are included in the graph. Rock was very dominant from the early 70s until mid 80s, making up for about 75% or more of the songs published in each year during that period. For other genres, R&B was very dominant from the early 90s to around 2005,

while EDM seems to only rise after 2010. Latin and rap music appear to start gaining attention after 1980, and it popularity has been relatively consistent until recently. Lastly, pop has been around since the earliest recorded time, but it only gains considerable attention during the last decade.

Codes for plots in this section are presented in the appendix

3.3 Data preprocessing

6000 tracks were extracted, consisting of 1000 songs for each genre, as specified by the founders. The predictors for the model include:

- Numerical: track popularity, danceability, energy, key, mode, speechiness, instrumentalness, liveness, valence, tempo and duration in milisecond
- Categorical: release year

Although mode seems like a categorical data as songs only have mode value of either 1 or 0. But since the data is already in the 0 and 1 structure, treating it as either numerical or categorical will make no difference to the model, since model will treat binary categorical variable the same way.

Furthermore, years are kept as categorical data as observation from figure 4 shows most of the time, music genres does not varies linearly with time. In other words, a genre will only be either more or less popular as time progress, if it is becoming less popular, it cannot go back. Furthermore, years with very few songs has to be combined and treated as a single year as models tend to not handle categorical variables with levels with too few observations well. Doing this will not affect the model much, as years with too few songs are very close to each other, and belongs to the earliest available songs in the dataset. Codes for this section and further notes and rationale on treatment of the release year can be found in the appendix.

3.4 Model building

To test the chosen model, from the 6000 observations, we used 4500 as the training data, and 1500 as the test data

Of the suggested models, only Linear Discriminant Analysis (LDA) does not have any hyperparameters to be tuned, or an aspect of the model that dictates how it will interacts with the predictors.

For K-nearest neighbors (KNN), we have to determine the optimal numbers of songs with similar characteristic and known genre that is optimal to make a prediction of the genre of a song with unknown genre.

For random forest, with 100 trees combined to build up the model, we have to tune the optimal number of predictors used in each tree and how the tree knows to stop making further split. For the latter, tree is built by keeping on making splits of observations based on the predictors, and we need to tune the number of observation such that the split can stop splitting a group.

To tune the models, we used bootstrapped datasets, or datasets that are created by drawing from the observations from the training data we have with replacement.

For KNN, the optimal number of neighbors is 100, for random forest, the number of predictors used in each tree is 4, and trees will stop splitting when there are 30 observation in a group.

To compare all three models, cross validation is applied, a method where we split the training data into groups (for this project, we chose five groups), build model on four and test the model on the other group. This is done five times, such that all group had a chance of being a tester and comparison of how the models overall performance is done to choose the best candidate model.

Finally, we refit the best model on the whole training data and test it performance on the test datasets. Codes for this section can again be found in the appendix for building models.

3.4.1 Model evaluation criteria and outcome

To assess the performance of the model, we used area under the curve (AUC), sensitivity and specificity.

Sensitivity measures for a song belongs to a particular genre, say Rock, what is the probability the model can classify it as a Rock song. For this problem, with six music genre, the model computes six sensitivity value and gives us the average of all of them.

Another criteria used is precision, which measure if a song is predicted to belong to a genre, what is the probability that that classification by our model is correct. The precision is again measured for genre, and the model's precision is the average precision of each of the six genre.

Specificity measures if a song do not belong to a particular genre , what is the probability that the model willl say that song does not belong to that genre, but the model does this by predicting the song to belong to the one of the five remaining genre. And the specificity given below will be the average for all genres. But because of how the model does this, this value will often be much higher than sensitivity.

Finally, with sensitivity and specificity, the model can then draw out ROC curves, or curves that measure model performance. Better models tend to have higher values of Area under the curve, hence our choice. Similar to sensitivity and specificity, the final AUC is the average of the AUCs for each individual genre.

