Análise exploratória das músicas brasileiras por década

Henrique Reichert

27/10/2021

Introdução

Ao longo das últimas décadas, o Brasil passou por diversos cenários políticos e econômicos que acabam afetando a sociedade e se refletindo na cultura de nosso país. Ao longo das décadas de 60 a 80, a ditadura militar foi um dos eventos que estimulou mudanças no ritmo e nas letras das músicas brasileiras. Já nas décadas mais atuais, um outro cenário impera e também altera a forma com que consumimos áudio.

Para avaliar como ocorreu esta evolução da música brasileira ao longo das décadas, este código analisa algumas variáveis típicas da sonoridade dos aúdios e também a composição das letras das músicas.

Seguiremos este documento com o download dos dados do Spotify, em que se avaliam algumas estatísticas disponibilizadas pelo próprio API da empresa.

Em seguida, faremos a seleção das músicas por década e o download das letras destas músicas, usando um API da Vagalume. Nesta etapa será preciso um trabalho de limpeza dos dados e conexão dos dados obtidos pelo Spotify e pelo Vagalume.

Por fim, faremos uma análise de sentimentos das músicas e observação das palavras mais usadas em cada uma das décadas da música brasileira.

Carregamento de pacotes e download dos dados do Spotify

Os pacotes básicos para esta etapa são os seguintes:

- Tidyverse, para manipulação dos dados
- Spotifyr, para download das estatísticas do Spotify
- Ggridges, para gráficos

```
#Carregando pacotes básicos

library(tidyverse)
library(spotifyr)
library(ggridges)
```

Agora será preciso obter as credenciais para acesso ao API do Spotify. Para isso, é preciso ter um cadastro na plataforma e acessar o site (https://developer.spotify.com/). Nele, você clica em 'Dashboard' e em 'Create an App', coloque o nome e descrição que achar mais adequado e tenha acesso a sua Cliente ID e Client Secret.

```
#Definindo minha ID do API do Spotify

Sys.setenv(SPOTIFY_CLIENT_ID = 'XXXXXXXXXXXXXXXXX') #Insira sua Client ID
Sys.setenv(SPOTIFY_CLIENT_SECRET = 'XXXXXXXXXXXXXXXXX') #Insira sua Client Secret

access_token <- get_spotify_access_token()
```

Existem várias formas de obter os dados do Spotify, a documentação consta no site mencionado acima. No nosso caso, queremos fazer o download das músicas por playlist, sendo neste caso as playlists criadas pelo próprio Spotify e referente às principais músicas de cada década.

```
#Definindo o autor das playlists, no caso é o próprio spotify
playlist_username <- 'spotify'

#Definindo as URIs das playlists que queremos
playlist_uris <- c('37i9dQZF1DWZc0bJQiqY2O', #Brasil Anos 50
                   '37i9dQZF1DX1Qj33psNIU5', #Brasil Anos 60
                   '37i9dQZF1DXaI4UKLLl2Z7', #Brasil Anos 70
                   '37i9dQZF1DWVRje2pMSCzM', #Brasil Anos 80
                   '37i9dQZF1DX7v47Gt49bv4', #Brasil Anos 90
                   '37i9dQZF1DWZ2DWnNKZTXs', #Brasil Anos 2000
                   '37i9dQZF1DWUXCP3AaoWLP', #Brasil Anos 2010
                   '37i9dQZF1DX0FOF1IUWK1W') #Top Brasil, como proxy para músicas atuais de 2020

#Baixando os dados e deixando apenas as colunas de interesse para essa análise

playlist_audio_features <- get_playlist_audio_features(playlist_username, playlist_uris) %>%
  select('playlist_name','danceability','energy','loudness','speechiness','acousticness',
         'instrumentalness','valence','tempo','track.name','track.popularity','track.id', 'track.artists
```

