Using Raw Streaming Data to Collect Deeper Data Attributes

Luka Vukovic

Import Libraries

```
library(stringr) # whitespace replacement
library(glue) # convenience
```

```
## Warning: package 'glue' was built under R version 4.0.3
```

```
library(wrappify) # lib for API wrapping
library(httr) # querying
```

```
## Warning: package 'httr' was built under R version 4.0.3
```

```
library(jsonlite) # json parsing
```

```
## Warning: package 'jsonlite' was built under R version 4.0.4
```

```
library(tidyverse) # wrangling
```

```
## Warning: package 'tidyverse' was built under R version 4.0.3

## Warning: package 'ggplot2' was built under R version 4.0.4

## Warning: package 'readr' was built under R version 4.0.4

## Warning: package 'dplyr' was built under R version 4.0.4

## Warning: package 'forcats' was built under R version 4.0.4
```

```
library(data.table) # wrangling
```

```
## Warning: package 'data.table' was built under R version 4.0.4
```

Import All Raw Data Files

Since there are multiple streaming CSV's for each user's data, we can use a recursive function to just import all of them into a list.

```
# Function that recursively imports all JSON files in
# a given directory and saves them to a list of dataframes

import_jsons <- function(path) {

    # Get list of files in path directory
    data_files <- list.files(path = path, recursive=T)

    # Function to pull JSON data into R
    data_puller <- function(path, file) {
        data <- fromJSON(glue('{path}/{file}'))
    }
    # Apply function over all directory files
```

```r
    RawSpotify <- sapply(data_files, data_puller, path = path)

    RawSpotify
}

usr1 <- import_jsons('Raw_Data/User1')
usr2 <- import_jsons('Raw_Data/User2')

c('File Names:', names(usr1))
```

```
## [1] "File Names:"             "StreamingHistory0.json" "StreamingHistory1.json"
## [4] "YourLibrary.json"
```

```r
c('File Names:', names(usr2))
```

```
## [1] "File Names:"             "StreamingHistory0.json" "StreamingHistory1.json"
## [4] "StreamingHistory2.json" "YourLibrary.json"
```

Streamed Tracks

Notice how there are multiple streaming files. Here we manually bind them based on the number of streaming files in each user's data.

```r
usr1_streams <- rbind(usr1$StreamingHistory0.json, usr1$StreamingHistory1.json)
usr2_streams <- rbind(usr2$StreamingHistory0.json, usr2$StreamingHistory1.json, usr2$StreamingHistory2.

paste('user1 streams:', nrow(usr1_streams))
```

```
## [1] "user1 streams: 18811"
```

```r
paste('user2 streams:', nrow(usr2_streams))
```

```
## [1] "user2 streams: 27491"
```

```r
head(usr2_streams)
```

```
##            endTime      artistName
## 1 2020-01-22 00:00           Ekali
## 2 2020-01-22 00:03          Mercer
## 3 2020-01-22 00:09        Pink Guy
## 4 2020-01-22 00:12  Carpenter Brut
## 5 2020-01-22 00:15          Tchami
## 6 2020-01-22 00:20 Golden Features
##                                                 trackName msPlayed
## 1 Babylon (feat. Denzel Curry) - Skrillex & Ronny J Remix   156923
## 2                                                    Boss   178345
## 3                             Fried Noodles (Getter Remix)   351428
## 4                                           Roller Mobster   214351
## 5                                                    Zeal   186666
## 6                                                 Do You?   297903
```

Library Tracks

Notice the `$YourLibrary` component has multiple sub list items like albums, shows, episodes, other, and tracks.

We only want to extract the tracks component since we're interested in their music taste.

```r
# sub list items
names(usr1$YourLibrary.json)
```

```
## [1] "tracks"       "albums"       "shows"        "episodes"     "bannedTracks"
## [6] "other"
```

```r
usr1_lib <- usr1$YourLibrary.json$tracks
usr2_lib <- usr2$YourLibrary.json$tracks

# rename columns for consistency
names(usr1_lib)[1] <- 'artistName'
names(usr2_lib)[1] <- 'artistName'
names(usr1_lib)[3] <- 'trackName'
names(usr2_lib)[3] <- 'trackName'

paste('user1 tracks:', nrow(usr1_lib))
```

```
## [1] "user1 tracks: 9135"
```

```r
paste('user2 tracks:', nrow(usr2_lib))
```

```
## [1] "user2 tracks: 1810"
```

```r
head(usr2_lib)
```

```
##      artistName                            album
## 1          Home                 Before the Night
## 2    Technikore                           Freakz
## 3    DJ Premier Beats That Collected Dust Vol. 2
## 4  Elder Island                     Elder Island
## 5      Camellia                           4orce!
## 6 Harris & Ford         Hard, Style & Volksmusik
##                              trackName
## 1             We're Finally Landing
## 2                            Freakz
## 3                            Change
## 4                   The Big Unknown
## 5 Can I friend you on Bassbook? lol
## 6          Hard, Style & Volksmusik
```

