Espotifai

Automatic Playlist Recommender

Lucas Emanuel Resck Domingues and Lucas Machado Moschen

School of Applied Mathematics
Getulio Vargas Foundation

August 7, 2020

GitHub Repository

1 Project statement

Question: Which song should we recommend based on a playlist and user or context information?

We can define playlist as a sequence of tracks (audio recordings). In this project we aim to study the playlist generation problem, that is, given a pool of tracks, a background knowledge about the user, and some metadata of songs and playlists, the goal is to create a sequence of tracks that satisfies some target as best as possible. The notebooks, scripts and work can be found in our repository.

2 The Datasets

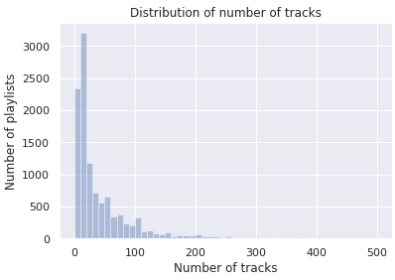
In order to get user information, we generated a list of usernames in a networked way. We visited the Last.fm webpage of several artists and considered three random users in the top listenings from three different coutries: Brazil, USA and United Kingdom. Using only these usernames, through the command `generate_lastfm_users.py`, we get additional Last.fm usernames using the `user.getFriends` method from the API. With some loops, we can get the network (or part of it). It's possible to have some bias, unknown yet.

Both databases were built using the Application Programming Interface (API) from the website, in special Spotify API and Last.fm API. Both are great players in the music business and that was the reason for us to use.

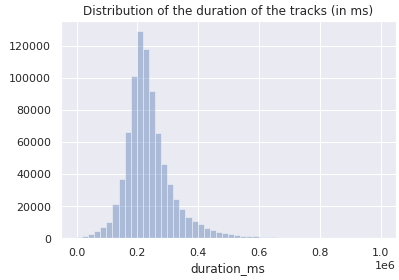
2.1 Spotify Database

We used Spotipy Python package to connect with Spotify's Web API. It allows us to gather information about public playlists, tracks and artists. Using the dataset of existing Last.fm users, we sampled a number of users and tested them to see if they have an account on Spotify. This way, we gathered information about these users' playlists. Everything that Spotify's API returns is a `.json` file. Playlists data were treated and saved as an `.pkl` file. Each track in the playlists dataset contains an ID, which identifies the tracks on Spotify and allows us to search for the tracks information. So we did it, treated the dataset and saved too. Again, we searched for the artists of the tracks information and what is called *audio features* of the tracks, that is, a pool of technical metrics like *danceability* (how much a track is danceable) and *instrumentalness* (how much we are convinced a track is instrumental). At the end, we have:

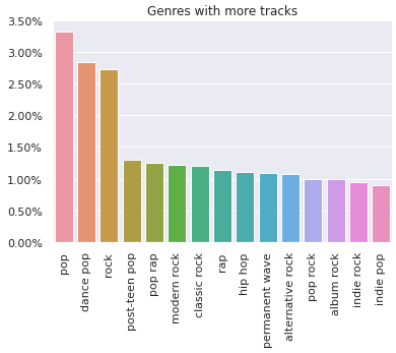
1. **Playlists**: Scrapped from users. 11617 playlists, only 6% duplicated.
    - Basic information about a playlist, like its ID, who is the owner, its name.
2. **Tracks**: Scrapped from playlists. 849397 tracks, 5% duplicated, 4.6% missing.
    - Playlist features, like its ID, when the track was added to it, who added it;
    - Track features, like duration (ms), popularity and name;



(a) Distribution of number of tracks among playlists. We can see that most of playlists have few tracks (100 or less), which is reasonable. We also found some outliers, like a playlist that has more than 13000 songs!



(b) Distribution of the duration of a track (in ms). We see that the mode is around 3min20s.



(c) Tracks have artists, and artists have genres. This is the result of our analysis of the genres of the tracks. We see what we would expect: pop, rock, rap and hip hop in the list. The percentages seem small, but there are many genres to consider (in our sample, 2297 genres). This analysis was generated on a sample of the data.

Figure 1: Visualizations of the data.

- Album features, like name and artists (name and ID);
- Artists features, like their names and IDS.

3. **Audio features**: Scrapped from tracks. No duplicated or missing values.
   - A lot of technical features from the audio, like *danceability*, *energy* and *loudness*. Specially important for the models.
4. **Artists**: Scrapped from tracks. Only one artist duplicated.
   - Very basic information, like the artist ID and its genres. The genres are the most important, because tracks and playlists don't have genres, only the artists.