Análise das músicas por década

Com os dados das músicas por década, podemos avaliar algumas diferenças da musicalidade a partir de estatísticas preparadas pelo próprio Spotify, são elas:

- Danceability, o quão dançável é a música, com base no ritmo, batidas por minuto e outros dados, varia de 0 a 1.
- Energy, representa a intensidade e atividade da música, um rock pesado tem alta energia, por exemplo, varia de 0 a 1 também.
- Loudness, é a média de decibeis que a música alcança, são músicas mais barulhentas e os valores variam entre -60 e 0 db.
- Acousticness, um índice de 0 a 1 para avaliar se a música é acústica.
- Valence, valor de 0 a 1 que representa a positividade da música, valores mais próximos de 1 são músicas mais felizes e eufóricas, do ponto de vista da sonoridade, e não da letra.
- Tempo, representa as batidas por minuto (BPM) da música.

Um detalhe importante é que o Spotify gera arquivos de apenas 50 músicas por vez. Neste caso, achamos razoável comparar as 50 principais músicas de cada década, então mantemos assim mesmo, trabalhando com 50 músicas para 8 playlists, resultando em 400 músicas.

Explicadas as variáveis, podemos avaliar em gráficos de histograma como se configuram as músicas nas décadas.

```r
#Renomeando as playlists para ficarem na ordem correta de visualização


playlist_audio_features[
  playlist_audio_features$playlist_name=='Top Brasil',1] <- "Brasil Anos 2020"
playlist_audio_features[
  playlist_audio_features$playlist_name=='Brasil Anos 90',1] <- "Brasil Anos 1990"
playlist_audio_features[
  playlist_audio_features$playlist_name=='Brasil Anos 80',1] <- "Brasil Anos 1980"
playlist_audio_features[
  playlist_audio_features$playlist_name=='Brasil Anos 70',1] <- "Brasil Anos 1970"
playlist_audio_features[
  playlist_audio_features$playlist_name=='Brasil Anos 60',1] <- "Brasil Anos 1960"
playlist_audio_features[
  playlist_audio_features$playlist_name=='Brasil Anos 50',1] <- "Brasil Anos 1950"
#As playlists de 2000 e 2010 não precisam ser renomeadas.



#Plotando gráficos comparativos das músicas das décadas

ggplot(playlist_audio_features, aes(x = valence,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
                         axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Valência das músicas por década")

ggplot(playlist_audio_features, aes(x = danceability,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
                         axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Dançabilidade das músicas por década")

ggplot(playlist_audio_features, aes(x = energy,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
                         axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Energia das músicas por década")

ggplot(playlist_audio_features, aes(x = loudness,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
```

```
                          axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Loudness das músicas por década")

ggplot(playlist_audio_features, aes(x = acousticness,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
                          axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Acústico das músicas por década")

ggplot(playlist_audio_features, aes(x = tempo,
                                    y = playlist_name,
                                    fill = playlist_name)) +
  geom_density_ridges(alpha = 0.5, jittered_points = TRUE) +
  theme_ridges() + theme(axis.title.x = element_blank(),
                          axis.title.y = element_blank()) +
  guides(fill=FALSE,color=FALSE) +
  ggtitle("Tempo (BPM) das músicas por década")
```

Download das letras das músicas

Aparentemente, há uma diferença notável nas letras das músicas mais antigas com as atuais. Para avaliar esta diferença com mais embasamento em dados, vamos fazer o download das letras destas músicas para poder comparar as décadas.

Para esta etapa, vamos precisar de alguns pacotes adicionais e do registro ao API do Vagalume.

