Spool Final Report
CS378 Data Mining SP18
Tyler Angert, Angela He, Rahul Nair
May 7, 2018

Motivation and contributions:

Shared music listening is a disconnected process right now as most music services (Spotify, Apple Music) focus on discovering, streaming, and curating music for individuals, not communities. Sharing music with friends who have similar tastes as you is a great way to discover something new, but music services aren't built for that right now, leaving most people to share songs individually through messaging or face-to-face interaction.

Using a dataset of Last.fm users with their top played artists, we set out to create communities of users who have similar listening interests. Using clustering, we were able to determine clusters of users with similar music tastes. These clusters were based off of top artists listened to for each user. So instead of algorithms curating what songs a user should listen to, communities of like-minded listeners can suggest music with each other.

The data mining behind Spool is the backbone for three music possibilities: first is location based streaming, where a user can explore what is being listened to in cities around the world. Second is the curation of playlists based on common artists between a group of users, such as creating automatic playlists for groups of friends during a party or road trip. Third is music community discovery, which allows people to subscribe to music from other people, instead of automatically generated genre playlists.

These clusters were then used to create a data visualization that allows one to interactively explore the clusters and see what other similar artists are commonly listened to in each cluster. Different k-values are depicted as well so it is possible to visually understand the impact different k-values have on the clusters.

Contributions:

- Tyler: cluster analysis, implementation, visualization
- Angela: user research, problem modeling, cluster analysis, visualization
- Rahul: research, problem modeling, data preprocessing

Related work/methods:

Because we are using a relatively common method of finding similar items (users), we referenced cluster analysis from Netflix, Amazon, and other companies that perform customer and user segmentation.

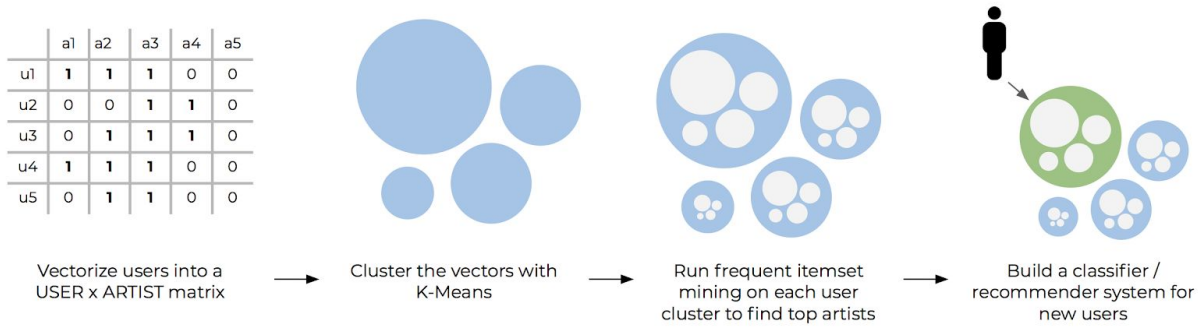
In relation to other work that involved finding similarities in listening habits of music users, the Spotify Nelson Project and the Spotify Recommendations API both are used to filter music; Nelson sets different filters for songs based on characteristics such as tempo and popularity, but also more ambiguous characteristics such as “acoustic-ness” and “danceability”. These measures are used to build a more personalized playlist that are more filtered beyond strict genres.

The Last.fm dataset has been used to create other projects before by other users, such as creating a recommendation system that builds personalized recommendations from listening history, user age, gender, and location (https://github.com/nachiketmparanjape/Music-Recommender-last.fm).

Similar datasets to the Last.fm one include the Million Song Dataset, which was provided for a Kaggle challenge to build the best music recommendation system possible.

Approach and methodology:

Data mining pipeline:



Because we had approximately 30,000 unique artists and 20,000 users to cluster, we were dealing with a very high dimensional space but our USER x ARTIST matrix was incredibly sparse. Since every user was only represented with between 5 - 15 top artists, each vector had approximately 29,990 0’s.

Then, instead of using a traditional boolean matrix (with 1's denoting an artist is listening to, and 0 not), we took the relative play count of each artist per user.

So, instead of using raw play count values (500 plays, 400 plays, etc.) we normalized each user's top user playcounts by dividing each play count value by the total number of artist plays they had. This way, even people who listen to similar artists are clustered by (essentially) how much they like listening to those artists.

