Clustering Last.fm users based on their listening events

A project within the data mining seminar at the University of Applied Sciences Salzburg[*]

Stoecklmair Jan Peer, BSc.
University of Applied Sciences Salzburg
Puch bei Hallein, Austria
fhs39850@fh-salzburg.ac.at

1 LAST.FM DATA ACQUISITION

All the data which were used within this project comes explicit from Last.fm. Last.fm comes with a handy API where you can get the data from the users listening events, top artists or tags. The API also provides an endpoint for the current charts. The charts endpoint are separated into top artists, top tracks and top tags. E.g., the charts are necessary to cluster the artists by their mainstreaminess. We used the first the first 40000 users of the LFM-1b dataset and fetched their data.

1.1 User data

Recent tracks, top artists and top tracks got fetched per each user. As the API has a limit of 500 entries per fetch, where the rest was paginated, we fetched per user 500 entries each, so 1 page each. So, in total, there are 1.500 entries per user, for recent tracks, top tracks and top artists. Couple of users do not have enough entries and could not reach the maximum of 1.500 entries.

1.2 Charts data

For the charts we used the endpoints for the top tags, top tracks and top artists. Again as mentioned above, the API just allows a maximum of 500 entries of each request, where the rest will be accessible in another ’page’. To get enough data, and to play around with them after, we first ensured to fetch 500 entries of each endpoint. So in total we got 1.500 entries from the charts.

1.3 Artist meta data

After all artists are fetched, users top artists and top artists of the charts, we could collect all artists in one file after the first data preparation step, which is mentioned in the next section, and iterate over every artist to get more meta information. This information contains playcounts, listeners and tags. E.g., tags are also quite useful to calculate the diversity of each user, more details about that in a later section.

2 DATA PREPARATION

The data preparation step converts the fetched raw data into working files. The main idea to put everything in such data was, that it is possible to work with references. E.g. there is one artists file, where all appearances of artists are saved, so in the tracks file could the reference be used to point to the artist. This approach would guarantee that the right artists got be chosen in the tracks file, as it also easier to pick the right artist by the reference number.

2.1 Artist preparation

The very first thing was to collect all information about the artists, because every next step is depending on the output of the artist preparation. Collecting all artists is very hard, because the artists are not just in the fetched data of users top artists or the top artists of the charts. Relevant artists could also hide in the fetched tracks itself. E.g. we could not assume that all the artists were fetched from the recent tracks of the user, as it could be that the user heard a song couple of minutes before with a brand new artist. So this specific artist would possibly not appear in the users top artists. So at the end every artist were collected with the current information: "Name" and "MBID" (MusicBrainz ID). "MusicBrainz is an open music encyclopedia that collects music metadata and makes it available to the public."[2]. The final result was an ‘artist.txt‘ file, with all found artists with the artists name and its MBID, if it exists. At the very same time the files for the users top artists and the top artists of the charts were generated.

2.2 Tracks preparation

The next step is to generate the tracks file this is a pretty easy step, as the only things to consider are the fetched top tracks, recent tracks and the users top tracks. For the tracks file just the name, MBID and the artist reference is stored. Also an own file for users top tracks, users recent tracks and top tracks were generated.

2.3 Tags preparation

The last thing, which is missing, are prepare the tags. Tags are fetched in the top tags of the charts and also stored in every artists, as every artist got specific user generated tags. Those tags are stored in one big file ‘tags.txt‘. To avoid variations of single tags, such as ‘Hip-Hop‘, ‘hip hop‘ or even ‘hipHop‘, we lowercased every tag and removed all special characters. t

[*]Supervised by Markus Schedl

3 DATA PROCESSING

After the preparation step, it is possible to work with the data easily and the generated files look like following:

```
/data/
 |__user_recent_tracks
 |  |__{USERNAME}.txt
 |  |__...
 |__user_top_tracks
 |  |__{USERNAME}.txt
 |  |__...
 |__user_top_artists
 |  |__{USERNAME}.txt
 |  |__...
 |__top_artists.txt
 |__top_tags.txt
 |__top_tracks.txt
 |__artists_with_tags.txt
 |__tracks.txt
 |__tags.txt
 |__users.txt
 |__artists.txt
```