Para o registro da API, é preciso seguir as etapas abaixo:

- Acessar o site https://auth.vagalume.com.br/ e fazer o cadastro,
- Entrar na área de APIs (https://auth.vagalume.com.br/settings/api/) e criar um novo aplicativo,
- Salvar a credencial do aplicativo.

```
# Pacotes adicionais necessários
library('vagalumeR') # Para download dos dados
library('bitsandends') # Para realizar aproximações de nomes e artistas

# ID do API da Vagalume
key_vagalume <- "XXXXXXXXXXXXXXXXXXXXXXXX" # Insira sua ID do Vagalume
```

Agora chega uma parte mais complicada. Não há uma conexão direta das músicas do Spotify com o Vagalume, isso gera duas dificuldades:

1- A playlist do Spotify obviamente não existe no Vagalume, então nós temos apenas um conjunto de músicas e artistas para fazer a busca no Vagalume,

2- A falta de um id obriga que a relação entre as duas bases seja realizada pelo nome da música, e já antecipamos que os nomes (tanto de artistas como de músicas) nem sempre estão iguais.

Para a primeira parte, de busca das músicas para download das letras, não encontramos uma alternativa de busca por nome da música, apenas por álbum ou por artista.

Assim, decidimos gerar uma lista de todos os artistas relacionados nas 400 músicas do Spotify, baixar as letras de todas as suas músicas e, por fim, filtrar apenas as músicas que constam na lista do Spotify.

É verdade que esta parte não parece muito eficiente, mas, por ora, não encontramos saída mais ágil para o download das músicas.

```
# Exportando a lista de artistas das músicas
artista <- do.call(rbind.data.frame, playlist_audio_features$track.artists) %>%
  select(id,name)
artista <- artista[!duplicated(artista$id),]
```

Um ponto importante para esta etapa e também para as futuras buscas e relações por músicas e artistas é a remoção dos acentos. Há alguns pacotes para isso, mas neste caso acabamos usando uma função própria para remoção dos acentos.

```
rm_accent <- function(str,pattern="all") {


  if(!is.character(str))
    str <- as.character(str)

  pattern <- unique(pattern)

  if(any(pattern=="Ç"))
    pattern[pattern=="Ç"] <- "ç"

  symbols <- c(
    acute = "áéíóúÁÉÍÓÚýÝ",
    grave = "àèìòùÀÈÌÒÙ",
    circunflex = "âêîôûÂÊÎÔÛ",
    tilde = "ãõÃÕñÑ",
    umlaut = "äëïöüÄËÏÖÜÿ",
    cedil = "çÇ"
  )

  nudeSymbols <- c(
    acute = "aeiouAEIOUyY",
    grave = "aeiouAEIOU",
    circunflex = "aeiouAEIOU",
    tilde = "aoAOnN",
    umlaut = "aeiouAEIOUy",
    cedil = "cC"
  )

  accentTypes <- c("´","`","^","~","¨","ç")

  if(any(c("all","al","a","todos","t","to","tod","todo")%in%pattern))
    return(chartr(paste(symbols, collapse=""),
                  paste(nudeSymbols, collapse=""), str))

  for(i in which(accentTypes%in%pattern))
    str <- chartr(symbols[i],nudeSymbols[i], str)

  return(str)
}
```

Agora, com a função de remover acentos, podemos gerar um loop para padronização do nome dos artistas e download das letras de cada um.

Vale mencionar que o loop abaixo vai gerar vários erros que vão interrompendo o download. Isto se deve à diferença dos nomes dos artistas na base do Spotify com a base do Vagalume, um exemplo claro é da dupla Sandy e Júnior, que na base do Vagalume consta sandy-junior (sem o e). Quando isso ocorrer, uma dica é ir diretamente ao site do Vagalume, procurar o nome do artista na seção de busca e verificar como consta a url da página do artista.