(*Note the model predicts a song belong to which genre by first computing the probability of a song belongs to each of the available six, and then choose the one with the highest probability, and it can adjust its prediction by changing this criteria for classification, and hence comes at a different sensitivity and specificity*)

And the outcomes of the models, using the above mentioned criteria are shown in table 1

Table 1: Models result

Metric	LDA	KNN	Random Forest
precision	0.492	0.483	0.538
roc_auc	0.801	0.797	0.839
sensitivity	0.478	0.483	0.544
specificity	0.895	0.896	0.909

All codes for model buildings and comparison can be found in the appendix

4 Discussion

4.1 Chosen model

Looking at table 1, the superior model is Random Forest, as it outperforms the other two in all three criteria. It considerably outshines KNN and LDA in Area under the curve measure and sensitivity, and achieve a slightly higher specificity score.

However, all three models seems to be very good at recognizing a song to not belong to particular genre, but not at telling that a song belongs to the correct genre. Although this behaviour is expected, the main objective is correctly categorizing the genre of a song, which the company should be looking at precision and sensitivity rate closer to one that even our best model does not seem to be closed to. But looking at another perspective, with pure guessing, assuming that a random song is equally likely to belong to one of the six categories, is expected to be correct one time out of six guesses, or at about 0.166 chance, which is significantly worse than the model.

Then, the chosen model will be rebuilt on the whole training data and tested on the test dataset.

4.2 Evaluation on the test data

Table 2: Random forest performance on the test data

Metrics	Performance
roc_auc	0.853
sensitivity	0.572
specificity	0.914
precision	0.566

Table 2 shows how the final model performs on a test dataset, which the model has not interact with throughout the entire building process. Compared to how it performs in cross validation, random forest performed better on the test dataset than it did during cross validation. This

is unexpected, as models tend to perform worse on an entirely new data than how it does on training data. Yet, this is not impossible considering the randomness when data is split, and this result should be expected to worsen should the model be used in practice. Regardless, not getting significantly worse prediction shows that our training data and building process did not overfit the model. And with Random forest being the best model we could also identifies which predictors are more important to this model than others, shown in figure 8 in the appendix along with the codes for the final model. For this model the release year of the song is the most important predictor.

However, despite achieving adequate result in specificity and AUC, with the objective of recommending songs to users based on the genre, the company needs to have the right genre classification most of the time. Meanwhile, sensitivity and precision result means that our best model can only make the correct classification with the probability slightly better than 50%. Consequently, before being put in practice, the model needs to be improved significantly, which can be done by allowing for more predictors.

5 Conclusion

In this project, we have identified that among different genre has different level of popularity, and pop has the highest while EDM has the lowest score in that regards. Moreover, throughout the recorded periods, popularity of all recorded tracks was relatively volatile.

We also discovered that rap stood out as the most speechy genre, whereas others has similar level of speechiness. Furthermore, each genre has different periods when they became very dominant and trendy. Lastly, different genre has slight different tempo and danceability level on average, meaning they could be useful for model building.

Regarding model building, K nearest neighbor performs well for this project's purpose with 100 neighbors used, while random forest works well with four predictor for each tree, and minimum number of observation of 30.

And out of the three suggested model, random forest is the best option. Despite performing very well in two of the three criteria, the model fails the most relevant category which is the ability to correctly classify a song.

If Spotify chooses to continue this project, the model will need to be fitted and tested on the whole dataset rather than a portion of it to accommodate the low computational power. Consequently, the model still need substantial refinement before Spotify can use it.

5.1 Limitations

Our process also contains some limitations:

- With limited computational power, only a portion of the songs were used, the predictive power of the model could be improved by having more data used to build it.

- With release years being a categorical variable, prediction is not possible for songs that make their debuts after 2020. This can be fixed with treating this variable as numeric and apply polynomials transformation to it to account for the facts that genre can have multiple periods of being trendy. Or the periods could be grouped together such that multiple years are in each period, and future year could also be classified accordingly (such as classical times for songs and contemporary time for song). But the former requires further tuning to choose the appropriate power for the polynomials, while the latter requires some knowledge regarding periods of the musical industry, which we do not have access to.
- Also regarding release being categorical variable, with the limitations mentioned, we opted to keep it as we did. But it is not recommended to use qualitative data with such a high number of levels.
- A potentially helpful predictor, track artist, has to be removed for the reason mentioned in data cleaning.