2.2 Last.fm Database

We used pyLast package for Python to help with the connection. It organizes all as an object with several functions. However, in order to retrieve all the information, it takes a long time to build all datasets: **Users info**, **Tracks info**, **Artists info** and **Tags info**, because we retrive a lot of information in different links from the same API. That's why in the first analysis we only considered a subset. We got the users randomly from the whole dataset of users. We saved the information in a `json` format. The variable types were fixed to help the analysis (convertion `string->integer`). The dates were converted to a standart type and after to datetime. Actually, as we used an API, the data wasn't bad. We created ids for tracks, tags and artists and save it in a `.csv` file. This structure allow us to save storage.

The considered datasets were:

1. **Users**: information about if is subscriber, when he/she has registered, playcounts, country, top artists and tracks and loved tracks. The last three are generated by the user and its listening behaviour. (no user had gender, age and number of playlists information and we deregarded.)
   - Users: 1000; users with no information: 1.
2. **Tracks**: information about reaching, playcounts, duration, listeners, similar tracks and top tags.
   - tracks: 9902; tracks with no information: 311.
3. **Artists**: information about number of listeners, plays, when he/she was published, top tracks, top tags and similar ones.

- artists: 1055; artists with no information: 0.

4. **Tags**: information about registration, taggings, reached people, top tracks and top artists.
   - tags: 1843; tags with no information: 4

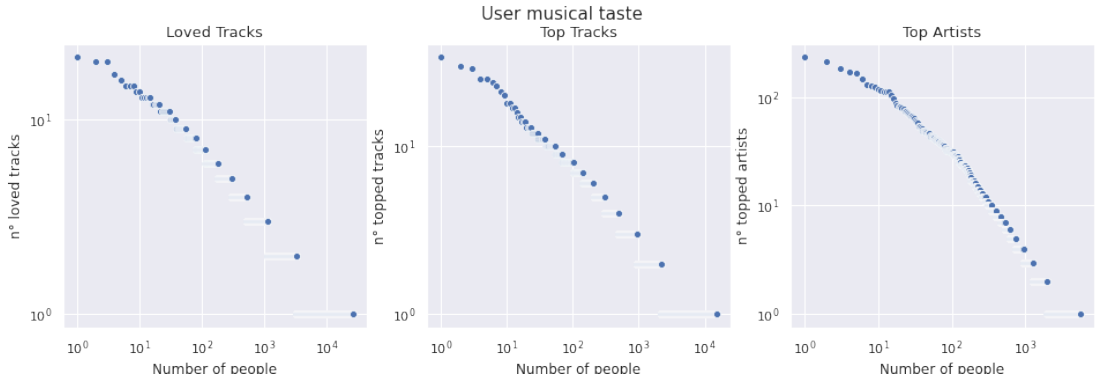


Figure 2: We can see the distribution of number of top/loved tracks and top artists. All have a Pareto distribution (line in log-log scale. This occurs in other graphics.)

3 Gained Insights

1. The distribution of several relations in the data respects a Pareto Distribution. It suggests a structure that can be modelled.
2. The functions `top` and `similar` can be useful, because it has global information from the Last.fm.
3. The dataset from Spotify is the unique with playlist information and is the principal for the next work.
4. Age and country seems to influenciate less than we thought. Lot of missing values.
5. The distributions of the Spotify audio features are very similar to those present on the Spotify's Web API. It suggests our data is quite representative.

4 Baseline Model

It's pretty simple and uses information retrieved from Last.fm. Based on a playlist (from Spotify), we get all the artists who participate in at least one song and attribute a weight (number of times it appears in the playlist). After, we get the 200 top artists of the user, weighted by the number of times the user listened to him/her. With that information, we get the intersection of the two sets and the correspond weight will be a product of the normalized weights. If the set is empty, the method does not work, unhappily. In the intersection set, we take the 10 artists with major weight and for each, we get his/her 10 top tracks and recommend the n tracks with major weight (number of playcounts by all the users in the plataform). It uses few information. However, the information is retrived from a big dataset built by Last.fm and it can be a good baseline to compare. The first big problem is that it favors artists with well-known songs. In our example, a playlist with brazilian popular music had all the recommends with the artist "Legião Urbana", despite the playlist has a variaty of artists from Brazil.