Lab Report - Mapping genre taxonomies

Tiange Zhu
Feb/2019

Step 1: Find the intersection of the four datasets, get their recording musicbrainz id.

I used pandas to read the files, stored four of them separately into four dataframes. In order to find the intersection, I first selected the “recordingmbid” column(musicbrainz recording ID) of the first dataset(in my program it was tagtraum) and transferred the IDs into a list. Then I used this list to filter the second dataset I read, only kept the rows that their ID appeared in last dataset. I then updated the list of intersection, using the new list to filter the third dataset. Repeat once more, the final intersection ID list was born.
After using the final intersection ID list to update previous dataframes, all four datasets only contain the intersection part. During the test, I noticed that there are some empty columns in the input files, I filtered them out with “isna()” and “drop()” function.
The size of four dataset and their intersection is shown below:

```
The number of tracks in tagtraum dataset:
486740
The number of tracks in lastfm dataset:
566710
The number of tracks in discogs dataset:
904944
The number of tracks in allmusic dataset:
1353213
The number of tracks in the final intersection is:
247716
```

Step 2: Adding sourcename as prefix to all the genres, make them appear as "sourcename---genre---subgenre".

While reading four files separately, I added source names as prefix to all the columns that contain genre information, using “.applymap()”function. There’s a introduction about the difference between apply(), map() and applymap(), which I found useful:
https://www.geeksforgeeks.org/difference-between-map-applymap-and-apply-methods-in-pandas/

Since I’m using applymap(), it applies to all the elements in the dataframe, including two columns that do not need a source prefix, which is “recordingmbid”, and “releasegroupmbid”. I find latter one useless for this task, so I dropped the column

out, and used apply again to replace the unwanted prefix in“recordingmbid” columns.

```
tagtraum = tagtraum.applymap(lambda x: 'tagtraum---{}'.format(x))
tagtraum['recordingmbid'] = tagtraum['recordingmbid'].apply(lambda x: x.replace('tagtraum---','').format(x))
```

Later when proceeding to the step 3, I realized there’s an easier way to do it, which is replacing the index by “recordingmbid” column. The ID became index, thus I could simply add prefix to everything, without deleting unwanted ones later, which would improve efficiency of the program.

Step 3: Remove the sub-genres that occur for less than 200 recordings.

I first combined four datasets into one, as shown below:

recordingmbid	genre1	genre2	genre3	genre4	genre5	genre6	genre7	genre8	genre9	genre10	...	genre1	genre2	genr
00000baf-9215-483a-8900-93756eaf1cfc	allmusic---pop/rock	allmusic---pop/rock---heavy metal	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	...	lastfm---metal	lastfm---metal---folkmetal	lastfm-n
0000167c-95f5-48eb-b15f-04790e09a765	allmusic---pop/rock	allmusic---pop/rock---club/dance	allmusic---pop/rock---dance	allmusic---pop/rock---dance-pop	allmusic---pop/rock---pop	allmusic---pop/rock---pop/rock	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	...	lastfm---metal	lastfm---metal---classicbritishheavymetal	lastfm-p
00002fc9-7283-44dd-bf6f-94c9492d0998	allmusic---rap	allmusic---rap---alternative rap	allmusic---rap---hip-hop/urban	allmusic---rap---midwest rap	allmusic---rap---underground rap	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	allmusic---nan	...	lastfm---hiphop	lastfm---nan	lastfm-n

One way of getting the list of unrepeated genre names is using the function “.unique()”. But later I found a more efficient way: I used “pd.value_counts()” function to count the occurrence of each genre, it would automatically get the names of every genre without repetition. According the result of counting, I filtered out ones that appeared less than 200 times. After that, I filtered out the unwanted ones(ones that doesn’t contain a subgenre, which can be detected by judging if there’s two “---”), also got rid of the empty values.
Counting the number of unfiltered and filtered genres:

```
The number of unfiltered genre is:
1706
The number of filtered genre is:
1060
```

Step 4: Compute occurrence matrix.

In order to compute a track-subgenre occurrence matrix, I transferred dataframe to dictionary. For each track I found all the genre tags, only kept the ones that’s in filtered genre list. Since the indices must be integer, the matrix has to be saved as “matrix[genre][song]” instead of “matrix[song][genre]”.