In the data processing step it is time to generate datasets to find patterns within the given data. Those datasets are combinations of different approaches, such as, the playcounts of the charts top tracks from the user. This combination could tell, if a user is more mainstream than another. We created six datasets, where every single dataset is normalized:

Charts Top Artists - User Top Artists (Dataset #1). In this dataset we counted per user, how many top artists the user has in common with the top artists in the charts. With this dataset we want to show how mainstream each user is. The result is stored in *data_processed/top_tags-user_top_artists.npz.*

Charts Top Tags - User Top Artists (Dataset #2). This dataset is similar to the previous one, but here we take the tags of each artist. With this dataset, we want to show how different it is to take the tags instead of the artists itself. The result is stored in *data_processedtop_tags-user_top_artists.npz.*

Charts Top Tags - User Recent Tracks (Dataset #3). Every track has their own artist, which has tags. These tags are counted by the top tags of the charts. So each dimension is one of the charts top tag, where over the dimensions entry is the count of the users counted tags. This dataset will be used to check how open the user is based on the tags of their recent listened artists. The result is stored in *data_processed/top_tags-user_recent_tracks_tags.npz.*

Charts Top Tracks - User Recent Tracks (Dataset #4). Here we introduced a dataset which should display how open each user is for new music. One dimension represents one of the top tracks of the charts. In this dataset we counted how many recent listening events a user have for each track in the top tracks. To reduce the dimensions we took just users where they got at least 200 top tracks. The result is stored in *data_processed/top_tracks-user_recent_tracks.npz.*

User Top Tracks - User Recent Tracks (Dataset #4). In this dataset we wanted to see how open every user is for new music. This got measured by the user's top tracks and the count of how many of these tracks the user listened in the last time. The result is stored in *data_processed/user_top_tracks-user_recent_tracks.npz.*

New Songs / Old Songs (Dataset #5). Here we counted how many new songs the user listened to based on its top tracks. Every track which is not listed in the user's top tracks list is "new", every other is already known, therefore "old". The result is stored in *data_processed/user_top_tracks-user_recent_tracks-short.npz.*

New Songs / Old Songs / Different Artists / Different Tracks (Dataset #6). This dataset is an extension to the previous one. It adds different listened artists, and different tracks of the recent listening events. The result is stored in *data_processed/all_mixed.npz.*

As you could see, every file is stored in a *npz* format. This format is a sparsed matrix format, which is used to save a matrix without any zero values. This is very helpful, as the normal matrix format of **data_processed/top_tags-user_top_artists.txt** had an originally size of around 80 megabytes, whereover the sparsed matrix with the *npz* format has just around 42 megabytes, which saved around 50% of space on the disk. This not just saves space on the disk, it also increases write and read speed of the matrix itself [3].

4 CLUSTERING APPROACH

The previous generated datasets are now ready for clustering, plotting and analyzing. As we generated a high dimensional matrix in each dataset it is not really possible to visualize such matrices and we have to reduce its dimensions. Dimensionality reduction is a simple method to get two or three dimensions out of a high dimensional matrix. A two or three dimensional matrix is easier to visualize than a 4 or even 1000 dimensional matrix. Dimensionality reduction is a part of the unsupervised learning [1]. For the dimensionality reduction within the single datasets we experimented with different algorithms of the Scikit-learn (sklearn) [7] library for Python, which are mentioned later. Algorithms such as t-distributed Stochastic Neighbor Embedding (t-SNE) are available within the sklearn library. Also the sompy library is in use. The sompy library is a Self Organizing Map (SOM) library for Python [6]

5 RESULTS

The following results will be different plots of the different datasets.

5.1 Dataset #1

The first dataset we analyzed was Dataset #1. We originally assumed that comparing the user's top artists with the top artists of the charts we can cluster all user's how mainstream they are.