Deixamos duas linhas comentadas no código, a primeira é uma forma de renomear o artista e a segunda é o código de remoção do artista, quando não é encontrado um similar na base do Vagalume.

```
#Baixando as letras de todas as músicas para cada artista da lista
songs <- data.frame()

for(i in artista[,2]){
  autor <- rm_accent(i)
  autor <- gsub(" ","-",autor)
  autor <- gsub("&","e",autor)
  autor <- gsub('\\.','',autor)
  song <- autor %>% purrr::map_dfr(songNames)
  songs <- rbind(songs,song)
}

#Possíveis ajustes que precisam ser realizados nos nomes
# artista[artista$name=="Abel Ferreira E Seu Conjunto",2] <- 'Abel Ferreira'
# artista <- artista[!artista$name=="Garoto",]
```

Com a base de letras dos artistas, agora temos que filtrar esta base apenas para as músicas listadas na base original. Como comentamos acima, o único link entre o download das letras com a lista do Spotify é o nome da música, então é recomendável fazer um tratamento dos caracteres para letras minúsculas e sem acento.

```
# Eliminando maiúsculas e acentos das duas bases de conexão
songs <- songs %>% mutate(songsac = str_to_lower(song))
songs$songsac <- rm_accent(songs$songsac)

lista <- playlist_audio_features %>%
  mutate(songsac = str_to_lower(track.name)) %>% select(playlist_name,songsac)
lista$songsac <- rm_accent(lista$songsac)

letraemusica <- lista %>% inner_join(songs, by = 'songsac') %>%
  distinct(songsac, .keep_all = TRUE)
```

Perdemos cerca de 140 musicas aqui, sendo que 31 somente da década de 2010. o que não é muito legal. Para tentar minimizar isto, podemos usar uma medida de distância entre as listas de músicas, para corrigir manualmente alguns nomes que não estão dando match.

```
#Nomes para corrigir
nomes_corrigir <- lista %>% anti_join(songs, by = 'songsac') %>%
  distinct(songsac) %>% pull(songsac)

# Calculando as 'distâncias' entre as músicas da base de letras e as que deram unmatch
```

```r
dists <- songs$songsac %>%
  map(levenshteinSim, str1 = nomes_corrigir)

# Encontrando os nomes de músicas mais similares nas duas bases
ordered_dists <- dists %>% map_dbl(max)
max_dists <- dists %>% map_dbl(which.max)

# Filtrando as que tem similaridade > 0.70
indexes_min_dist <- which(ordered_dists > 0.70)
songs_min_dist <- songs$songsac[indexes_min_dist]
index_lyrics <- max_dists[which(ordered_dists > 0.70)]

# Salvando as mais similares em um data.frame
results_dist <- data.frame(from_lista = nomes_corrigir[index_lyrics],
                           from_songs = songs_min_dist)

# Corrigindo os nomes de algumas musicas
songs <- songs %>%
  mutate(songsac = case_when(
        songsac == 'era um garoto que como eu amava os beatles e os rolling stones' ~
          'era um garoto, que como eu, amava os beatles e os rolling stones',
        songsac == 'maria, maria' ~ 'maria maria',
        songsac == 'como uma onda (zen surfismo)' ~
          'como uma onda - zen-surfismo',
        songsac == 'recuerdos de ypacarai' ~ 'recordacoes de ypacaray',
        songsac == 'roda viva' ~ 'roda-viva',
        songsac == 'mas que nada' ~ 'mas, que nada!',
        songsac == 'o que e o que e ?' ~ 'o que e o que e?',
        songsac == 'mulher rendeira' ~ 'mule rendeira',
        songsac == 'mon amour, meu bem, ma femme' ~
          'mon amour, meu bem, ma famme',
        songsac == 'descobridor dos sete mares' ~
          'o descobridor dos sete mares',
        songsac == 'apenas um rapaz latino-americano' ~
          'apenas um rapaz latino americano',
        songsac == 'meu sangue ferve por voce' ~
          'meu sangue ferve for voce',
        songsac == 'amor, meu grande amor' ~ 'amor meu grande amor',
        songsac == 'dancing days' ~ "dancin' days",
        songsac == 'colcha de retalhos' ~ 'colcha de retalho',
        songsac == 'araketu bom demais' ~ 'ara ketu bom demais',
        songsac == 'vermelho (part. david assayag)' ~
          'vermelho (feat. david assayag)',
        songsac == 'a casa amarela' ~ 'casa amarela',
        songsac == 'que sorte a nossa' ~ 'que sorte a nossa - live',
        songsac == 'ninguem explica deus (part. gabriela rocha)' ~
          'ninguem explica deus (feat. gabriela rocha) - ao vivo',
        songsac == 'localiza ai' ~ 'localiza ai bb',
        songsac == 'faking love (ft. saweetie)' ~
          'faking love (feat. saweetie)',
        TRUE ~ songsac))
```