This can be defined as a function *Prop* for any given Artist A_i and any given User U_c.

$$Prop(A_i) = \frac{PlayCount_{A_i}}{PlayCount_{Total\ U_c}}$$

Ex: a user whose top 3 artist play counts are:

[500,300,400]

would be normalized to:

[500 / 1200, 300/1200, 400/1200] = [0.417, 0.25, 0.33]

In order to create the matrix efficiently, we used a a CSR (Compressed Sparse Row matrix).

This stores a sparse matrix (mostly 0's) in the form of three 1D arrays:

Array 1: row indices (denotes *user*)
Array 2: column indices (denotes *artist*)
Array 3: data (playcount proportion per *artist*)

After enumerating all of the unique artists and assigning them to unique integer values, we could then create our vectors. So, if we were choosing the top 3 artists for each user (in this case, user 0, 1, and 2), our arrays could look something like

[0, 0, 0, 1, 1, 1, 2, 2, 2 … $user_N$, $user_N$, $user_N$]
[5, 3, 2, 1, 2, 7, 9, 8, 4 … $artist1_N$, $artist2_N$, $artist3_N$]
[0.7, 0.2, 0.1, 0.5, 0.4, 0.1, 0.4, 0.2, 0.2 … $prop1_N$, $prop2_N$, $prop3_N$]

Therefore, when read down all three arrays vertically, each column is essentially a set of "coordinates" for each data point. Using scikit-learn, we were able to convert the sparse storage of this matrix into a dense (fully populated) matrix and finally cluster our

results with K-Means. For speed purposes, we used MiniBatch K-Means which only slightly increased the SSE of our clusters. We didn't get a chance to implement a custom classifier, but the K-Means model from scikit-learn can automatically classify any new tuple using KNN.

Once the data was clustered, we then created an algorithm to convert it into a JSON tree that can be parsed by D3.js, a popular data visualization library.

Evaluation and results:

We first found the the open-source Million Song Dataset, a collection of audio features and metadata for a million contemporary popular music tracks. However, given the need to assess the link between music and its listeners, a further examination of the complementary datasets took us to Last.fm's 360K, which aggregates music data with listener metadata.

Our data came in this format:

The **user profile data** contains rows that look like:

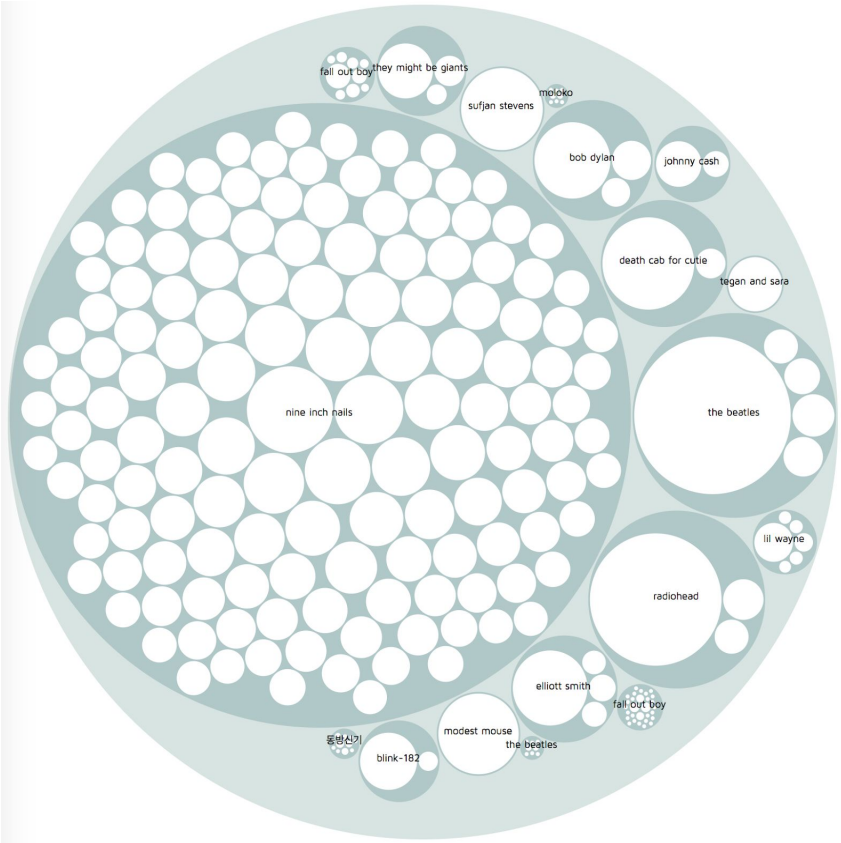
```
[ user_id, gender, age, country, join_date ]
```