6 Appendix

6.1 Setting up packages and working directory

```
setwd("C:/Users/thain/Google Drive/Adelaide uni/Maths7107/Final report")
library(readr)
library(ggplot2)
library(tidymodels)
library(tidyverse)
library(RColorBrewer)
library(inspectdf)
library(discrim)
library(kknn)
library(vip)
```

6.2 Importing and cleaning data

```
## importing
spotify_songs <- read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/

## Choosing data of interest (including those that can be used
## to verify that observations are duplicate)
song_data<-spotify_songs%>%
  dplyr::select(-track_id,-track_album_id,-playlist_id,
                -track_name,-track_album_name,
                -playlist_subgenre,-playlist_name)

## cleaning data and removing unused column
song_data_cleaned<-song_data%>%
  mutate(release_year=as.factor(substr(track_album_release_date,
                                       1,4 )), ##Extracting year
         genre=factor(playlist_genre))%>%#(factorize genre)
  distinct()%>% ##Check for duplicate, non found
  dplyr::select(-track_album_release_date ,-playlist_genre,
                -track_name,-track_artist)
```

Checking number of song performed by artist who have more than 10 song in the dataset

```
song_artist<-song_data%>%
  add_count(track_artist)%>%
```

```
    filter(n>=10)
nrow(song_artist)
```

```
## [1] 13362
```

Click here to go back to the data cleaning section

6.3 Codes for EDA

Graphs included in Methods section

```
#Popularity vs genre

song_data_cleaned%>%
  ggplot(aes(x=genre,y=track_popularity,fill=genre))+
  geom_boxplot()


#Speechiness vs genre
song_data_cleaned%>%
  ggplot(aes(x=genre,y=speechiness,fill=genre))+
  geom_boxplot()


## Popularity over time




song_data_cleaned%>%
            add_count(release_year,
              name='year_count')%>%
  filter(year_count>150)%>%
  ggplot(aes(x=as.factor(release_year),y=track_popularity,
             fill=as.numeric(release_year)))+
  geom_boxplot()+
  theme(axis.text.x = element_text(angle = -90, hjust=0),
        legend.position="none")+
  xlab('release year')+
  scale_fill_gradient(low='lightblue',high='lightblue')


## Genre vs time
```

```
song_data_cleaned%>%
  add_count(release_year,
            name='year_count')%>%
  filter(year_count>50)%>%
  ggplot(aes(x = release_year,
                                fill = genre)) +
  geom_bar(position = "fill", col = "black") +
  theme(axis.text.x = element_text(angle = -90, hjust=0))+
  ylab('Proportion')
```

Graphs that was referred to and their code

Click here to go back to the EDA section

```
song_data_cleaned%>%
  ggplot(aes(y=tempo,x=genre,fill=genre))+
  geom_boxplot()
```