In Figure 1 there is one big cluster on the bottom left, and there are also some single data points all over the rest of the plot. First we thought this big cluster says that there are a lot of users which are listening to the top artists and are therefore mainstream. To proof our assumption we picked in total four users from the plot, by adding labels over the data points on the plot and picking the users manually. It seems that our assumption was wrong, the users in the cluster on the bottom left, have a lot of 0 values, which means they did not listen to the artists of the charts. So the bottom left

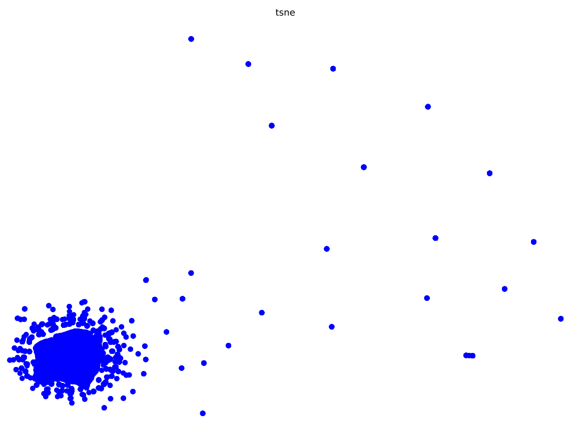

Figure 1: Dataset #1 with dimensionality reduction by t-SNE

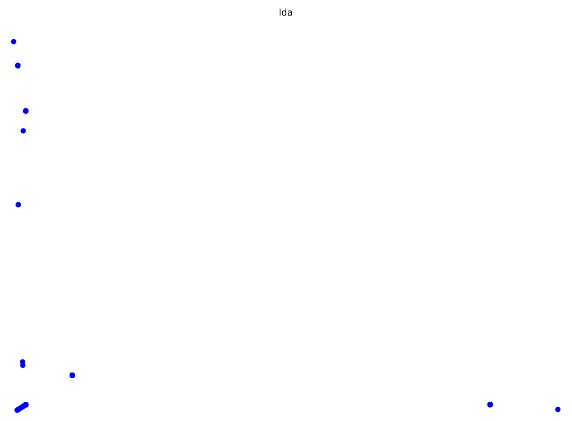

Figure 2: Dataset #1 modeled by LDA

cluster are users which are not mainstream, where over couple of users, which are spread all over the plot, are more mainstream and listened to couple of artists from the charts. To see if other algorithms would cluster it similar we tried the Latent Dirichlet Allocation (LDA), which is a Bayes algorithm [5]. Figure 2 has a lot of users on the bottom left and couple of users on top and right, which is not really good to see on this plot. According to Figure 1 and the previous analyzes discussed above, Figure 2 shows the same result, the users we picked are in the same clusters.

5.2 Dataset #2

The dataset #2 was really bad with the same settings at the t-SNE. In Figure 3, to get better clusters, we set the perplexity to 100, which was set to 10 on the other plot. There are some obvious clusters, especially on bottom left and middle top. In this plot it seems that the users are really spread all over the plot, where over in Figure 1 it was not that spread.