I then transferred the matrix from dictionary to Dataframe, and replaced indexes with their “recordingmbid”.

recordingmbid	allmusic---pop/rock---alternative/indie rock	tagtraum---rock/pop---alternative	tagtraum---rock/pop---pop	allmusic---pop/rock---alternative pop/rock	allmusic---pop/rock---pop/rock	allmusic---pop/rock---heavy metal	allmusic---pop/rock---hard rock	allmusic---pop/rock---contemporary pop/rock	tagtraum---rock/pop---metal	discogs---rock---alternative rock	...	lastfm---pop---westcoastswing
00000baf-9215-483a-8900-93756eaf1cfc	0	0	0	0	0	1	0	0	1	0	...	0
0000167c-95f5-48eb-b15f-04790e09a765	0	0	0	0	1	0	0	0	0	0	...	0

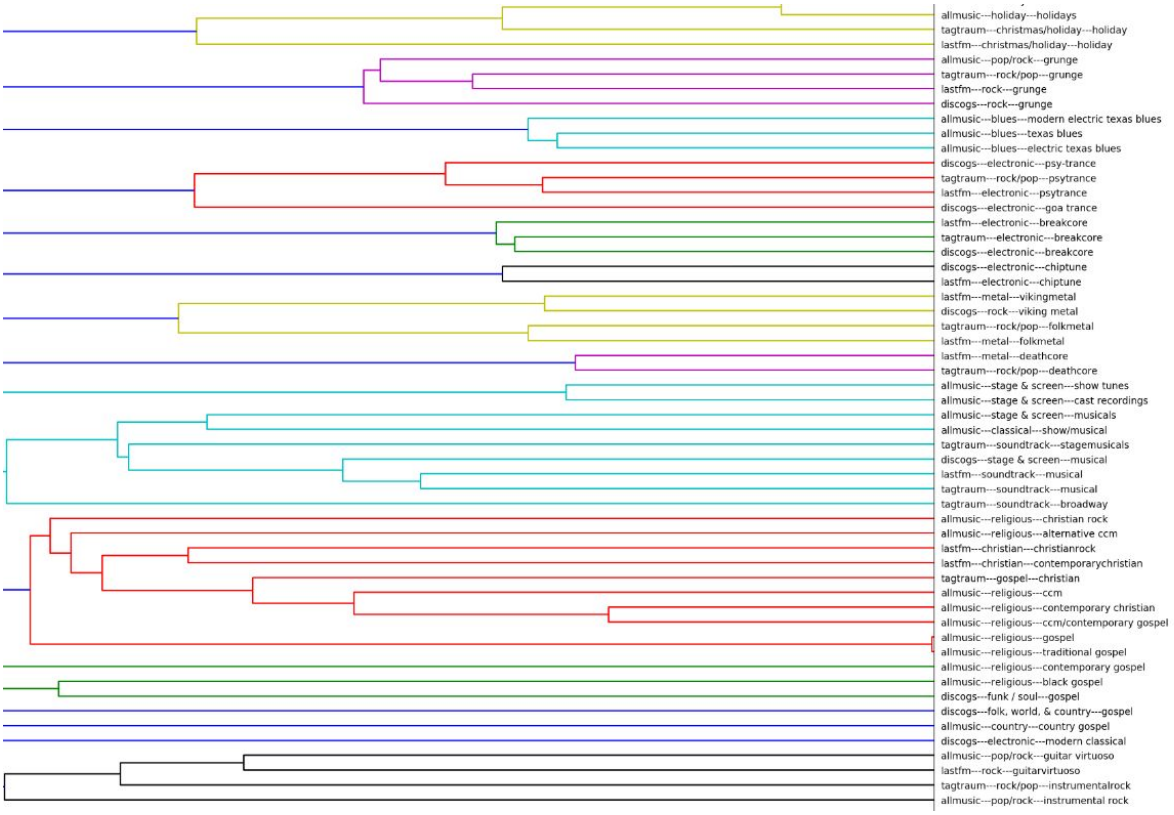
Step 5: Compute the distance matrix (cosine distance) for subgenres across all sources.

In this step, I used a function called cosine_similarity, from “sklearn.metrics.pairwise” library.(source:https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html). The input for the function was transposed occurrence matrix.

Step 6: Perform a clustering.

In this step, I used linkage and dendrogram function from “scipy.cluster.hierarchy” as the introduction suggested. I changed the plot size and DPI to show the plot. I attached the whole plot in ZIP file.

Dendrogram analysis:



Below is a part of the output. As we can see, the result of using cosine distance to show the “distance” of the genres are very similar to what we usually expect. “allmusic-religious-contemporary christian” is very close to “allmusic-religious-contemporary gospel” since gospel music is always Christian. Same tags from different sources usually have a pretty close distance. I found the farest distance in the dendrogram seems to be from holiday music(Christmas songs) to reggaeton.

According to my test, the whole program takes 7 mins to run. I find distance matrix computation and clustering take longest time.