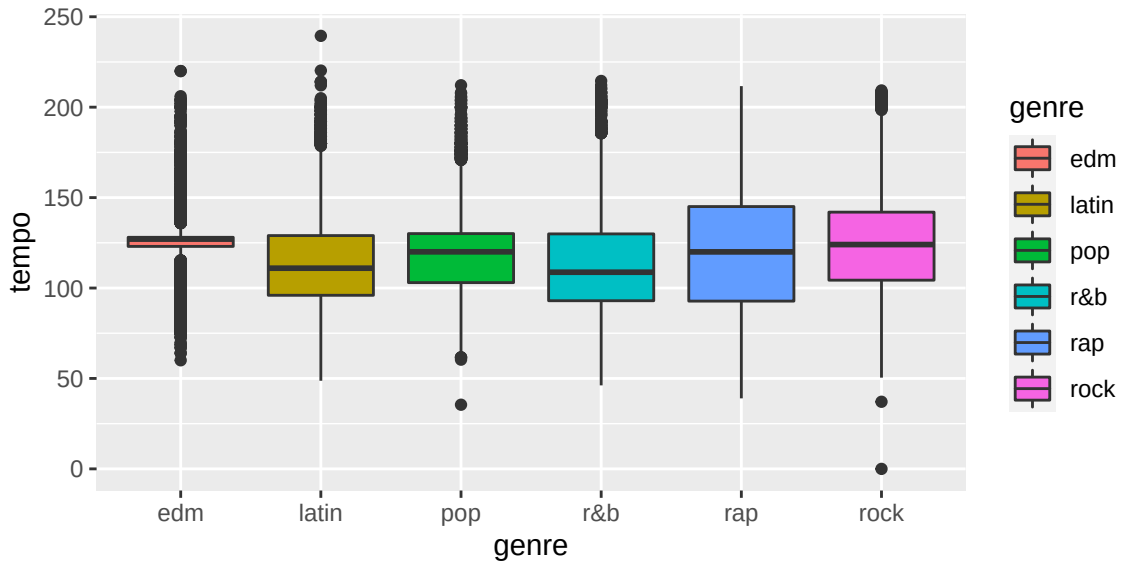

Figure 5: Tempo vs genre

```
song_data_cleaned%>%
  ggplot(aes(y=danceability,x=genre,fill=genre))+
  geom_boxplot()
```

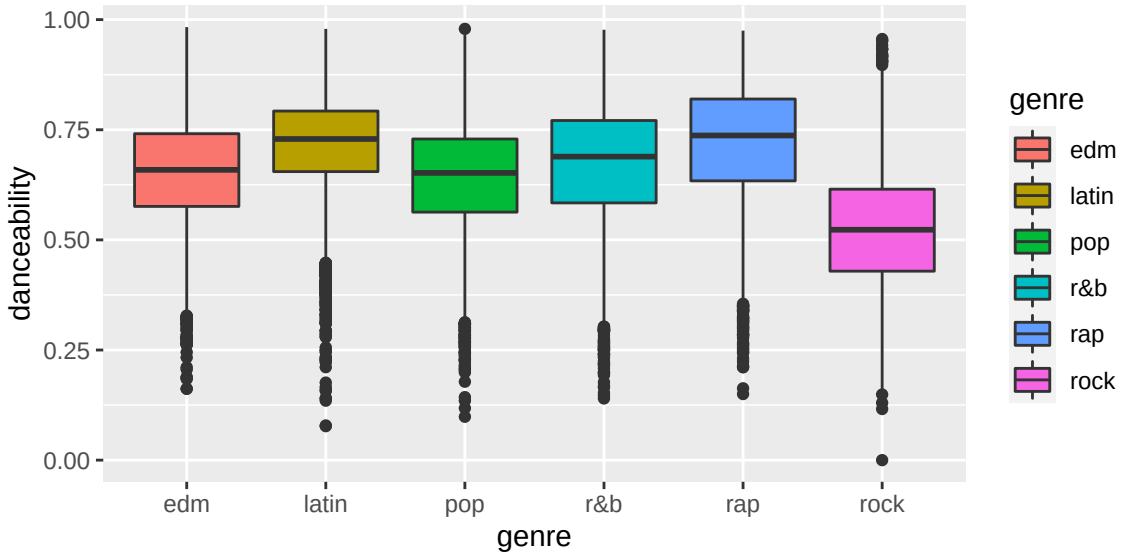

Figure 6: Danceability vs genre

Click here to go back to the EDA section

6.4 Codes for data preprocessing

Further notes on treatment of the release year predictor

Some release years with very few songs had to be combine together from the 6000 extracted songs (for this project, we combined years with fewer than 15 songs). This is done as models are likely to fail if years with too few song is kept, as when splitting data for train and test, the model will not work if testing data contain years that training data do not.

Furthermore, 15 is chosen for the following reasons:

- This cut off enables the model to run without any error, only for this particular 6000 observations.
- Years with fewer than 15 are very closed to each other, and also belongs to the earliest time recorded in the dataset (in the early 1980s or before that). These year can be assumed to be relatively similar in terms of the genre trends as seen in figure 4 , and the lumped year would not combine very different time periods. As shown in the graph below

```
song_data1%>%
  group_by(release_year)%>%
  count()%>%
  filter(n<21)%>%
  ggplot(aes(x=release_year,y=n))+
  geom_col()+
  geom_hline(yintercept = 15,color='red')+
  theme(axis.text.x = element_text(angle = -90, hjust=0))
```

