Bridging the Gap: Solving Music Disputes with Recommendation Systems

Duncan Carlmark
Halıcıoğlu Data Science Institute
University of California San Diego
dcarlmar@ucsd.edu

Sarat Sreepathy
Halıcıoğlu Data Science Institute
University of California San Diego
ssreepat@ucsd.edu

Nayoung Park
Halıcıoğlu Data Science Institute
University of California San Diego
nap015@ucsd.edu

ABSTRACT

Many have probably found themselves in an uncomfortable conversation in which a parent is questioning why the song playing over a bedroom speaker is so loud, repetitive, or profane. If someone has never had such a conversation, at the very least they have probably made a conscious decision to refrain from playing a certain genre or artist when their parents are around. Knowing what music to play in these situations does not have to be an elaborate, stressful process. In fact, finding appropriate songs can be made quite simple with the help of recommendation systems.

Our solution to this issue actually consists of two recommendation systems that function in similar ways. The first takes music that parents enjoy and recommends it to their children. The second takes music that children enjoy and recommends it to their parents. Both of these recommendation systems create their own individual Spotify playlists that try to "bridge the gap" between the music tastes of parents and their children. Through user testing and user interviews we found that our recommenders had mixed success in creating playlists that could be listened to by children and their parents. The success of our recommendations seemed to be largely correlated with the degree of inherent similarity between the music tastes of children and their parents. So while our solution is not perfect, in situations where overlap between parents and children exist, our recommender can successfully "bridge the gap".

1 INTRODUCTION

It is not uncommon for teenagers and young adults to have disagreements with their parents. Usually these points of contention involve things like chores, curfews, and other byproducts of living under the same roof. However, these discussions may occasionally spill over into other areas of life, namely choice of music. Teenagers can often find themselves in uncomfortable situations where the songs they play are too loud, unfamiliar, or profane for their parents. In order to avoid these scenarios, teens must keep a mental record of what music their parents disapprove of and where this music is contained in their playlists and music libraries. Some teens might rather opt for complete silence in the presence of their parents than keep track of all this information. Finding music to play around one's parents does not have to be such a complicated or stressful process. In fact, this process can be made quite simple with the help of a recommendation system. To solve the disputes caused by music between teenagers and their parents, we have developed the music recommender website, bridgingthegapwithmusic.com. This web application leverages information from our users, teenagers and young adults, to quickly recommend music that they can listen to and enjoy with their parents.

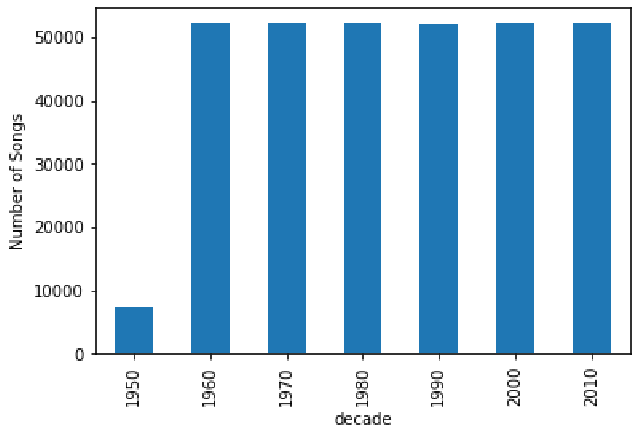


Figure 1: Billboard's Song Distribution By Year

Our recommendation system is designed to do two main tasks: recommend parents' music to users, and recommend users' music to parents. Each of these tasks has their own unique algorithm that generates a final playlist of recommendations on the user's Spotify account. Both recommendation algorithms try to find a rough middle ground between user and parent music tastes. With this in place, the user is able to quickly and automatically generate playlists which can be safely listened to and enjoyed in the presence of their parents.

2 DATASETS

2.1 Billboard

For the sample playlist generation for parent-to-user recommendations, we use the Billboard Weekly Hot 100 Singles dataset[1], which contains the top 100 songs every week from 1958 to 2019 (Figure 1). The columns of the dataset include: url, WeekID, Week Position, Song, Performer, SongID (concatenation of Song and Performer), Instance (number of times the song has appeared on the chart; each song starts with 1 and increments every time it disappears and reappears on the chart), Previous Week Position, Peak Position, and Weeks on Chart. There are 320,000 rows in the dataset containing statistics about 28,000 distinct songs. To generate features used in the recommender, we grouped the data by SongID to get the average week position, the first week it appeared on Billboard, the last week it appeared on Billboard, as well as the number of weeks it has been on the chart.