The top **user artist data** contains rows that look like:

```
[ user_id, artist_id, artist_name, play_count ]
```

First, in order to obtain relatively "easy" data to cluster, we took a subset of the original dataset; first, by removing incomplete rows (missing gender, country, etc.) and then by taking a random sample of **20,000 US users**. We then placed entire top artist dataset into a **Pandas** dataframe and **Grouped** all of the artists by **userID**.

This allowed us to iterate through all the sampled US users, extract the top artists for each user, and then convert it into our user-artist matrix. Below, find visualizations of our first clustering trials with the Last.fm data, and further visualizations that included improvements to minimum support and the number of top artists searched for. The clusters are labeled by the most popular artist within that user cluster.

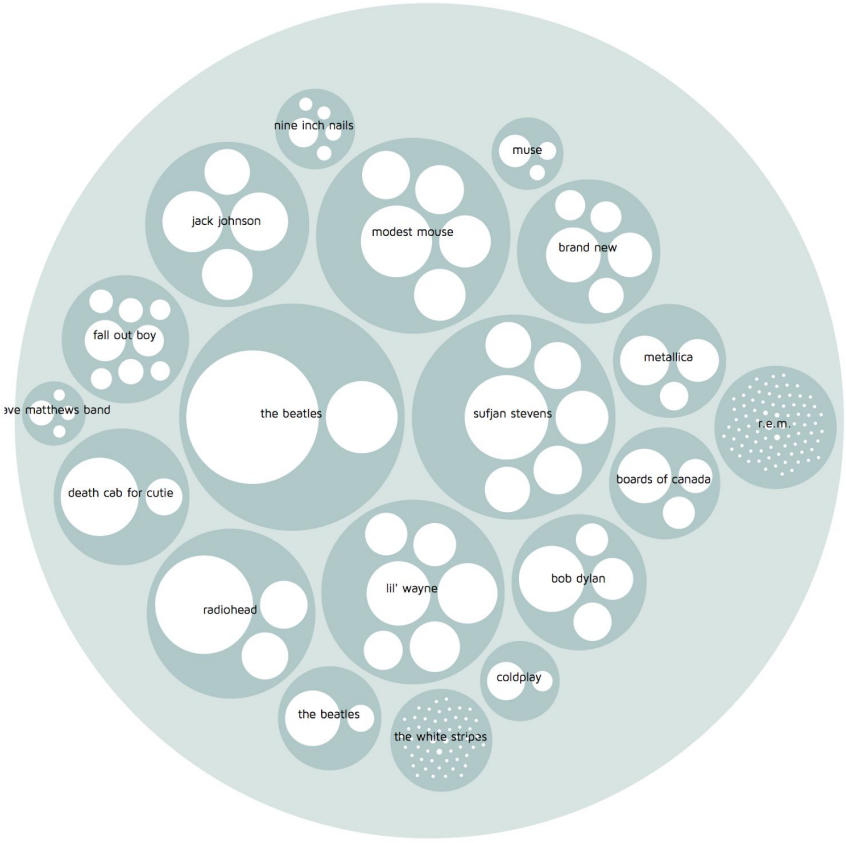


Preliminary cluster results

K = 19 (with one empty cluster)

User top-artist count: **5**

Min support of users for each top artist: 10%



Final cluster results

K = 19 (with one empty cluster)