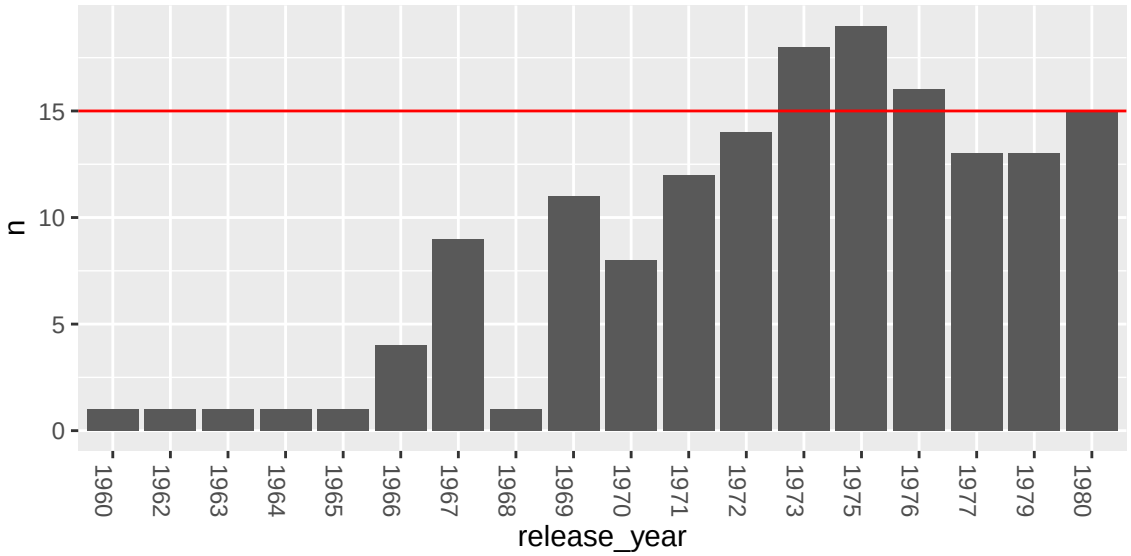

Figure 7: Lumped year (Years with number of songs below the red line)

```
# extracting the data for analysis
set.seed(1846505)

song_data1<-song_data_cleaned%>%
  group_by(genre)%>%
  slice_sample(n=1000)

## An additional steps for 'lumping' years with limited songs is needed so that linear

release_year_lumped=tibble(
  release_year_lumped=fct_lump_min(song_data1$release_year,15))


song_data1<-cbind(song_data1,release_year_lumped)


# Splitting data
set.seed(1846505)
song_split<-initial_split(song_data1,strata=genre)
song_train<-training(song_split)
song_test<-testing(song_split)




# Splitting data
set.seed(1846505)
```

```
song_split<-initial_split(song_data1,strata=genre)
song_train<-training(song_split)
song_test<-testing(song_split)


# Preprocessing data

# train
song_recipe<-recipe(genre~.,data=song_train)%>%
  step_normalize(danceability,tempo,speechiness,track_popularity,
                 loudness,mode,energy,key,acousticness,
                 instrumentalness,liveness,valence,
                 duration_ms)%>%
  step_corr()%>%
  prep()



song_train_processed<-juice(song_recipe)%>%
  mutate(release_year_lumped=factor(release_year_lumped))

#test
song_test_processed<-bake(song_recipe,song_test)%>%
  mutate(release_year_lumped=factor(release_year_lumped))
```

Click here to go back to the data preprocessing section

6.5 Codes for model building

Linear discriminant analysis

```
criteria=metric_set(roc_auc, sensitivity, specificity,precision)

doParallel::registerDoParallel()

lda_spec<- discrim_linear( mode = "classification" ) %>%
  set_engine( "MASS" )
lda_cv<-fit_resamples(object=lda_spec,
          preprocessor =recipe(genre~.,
                       data=song_train_processed%>%
                         dplyr::select(-release_year)),
                       resamples = song_train_cv,
          metrics=criteria)
```

Bootstrapping for KNN and random forest

```
set.seed(1846505)
song_boots <- bootstraps( song_train_processed, times = 10,  strata = genre )
```

Creating cross validation folds

```
set.seed(1846505)
song_train_cv<-vfold_cv(song_train_processed,v=5)
```