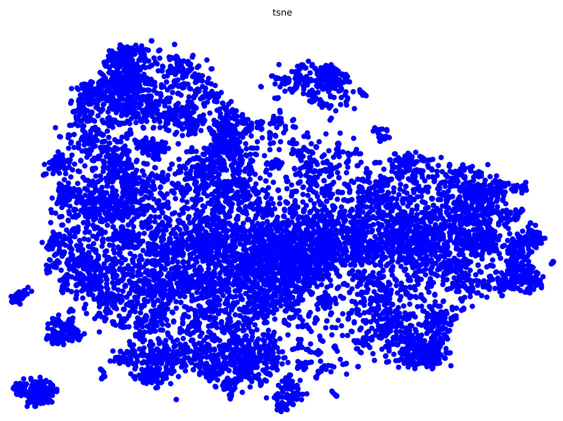

Figure 3: Dataset #2 with dimensionality reduction by t-SNE

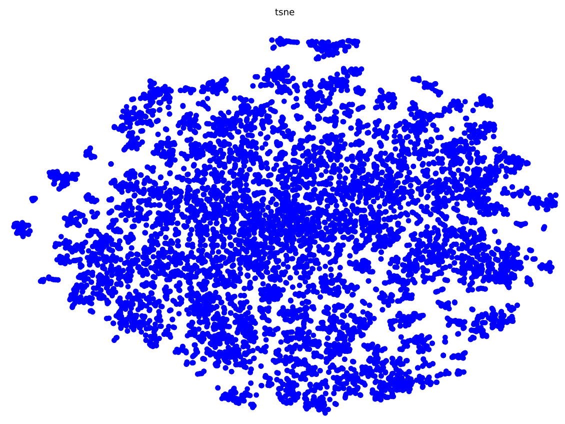

Figure 4: Dataset #3 with dimensionality reduction by t-SNE

5.3 Dataset #3

Also dataset #3 does shows some clusters in it, but the users are pretty spread all over the plots. This plot is in our opinion not well suited to evaluate. This dataset was originally meant to cluster users how mainstream they are, but as the clusters are spread that much, we would say this dataset was not well defined. It will be probably better, if not all tags of the tracks get used, but just the very first tag.

5.4 Dataset #4

Dataset #4 has the best looking plots, they are well clustered. To proof that these cluster really have some meaning, we are analyzing them in detail as well.

In Figure 5 we used the Spectral Embedding (SPE). SPE is a non-linear dimensionality reduction algorithm: "Forms an affinity matrix given by the specified function and applies spectral decomposition to the corresponding graph laplacian. The resulting transformation is given by the value of the eigenvectors for each data point." [4].

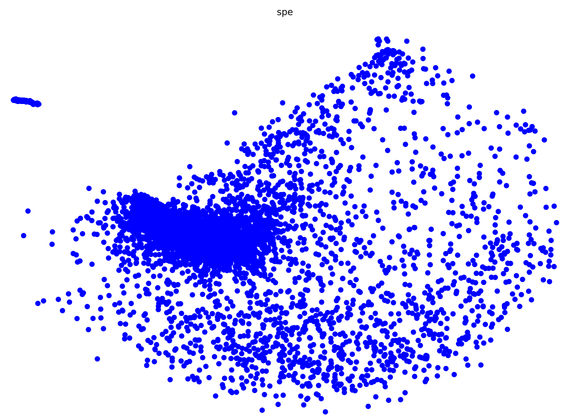

Figure 5: Dataset #4 with dimensionality reduction by Spectral Embedding

In this plot it is good to see, that SPE is a non-linear dimensionality reduction, as the data points itself are following a kind of curved line from the left to the right of the plot. In this plot there are four obvious clusters. Where one is totally separated from the other three. Those three which are combined in the bigger cluster, can be separated into the one in the middle left, the one on the bottom and the top. Now we picked four users within the plot. We picked one user of the middle left cluster (user 9582), one of the cluster which is totally separated (user 4484), another user in the cluster in the bottom (2532) and a user on the total top of the plot (user 1378). By analyzing them manually it seems that the cluster in the middle left combine all users which are not listening to the top tracks at all. All those users are listening to new songs. But the small cluster on the left, which is totally separated, includes users, which are the most open users, as they listen to very little tracks in their open tracks. It seems that user which are clustered on the top and bottom are users which are not really open for new songs, they recently listened to their top tracks. Users in the top cluster, are a bit less open than the users in the bottom cluster. So the dataset #4 worked out pretty well, also with the SPE algorithm.

As the SPE showed good clusters we assumed to get good clusters with t-SNE as well. In the t-SNE there are more significant clusters. We wanted to know in which clusters the four chosen users are now. The plot has an inner and outer cluster. The chosen users are all in the inner cluster. Within the inner cluster there is a kind of paw on the bottom. There are all users which did not recently listened to their top tracks. User 9582, which is most open user within the chosen users, is located in the biggest cluster of the paw (the 4 round clusters). Second most open user is in the smallest of those 4 clusters, the one right next to the biggest cluster on the right. The other users, which are not open for new music and mostly listening to already known music are located on top of the paw cluster, the stripe shape. The stripe shaped cluster can be separated into two clusters, a very big one on the left, and a small cluster on the right. The small cluster on the right, might be the same cluster as the small cluster, which was really far away from the main cluster shown in Figure 5. In this small cluster is also the user, which is totally not open for new users (user 1378). The bigger stripe cluster includes users which are not really open for new music, but still try to listen to new music sometimes. This includes also the third most open user of our four chosen users, user 2532.