Agora, com a correção destes nomes, podemos refazer a conexão das músicas entre as duas bases e já realizar

a busca das letras das músicas pelo pacote do Vagalume.

```
# Refazendo a inner join com os nomes corrigidos
letraemusica <- lista %>% inner_join(songs, by = 'songsac') %>%
  distinct(songsac, .keep_all = TRUE)


# Baixando as letras das musicas
lyrics <-  letraemusica %>% pull(song.id) %>%
  map(lyrics, artist = "name", type = "id", key = key_vagalume) %>%
  map_dfr(data.frame)

# Inserir dados das playlists na base lyrics
lyrics <- left_join(lyrics[,-c(2,3,6:8)],letraemusica[,-3])
```

Análise das letras das músicas

Com as duas bases conectadas, agora podemos fazer análise das letras das músicas por década. Para isso, precisaremos de mais alguns pacotes adicionais.

```
library(tidytext) # Para manipulação dos dados de texto
library(lexiconPT) # Para download de stopwords e sentimento das palavras
library(wordcloud) # Para gerar wordclouds
```

A avaliação das letras também é mais eficaz quando eliminamos os acentos e trabalhamos apenas com letras minúsculas, então o primeiro passo é realizar uma limpeza dos dados e também para separá-las individualmente.

Já inserimos também uma sequência de comandos para gerar stopwords e elimná-las da lista.

```
# Colocando todas as palavras em minúsculo e retirando os acentos
lyrics <- lyrics %>%
  mutate(text = str_to_lower(text)) %>%
  mutate(text = rm_accent(text))

# Baixando uma lista de stopwords em pt
stopwords <- data.frame(word = tm::stopwords('portuguese'))

# Em alguns casos vimos uma mensagem de remoção da letra no site do Vagalume
lyrics$drop <- grepl("a musica de autoria",lyrics$text)
lyrics <- lyrics %>% filter(drop==FALSE) %>% select(-drop)

#Criando data frame vazio para inserir as palavras por ano
unnested <- data.frame()

# Quebrando as frases em palavras e removendo as stopwords
for (i in unique(lyrics$playlist_name)){
  unnest <- lyrics %>% filter(playlist_name == i) %>%
  select(text) %>%
  unnest_tokens(word, text, token = "ngrams", n = 1) %>%
  anti_join(stopwords, by = c("word" = "word"))
  unnest$decada <- i
  unnested <- rbind(unnested,unnest)
}
```

Agora podemos começar as análises das palavras mais comuns e do sentimento das músicas.

```
# Contando as palavras que mais apareceram
unnested %>% group_by(decada) %>%
  count(word) %>%
  arrange(desc(n)) %>%
  slice(1:10)

# Para gerar uma lista das palavras mais comuns por década,
# podemos criar uma função para facilitar a repetição
palavras_comuns <- function(ano){
    unnested %>% filter(decada==paste0('Brasil Anos ',ano)) %>%
    count(word) %>%
    # tirar palavras que aparecem pouco
    #filter(n < quantile(n, 0.999)) %>%
    top_n(n = 30) %>%
    ggplot(aes(reorder(word, n), n)) +
    geom_linerange(aes(ymin = min(n), ymax = n, x = reorder(word, n)),
                   position = position_dodge(width = 0.2), size = 1,
                   colour = 'darksalmon') +
    geom_point(colour = 'dodgerblue4', size = 3, alpha = 0.9) +
    coord_flip() +
    labs(x = paste0('Top 30 palavras mais comuns na década de ',ano),
         y = 'Contagem') +
    theme_bw(14)
}

palavras_comuns(1950)
palavras_comuns(1960)
palavras_comuns(1970)
palavras_comuns(1980)
palavras_comuns(1990)
palavras_comuns(2000)
palavras_comuns(2010)
palavras_comuns(2020)
```