KNN

```
## Model specification
knn_spec<-nearest_neighbor( mode = "classification",
                                    neighbors = tune())%>%
  set_engine('kknn')

## Tuning number of neighbors
neighbor_grid <- grid_regular( neighbors(range=c(1,100)),
                               levels = 20)
doParallel::registerDoParallel()
knn_tuned <- tune_grid( object = knn_spec,
                        preprocessor = recipe(genre ~ . ,
                                    data = song_train_processed%>%
                         dplyr::select(-release_year)),
                        resamples = song_boots,
                        grid = neighbor_grid )

#Choosing the optimal number and finalizing the model
best_knn_auc <- select_best( knn_tuned, "roc_auc" )


final_knn_song <- finalize_model( knn_spec, best_knn_auc )


# Testing the model with cross validation
knn_cv<-fit_resamples(object=final_knn_song,
                      preprocessor =recipe(genre~.,
                                           data=song_train_processed%>%
                        dplyr::select(-release_year)),
                      resamples = song_train_cv,
              metrics=criteria)
```

Random forest

```r
# Model specification
rf_spec <- rand_forest( mode = "classification",
                            trees = 100,
                            mtry = tune(),
                            min_n = tune()) %>%
  set_engine( "ranger", importance = "permutation")

# Tuning the model
rf_grid <- grid_regular( finalize( mtry(), song_train_processed %>% select( -genre ) ),
                            min_n(),
                            levels = 5)


doParallel::registerDoParallel() # This makes macs run a little faster
rf_tuned <- tune_grid( object = rf_spec,
                       preprocessor = recipe(genre ~ . , data = song_train_processed%>%
                        dplyr::select(-release_year)),
                       resamples = song_boots,
                       grid = rf_grid )

# Choosing the optimal parameters and finalizing the model
best_rf_auc <- select_best( rf_tuned, "roc_auc" )

final_rf_song <- finalize_model( rf_spec, best_rf_auc )

# Testing the model with cross validation
rf_cv<-fit_resamples(object=final_rf_song,
                     preprocessor =recipe(genre~.,
                                          data=song_train_processed%>%
                                            dplyr::select(-release_year)),
                     resamples = song_train_cv,
              metrics=criteria)
```

Click here to go back to the model building section

6.6 Codes for final model

```r
## Fitting the model on the whole dataset
final_model<-final_rf_song%>%
  fit(genre~.,data=song_train_processed%>%
        dplyr::select(-release_year))
```

```
## Prediction on the test set

# Probability prediction
prob_pred<-predict(final_model,song_test_processed,type='prob')

# Classification of the model
song_pred<-predict(final_model,song_test_processed)

# Combining the above
song_pred<-bind_cols(true=song_test_processed$genre,
                     predicted=song_pred,
                     prob_pred)


#  sensitivity and specificity
categorical_metrics <- metric_set(sensitivity, specificity,precision)
outcome<-song_pred%>%
  categorical_metrics(
    truth=true,
    estimate=.pred_class
  )
# AUC and combining all results to a table
outcome=rbind(song_pred %>%
  roc_auc(
    truth = true,
    estimate = .pred_edm,.pred_latin,.pred_pop,`.pred_r&b`,
    .pred_rap,.pred_rock
  ),outcome)
outcome=outcome[,c(1,3)]
colnames(outcome)=c('Metrics','Performance')
```