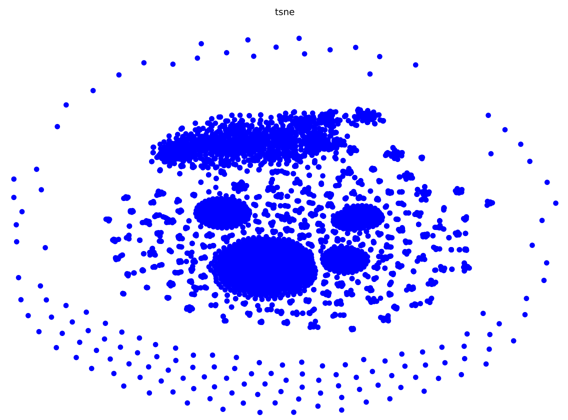

Figure 6: Dataset #4 with dimensionality reduction by t-SNE

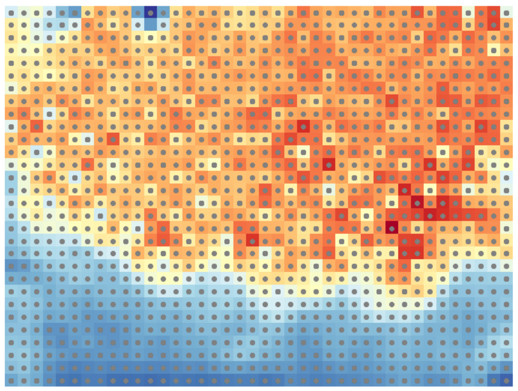

Figure 7: Dataset #4 clustered with SOM

As the most users seams to be not open for new music, we want to see that result also in a SOM. We used, as mentioned above, sompy. Figure 7 shows the plotted result. Based on the other plots, you can see that all the red parts are users which are not open for new music.

5.5 Dataset #5 and #6

We skipped analyzing dataset #5 and #6, as the results were not good enough to show them, but they also did not have many features. The result might improve if the features were better chosen.

6 CONCLUSION

We found out that dataset #4 gave the best results, so it is good to cluster people who are open for new music, based on their recent listened tracks and their top tracks. It might be interesting for the future work, to specialize dataset #4 to get even better results. Other datasets, especially #5 and #6, were not really well suited to cluster users. Also the dimensionality reduction with t-SNE was the best from all tried ones. MDS, which was not shown in this report, but in the appendix, took a lot of time to just plot the result, where the result was also not satisfying. Clusters were not clear visible, which was a reason why it was not shown in this report.

REFERENCES

[1] [n. d.]. 2.2. Manifold learning âĂŤ scikit-learn 0.19.1 documentation. ([n. d.]). http://scikit-learn.org/stable/modules/manifold.html

[2] [n. d.]. MusicBrainz - The Open Music Encyclopedia. ([n. d.]). https://musicbrainz.org/

[3] [n. d.]. scipy.sparse.save_npz âĂŤ SciPy v1.0.0 Reference Guide. ([n. d.]). https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.save_npz.html

[4] [n. d.]. sklearn.manifold.SpectralEmbedding âĂŤ scikit-learn 0.19.1 documentation. ([n. d.]). http://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html

[5] Matthew D. Hoffman, David M. Blei, and Francis Bach. 2010. Online Learning for Latent Dirichlet Allocation. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1 (NIPS'10)*. Curran Associates Inc., USA, 856–864. http://dl.acm.org/citation.cfm?id=2997189.2997285

[6] Vahid Moosavi. 2018. SOMPY: A Python Library for Self Organizing Map (SOM). (Jan. 2018). https://github.com/sevamoo/SOMPY original-date: 2014-08-24T12:06:09Z.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.