Getting Genres from Artist Names

Since thousands of artists need to be queried iteratively, this cell may take several minutes to run.

```r
# Function to query genres via the Wrappify API wrapper
get_artists_genres <- function(artist_vector) {

    # unique artists to avoid redundant API queries
    artists <- unique(artist_vector)

    # remove white space
    artists <- str_replace_all(artists, pattern=' ', replacement='')

    # API token via wrappify
    auth_token = getAuthenticationToken()

    # collection
    artistGenres <- NULL

    for(i in 1:length(artists)) {
      # query
      url <- glue('https://api.spotify.com/v1/search?q={artists[i]}&type=artist&limit=1')
      response <- GET(url, add_headers(Accept = 'application/json',
                                        Authorization = paste('Bearer', auth_token)))
      # format data and append
      genres <- try(unlist(content(response)$artists$items[[1]]$genres))
      if (is.null(genres) | class(genres)=='try-error') { genres <- 'Unknown' }
      artistGenres <- rbind(artistGenres, data.frame(artistName = artists[i], genres = genres))
      Sys.sleep(0.1)
    }

    artistGenres

}
```

Getting Music Metadata for Tracks

In addition to genre information, we can get song metadata such as danceability, energy, and valence. To do this we need to convert track names into their respective IDs so two functions are coded below, one for the IDs and one for the audiofeatures.

Note running this cell can take several minutes to several hours to run since Spotify's track search api queries iteratively.

```r
# Function to convert vector of song names to Spotify IDs
get_track_ids <- function(track_vector) {

  tracks <- str_replace_all(track_vector, pattern=' ', replacement='%20')

  # API token via wrappify
  auth_token = getAuthenticationToken()

  track_ids <- data.frame(trackName = NULL, id = NULL)

  for (i in 1:length(tracks)) {
        # query
        url <- glue('https://api.spotify.com/v1/search?q={tracks[i]}&type=track&limit=1')
        response <- GET(url, add_headers(Accept = 'application/json',
                                          Authorization = paste('Bearer', auth_token)))

        id <- try(content(response)$tracks$items[[1]]$id)

        if (is.null(id) | class(id)=='try-error') { id <- NA }

        tmp <- data.frame(trackName = track_vector[i], id = id) # 10 & 12 are name and id
        track_ids <- rbind(track_ids, tmp)
        Sys.sleep(0.1)
  }

  track_ids
}
```

```
# Function that get audio features of a song whether you have the ID or not
get_track_metadata <- function(track_vector, track_ids = NULL) {

    tracks <- track_vector
    n_tracks <- length(tracks)

    # batch since Spotify can only do 50 IDs at a time
    batches <- ceiling(n_tracks/50)
    last_batch_length <- n_tracks%%50

    if (is.null(track_ids)) {
      track_ids <- get_track_ids(tracks) # this is a choke point for speed
    } else {    }  # do nothing

    # API token via wrappify
    auth_token = getAuthenticationToken()

    # collection
    track_features <- NULL

    for (batch in 1:batches) {

        start_idx <- 50*(batch-1)+1

        if (batch < batches) {
            end_idx <- start_idx + 50
        } else {
            end_idx <- start_idx + last_batch_length
        }

        # query
        ids <- paste(track_ids[start_idx:end_idx], collapse = ',')
        url <- glue('https://api.spotify.com/v1/audio-features?ids={ids}')
        response <- GET(url, add_headers(Accept = 'application/json',
                                         Authorization = paste('Bearer', auth_token)))
        # format data and append
        batch_features <- rbindlist(lapply(content(response)$audio_features, data.frame))
        track_features <- rbind(track_features, batch_features)
        Sys.sleep(0.1)
    }

    track_features <- track_features[, c(1:11, 13, 17, 18)] # index all

    track_features <- left_join(data.frame(trackName = track_vector, id = track_ids), track_features, b
    track_features
}
```

Combining Get Audio Features and Get Genres

In order to predict user behaviour, we can do two things. Either we learn the user's personality and predict behaviour off of static library information, or we extract how the user's personality changes over time from their streaming data which is more informative for predicting behaviour.

For now, we've used only extracted information from the user's library (liked songs) capturing their genre and audio feature preferences. This will give us only probabilistic preferences for user music taste.

Streaming data extraction for user preferences over time wasn't included in the project simply because **streaming data is substantially larger than library data and would take hours to fully query for audio features.** Even if we don't model preferences over time, a fair assumption to make is that general user music preferences don't change drastically in short periods of time, so we'll try modeling their behaviours solely based on the static image of their song library.