Podemos também criar uma função para gerar as wordclouds de cada década.

```
wordcloud_ano <- function(ano){
  unnested %>% filter(decada==paste0('Brasil Anos ',ano)) %>%
    count(word) %>%
    with(wordcloud(word, n, family = "serif",
                   random.order = FALSE, max.words = 50,
                   colors = c("darksalmon", "dodgerblue4")))

}

wordcloud_ano(1950)
wordcloud_ano(1960)
wordcloud_ano(1970)
wordcloud_ano(1980)
wordcloud_ano(1990)
wordcloud_ano(2000)
```

```
wordcloud_ano(2010)
wordcloud_ano(2020)
```

A avaliação do sentimento das músicas por década pode ser feita de forma muito parecida com as funções acima.

```
# Buscando os sentimentos do pacote lexiconPT
sentiments <- oplexicon_v2.1 %>%
  mutate(word = term) %>%
  select(word, polarity)

# Juntando os sentimentos com as palavras presentes nas músicas
add_sentiments <- lyrics %>%
  select(text, playlist_name) %>%
  group_by_all() %>%
  slice(1) %>%
  ungroup() %>%
  unnest_tokens(word, text) %>%
  anti_join(stopwords, by = c("word" = "word")) %>%
  inner_join(sentiments, by = c("word" = "word"))


sentimento_ano <- function(ano){
add_sentiments %>% filter(playlist_name==paste0('Brasil Anos ',ano)) %>%
  group_by(polarity) %>%
  count(word) %>%
  filter(n < quantile(n, 0.999)) %>%
  top_n(n = 15) %>%
  ggplot(aes(reorder(word, n), n)) +
  geom_linerange(aes(ymin = min(n), ymax = n, x = reorder(word, n)),
                 position = position_dodge(width = 0.2), size = 1,
                 colour = 'darksalmon') +
  geom_point(colour = 'dodgerblue4', size = 3, alpha = 0.9) +
  facet_wrap(~polarity, scales = "free") +
  coord_flip() +
  labs(x = 'Top 15 palavras mais comuns',
       y = 'Contagens', title = "Sentimentos") +
  theme_bw(14)
}

sentimento_ano(1950)
sentimento_ano(1960)
sentimento_ano(1970)
sentimento_ano(1980)
sentimento_ano(1990)
sentimento_ano(2000)
sentimento_ano(2010)
sentimento_ano(2020)


# Gera uma lista agregada do sentimento por década
summ <- add_sentiments %>%
  group_by(playlist_name) %>%
```

```
  summarise(mean_pol = mean(polarity))

# Plota o resultado da média de sentimento das músicas por década
summ %>%
  arrange(desc(mean_pol)) %>%
  ggplot(aes(reorder(playlist_name, mean_pol), mean_pol)) +
  geom_linerange(aes(ymin = min(mean_pol), ymax = mean_pol,
                     x = reorder(playlist_name, mean_pol)),
                 position = position_dodge(width = 0.2), size = 1,
                 colour = 'darksalmon') +
  geom_point(colour = 'dodgerblue4', size = 3, alpha = 0.9) +
  labs(x = 'Músicas', y = 'Sentimento') +
  theme_bw(14)
```

Referências

Wundervald, Bruna and Trecenti, Julio, Music Data Analysis in R. Acesso: https://github.com/brunaw/SER2019, 2019. }