Variable importance plot

```
final_model%>%
  vip()
```

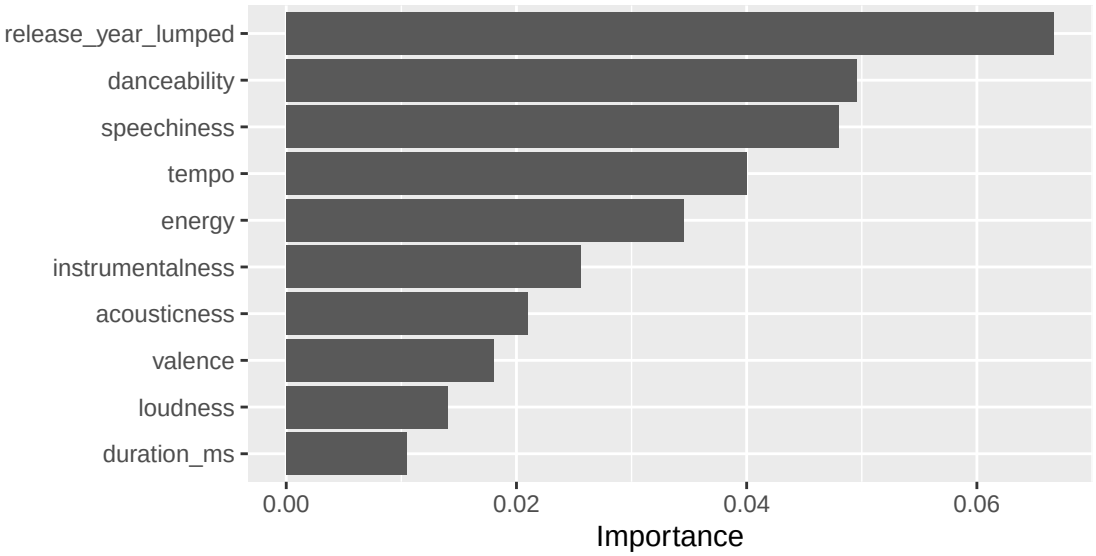

Figure 8: Variable importance plot

ROC plots for the final model

```
song_pred %>%
  roc_curve(
    truth = true,
    estimate = .pred_edm,.pred_latin,.pred_pop,`.pred_r&b`,
    .pred_rap,.pred_rock
  ) %>%
  autoplot()
```

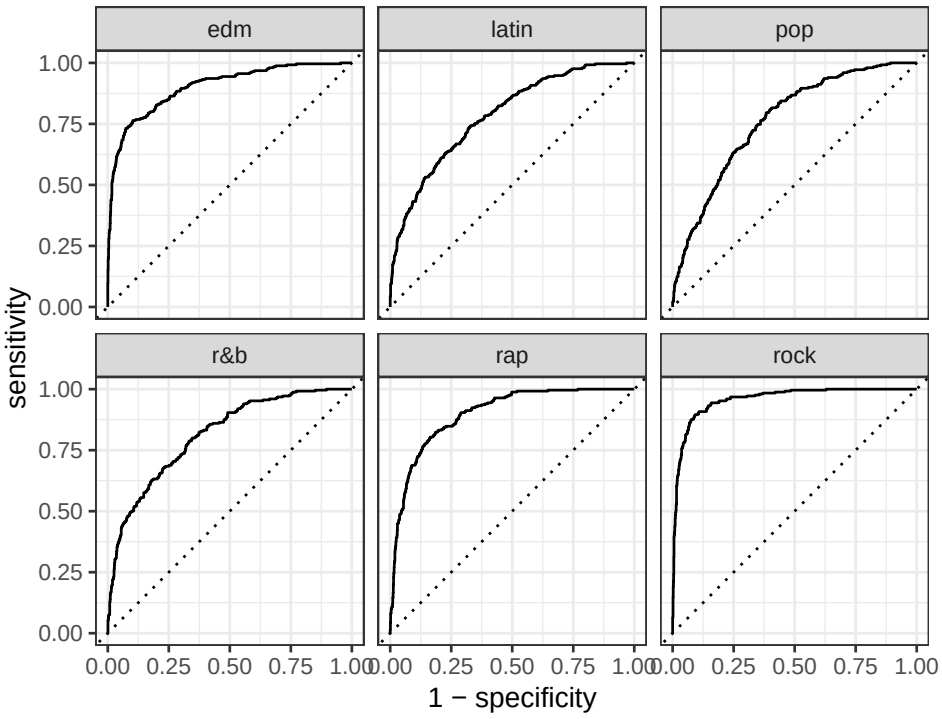

Figure 9: ROC plots of each genre of the final model

Click here to go back to the discussion section

7 Reference

Spotify 2020, *Behind the Playlists: Your Questions Answered by Our Playlist Editors*, Viewed 22 April 2022, https://artists.spotify.com/en/blog/behind-the-playlists-your-questions-answered-by-our-playlist-editors