Matteo Fabbri

Report

The most important aspects of data pre-processing are; memory management while mapping multiple large .csv files, working only with the needed data, speed up the .csv file lookup process as much as possible. Since I want to do everything remotely on Colab, RAM allocation has priority over storage memory. The term "filter" is used to describe the process of selecting only some rows -the ones related to songs for which there has been at least 1 listening in 2022- out of the provided .csv files. I mainly took 2 approaches in order to obtain a matric of User IDs <-> Artists IDs mappings;

Approach 1 **Data pre-processing-2.ipynb**	**Approach 2** **Data pre-processing-3.ipynb**
Create a series of 2 columns .csv files, where the first column is always User_ID. The first .csv file maps User_id <-> MessyBrainzID for the whole 2022 year, and the last .csv file maps User_id <-> Artist_name. To speed up the .csv file look up for each mapping, I used sets (representing unique occurrences of items). ● User_id <-> MessyBrainzID (entire 2022) .csv file ● Filter listenbrainz_msid_mapping.csv (MessyBrainzID <-> MusicBrainzID) ● Create a set of unique MusicBrainz recordings IDs ● Create a dictionary of MessyBrainzID <-> MusicBrainzID mappings. This will make the creation of the user_id <-> MusicBrainzID mappings .csv file much faster ● Create a user_id <-> MusicBrainzID mapping .csv file ● Filter canonical_musicbrainz_data.csv (MusicBrainzID <-> ArtistID) in a new .csv file. Create a dictionary of MusicBrainzIDs <-> ArtistIDs mappings. This will make the creation of user_id <-> Artist_IDs mappings faster. ● Create a user_id <-> ArtistID mapping .csv file. Create a set of unique ArtistIDs ● Create a dictionary out of musicbrainz_artist_mbid_name.csv, taking only the rows of interest ● Create a user_id <-> ArtistName mapping .csv file	This approach starts from the file msid_mbid_filtered_mapping_file.csv created in Data pre-processing-2.ipynb and then maps .csv table files only with dictionaries, with no additional .csv files. The takeaway of this is that; ● dictionaries' list of keys can be used the same way I used sets in the other approach, and I did not really needed sets ● non-nested dictionaries can abstract medium-size tables mappings ● dictionaries have a constant time complexity of O(1) I think this approach is cleaner than the other. I also tried to compute, for each artist, the listenings count for each user, using nested dictionaries (a key for each artist name with value a dictionary with, for each user that has listened at least once to that artist, user id as key and listenings count as value) like; {artist_name: {user_id: listenings_count, user_id: listenings_count, …}, artist_name: {...}, …} This quickly filled the available RAM, so I calculated the listenings count with Pandas.

I tried as much as possible to prevent data-loss in case of full-RAM crashes, by, for each cell, loading (from serialized file), consuming, dumping (to serialized file) and de-allocating data-structures. For the collaborative filtering part, before converting the matrix into a coordinate format sparse matrix, I first tried to re-format it (unique user IDs as rows, unique artist names as columns, listening counts as mappings) in several ways (Pandas.DataFrame.pivot() uses too much RAM, formatting a DataFrame with a dictionary also uses all available RAM.. the only approach that worked was creating a 27 GB .csv file) before finding out that it was probably not necessary since the Implicit library can do it for me, after changing some code. I would probably have managed to do it by directly creating it in the coordinate format (dictionary with tuple of user_id and artist name as key, and listenings count as value). I instead used pandas.DataFrame.groupBy() to compute the count of listenings, and then, after realizing that Implicit has some issues with a matrix in the format user_id, artist_name, count, I created a matrix with format user_id, artist_mbid, artist_name, count and use artist_mbid for any back-end calculation and artist_name only for the user "interface".

For 'Tame Impala', my system listed as top similar artists; MGMT, Gorillaz, Radiohead, The Strokes. My Spotify, for the same artist, listed; Mild High Club, Pond, Crumb, Unknown Mortal Orchestra, MGMT. This is not due to some extra user-tailored filtering/ordering done by Spotify, because I actually listen more to Gorillaz, Radiohead, The Strokes than the ones listed by Spotify, and also my friends visualize the same similar artists for Tame Impala.

For 'The Chemical Brothers', my system lists; Fatboy Slim, The Prodigy, Massive Attack, while Spotify lists different artists for the top 10 matches, and then lists Fatboy Slim and The Prodigy. This might be also due to the restricted amount of artists available in MusicBrainz comparing to Spotify.