Another important point, is using stream data to collect play counts, then applying the play counts as a weight to audiofeatures. That way, song features are weighed according to user preference by play counts. Otherwise, we're assuming each liked song in the user library is liked equally, which is often not true.

```
# Takes a dataframe with artistNames, trackNames, and queries for each tracks
# audiofeatures and each artist's genres
get_genres_features <- function(artist_track_frame) {
  # must have $artistName, $trackName columns

  # remove whitespace for joining
  artist_track_frame$artistName <- str_replace_all(artist_track_frame$artistName, pattern=' ', replaceme

  # get genres
  genres <- get_artists_genres(artist_track_frame$artistName)

  track_ids <- try(artist_track_frame$id)

  if (is.null(track_ids)) {
    # get track_ids and features
    features <- get_track_metadata(artist_track_frame$trackName)
  } else {
    # get features using existing IDs
    features <- get_track_metadata(artist_track_frame$trackName, track_ids = track_ids)
  }

  # join features on track name
  artist_track_frame <- left_join(artist_track_frame, features, by = c('trackName'='trackName', 'id' =

  # join genres on artistName
  artist_track_frame <- left_join(artist_track_frame, genres, by = c("artistName" = "artistName"))

  artist_track_frame

}
```

Query for Genres and Audio Features for library and streams

Roughly 57 thousand queries need to be made. Leave this to run overnight as it may take several hours.

After running it once and realizing something was wrong with the joins, I'm re-running the queries but with the track IDs that were already captured to speed things up. We can do this by importing the file and joining IDs on names from the original raw data.

```
# User 1
#usr1_lib2 <- read.csv('../Enhanced_Data/User1/usr1_lib.csv')
#usr1_streams2 <- read.csv('../Enhanced_Data/User1/usr1_streams.csv')
```

```r
# User 2
#usr2_lib2 <- read.csv('../Enhanced_Data/User2/usr2_lib.csv')
#usr2_streams2 <- read.csv('../Enhanced_Data/User2/usr2_streams.csv')


# Keep trackNames and IDs
usr1_lib_ids <- usr1_lib2[,c(3,4)] %>%
                    group_by(trackName, id) %>%
                    summarise()
usr1_stream_ids <- usr1_streams2[,c(3,5)] %>%
                    group_by(trackName, id) %>%
                    summarise()

usr2_lib_ids <- usr2_lib2[,c(3,4)] %>%
                    group_by(trackName, id) %>%
                    summarise()
usr2_stream_ids <- usr2_streams2[,c(3,5)] %>%
                    group_by(trackName, id) %>%
                    summarise()

NA_remover <- function(df) {
  # Keep only rows without NA values
  df <- df[!rowSums(is.na(df)) >= 1, ]
  df
}

# Remove NAs
usr1_lib_ids <- NA_remover(usr1_lib_ids)
usr1_stream_ids <- NA_remover(usr1_stream_ids)
usr2_lib_ids <- NA_remover(usr2_lib_ids)
usr2_stream_ids <- NA_remover(usr2_stream_ids)

# Join to original frames
# Use right join so only songs with IDs are kept
usr1_lib <- right_join(usr1_lib, usr1_lib_ids, by = c('trackName' = 'trackName'))
usr1_streams <- right_join(usr1_streams, usr1_stream_ids, by = c('trackName' = 'trackName'))
usr2_lib <- right_join(usr2_lib, usr2_lib_ids, by = c('trackName' = 'trackName'))
usr2_streams <- right_join(usr2_streams, usr2_stream_ids, by = c('trackName' = 'trackName'))

# Note: roughly 20-30% of data is lost because queries can't consistenyl recover IDs for
# songs with words like [remastered] or [live] or symbols
```

```r
# User 1
usr1_lib2 <- get_genres_features(usr1_lib)
usr1_streams2 <- get_genres_features(usr1_streams)

# User 2
usr2_lib2 <- get_genres_features(usr2_lib)
usr2_streams2 <- get_genres_features(usr2_streams)

# Bug in join causing many many duplications of genre joins
# Address by grouping on all columns
grouper <- function(df) {
  grouped_df <- df %>%
                  group_by_all() %>%
                  summarize()
  grouped_df
}

usr1_lib3 <- grouper(usr1_lib2)
usr1_streams3 <- grouper(usr1_streams2)
usr2_lib3 <- grouper(usr2_lib2)
usr2_streams3 <- grouper(usr2_streams2)


## Export
write.csv(usr1_lib3, '../Enhanced_Data/User1/usr1_lib.csv', row.names=FALSE)
write.csv(usr1_streams3, '../Enhanced_Data/User1/usr1_streams.csv', row.names=FALSE)
write.csv(usr2_lib3, '../Enhanced_Data/User2/usr2_lib.csv', row.names=FALSE)
write.csv(usr2_streams3, '../Enhanced_Data/User2/usr2_streams.csv', row.names=FALSE)
```