[1]https://data.world/kcmillersean/billboard-hot-100-1958-2017

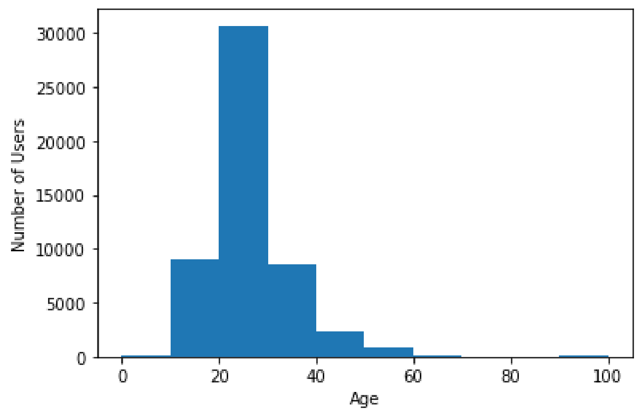


Figure 2: Last.fm's User Distribution By Age

2.2 Last.fm

For both parent-user and user-parent recommendation tasks, we work with two datasets from Last.fm[2], a music website based in the U.K., to get their users' listening history. The first dataset contains user profile data for all Last.fm users. After cleaning the dataset to deal with missing values, there are about 284,000 distinct users in the dataset with profile information including: UserID, gender, age, country, and date registered (joined Last.fm). We filter the dataset based on country to work primarily with recommending songs to people who live within the United States which reduces the number of users to around 52,000. For user ages, we could see that the users skewed younger with the majority between 20 to 40 (Figure 2). The second dataset contains the user listening history including columns: User ID, Artist ID, Artist Name, and Number of Plays. There are about 17 million entries in the entire dataset, and about 2.5 million entries when filtered by users living in the United States.

2.3 Spotify

Spotify[3] has a significant amount of information about artists and their music and we use Spotify's API to access this information and integrate it into our existing data. Our primary use of the Spotify API is to pull user listening information. Unfortunately, Spotify only allows access to the 50 most recent observations in a user's listening history, so instead we scrape a user's playlists to understand the different artists that they listen to. This does not provide as much information about the frequency of certain artists in a user's listening history, but considering the alternative it is a much better source of information.

We also use the API to pull artist objects that contain information about what genres an artist is defined by, and what other artists are related to them. We use this information in both of our recommenders so that we can filter recommended artists by user or parent genre preferences. The related artist information is used to broaden our artist recommendations and include artists that are not present in any of our datasets or the user's listening history.

[2]https://www.kaggle.com/neferfufi/lastfm
[3]https://developer.spotify.com/

Spotify also provides sonic information for all of the tracks listed on their platform in the form of track objects and audio feature objects. These objects contain features that describe a song in terms of its key, tempo, acousticness, and so on. These features are mainly used for some elements inside our parent-to-user recommendations, but other than that our project does not make much use of them. The track object also states whether a song is explicit or not which allows us to filter our recommendations to exclude profanity or other explicit content.

2.4 User Input

Prior to generating recommendations, we ask our users for some information regarding their parents. Since we make the assumption that parents do not have a Spotify account with listening history that we can leverage, we must ask these questions so we have some understanding of the parent's music taste. These questions are essentially limited to the parent's age, preferred genres, and a preferred artist. Parent age is used to approximate the era a parent grew up in and therefore the music they might have listened to growing up. The preferred music genres and the preferred artist are used to help our recommender understand what genres the parent is interested in, and to filter out any irrelevant recommendations.

3 METHODS

The recommender is split into two parts for different purposes:

(1) parent-to-user recommendations,
(2) user-to-parent recommendations

An optional sample recommender step exists for parents (or users) who do not have a Spotify account, or do not have enough listening history for us to work with.

3.1 Sample TopPopular Recommender

For this sample recommender, we use a TopPopular recommender where we rank the "popularity" of each song in the Billboard dataset and return the most popular songs (Figure 3). All songs in Billboard are filtered first by their release dates: we assumed the relevant timeframe for the parent's age to be from when they were 15 years old, to when they were 30 years old based on the premise that an individual's music preference is determined around age 15 and that they stop listening to new songs by the age of 30 [6]. The songs are then filtered by the parent's preferred genres and artists, and ordered by their popularity, which is determined by three statistics: instance, average weekly position, and the number of weeks the song was on the chart. The songs are ordered by lowest instance, lowest average weekly position and highest number of weeks to exclude seasonal songs like Christmas carols with high instance values, and prioritize songs that were ranked higher on the chart for longer. The final output contains the top songs that take the parent's input into account.