User top-artist count: **15**

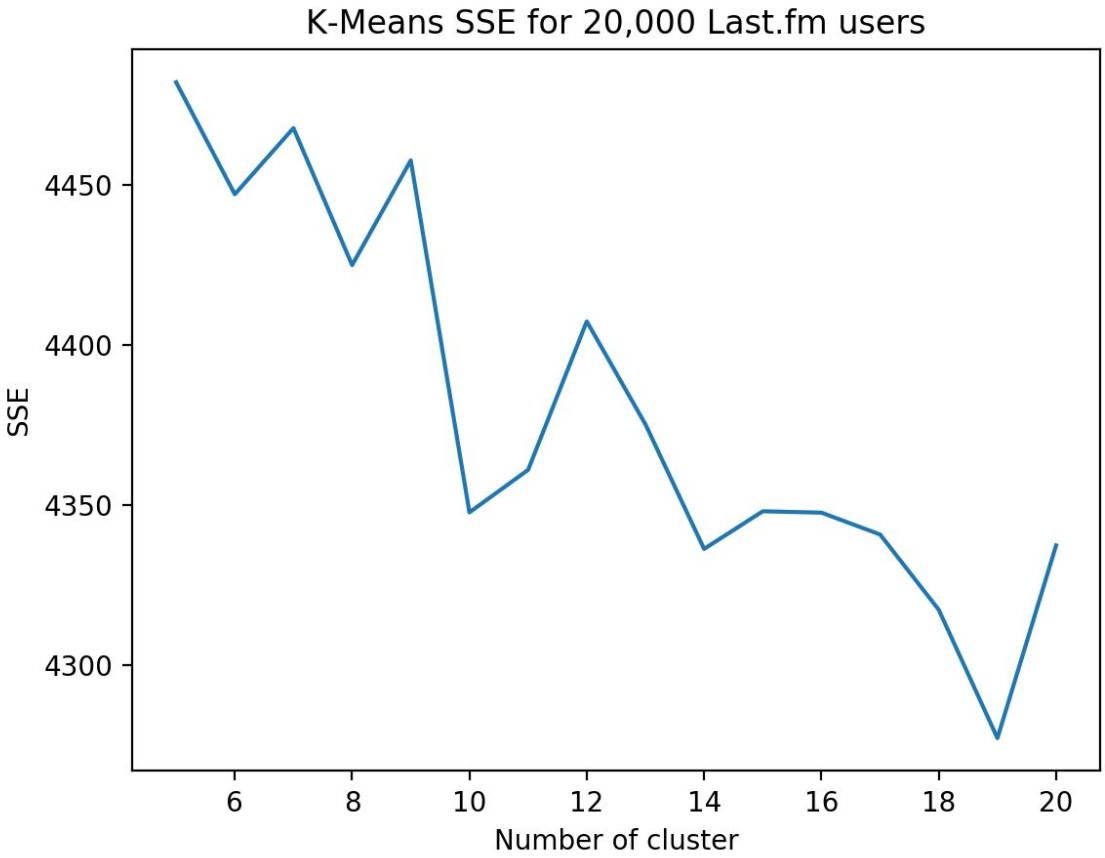
Min support of users for each top artist: 20%

A huge difference between our preliminary and final results (seen very clearly in a comparison between the two visualizations) shows that we refined our algorithm to pick out much more accurate and specific clusters.

Because our clusters depended on the frequent itemsets of top artists for each user cluster, tuning the minimum support and number of top artists used to cluster the users were pivotal in refining our results. We found that using between 10 and 15 top artists to distinguish and cluster users was optimal, since they contained (on average) approximately 90% of the total listening history of the users.

Evaluation metrics:

We used SSE (or cluster inertia in scikit-learn) as a way to determine the optimal size of our clusters.



We can see here that the optimal cluster size (that didn't specify/isolate too many individual users) ended up being around k=19. We tried larger numbers of K with our final optimizations and also found that k = 25, 26, and 27 ended up being pretty good, but the SSE would oscillate quite a bit.

We tried to incorporate TF-IDF vectorization in order to discount globally popular artists, but (probably because we implemented it incorrectly), lead to far worse clustering overall, so we stuck to just using the artist play count proportions as the value for each artist in the user vectors. This ended up making some really popular artists (Like Radiohead and The Beatles) appear in multiple clusters, but our results ended up still being pretty descriptive of general genres (Rock, Metal, Rap) along with exposing some more unique subgenres (K-Pop).

Conclusion and future work:

Preprocessing our data was a large step in order to suit our clustering needs, and this required vectorizing the users into a USER x ARTIST matrix. We also explored different clustering techniques. Our first thought was to use agglomerative clustering so that, at any hierarchical height, you could see different subgenres/subcommunities of users. However, because agglomerative clustering has time complexity of $O(n^3)$, we quickly found it impractical to cluster any subset of our dataset larger than 100 users. Once we switched to KMeans, our results were coming much more quickly and accurately.

We will also try to continue this as a personal project and perhaps implement a mobile version that uses real time location and spotify data to create user-driven "radio" stations, or crowd-sourced radio as described in our presentation.