3.2 Parent-User Hybrid (CF+CBF) Recommender

The parent-to-user recommender uses a hybrid approach, meaning that it uses both collaborative filtering (CF) and content-based

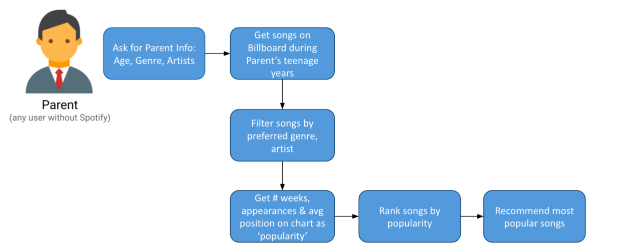

Parent
(any user without Spotify)
Ask for Parent Info: Age, Genre, Artists
Get songs on Billboard during Parent's teenage years
Filter songs by preferred genre, artist
Get # weeks, appearances & avg position on chart as 'popularity'
Rank songs by popularity
Recommend most popular songs

Figure 3: Workflow of Sample Recommender

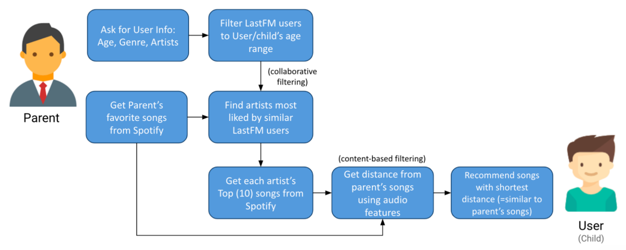

Parent
Ask for User Info: Age, Genre, Artists
Filter LastFM users to User/child's age range
(collaborative filtering)
Get Parent's favorite songs from Spotify
Find artists most liked by similar LastFM users
(content-based filtering)
Get each artist's Top (10) songs from Spotify
Get distance from parent's songs using audio features
Recommend songs with shortest distance (=similar to parent's songs)
User
(Child)

Figure 4: Workflow of Parent-User Recommender

filtering (CBF) techniques to recommend songs (Figure 4). We decided not to rely solely on CF methods unlike our user-to-parent recommender, due to the Last.fm dataset having a limited number of users aged over 40, which is where our parents are more likely to fit in.

The CF part of this hybrid recommender uses LightFM [3], a Python library using matrix factorization for implicit and explicit feedback. Using Spotipy [4], the Python library for Spotify's Web API, we get the parent's top 50 tracks on Spotify, get the artist for each track as well as the number of songs each artist had produced in the 50 songs. The Last.fm data also contains each user's listening frequency for each artist, so we use this frequency as the 'weight' for each item, meaning that the artist that is more listened to by the parent would be of more importance. We decided to use LightFM for the CBF part of this model as it uses latent representation to address the cold-start problem by grouping items that are liked by the same users, even if the items do not share similar features. To reduce the amount of data we work with and to improve the recommender's performance, we ask the parent for their user/children's age and only get Last.fm data within a similar age range as the user, getting users that are up to 2 years younger or older than the child user. Each user-artist interaction in both the parent's listening history as well as the Last.fm data is a positive interaction (instead of a negative interaction, where a user specifies that they did not like the item) used to build LightFM's recommender model, which also stores every user and artist in the interactions. LightFM essentially works by representing each user and item fed into the model as the sum of latent representation, and finding the dot product of those representations using the equation:

$$r_{u_i} = f(q_u * p_i + b_u + b_i) \tag{1}$$

where q_u represents the user's latent representation (= sum of latent vectors), p_i represents the item's latent representation, and with b_u and b_i each representing bias terms for the user and the item [2]. We chose WARP (Weighted Approximate-Rank Pairwise) for the loss function as it resulted in a higher AUC than BPR (Bayesian Personalized Ranking), the only other loss function suitable for having only positive interactions available as data. For each user in the model, the artists are ranked by the dot product; we get the artists with the highest dot product value as an output from this model.

Although LightFM has CBF functionality, we found out that its item features implementation does not work properly and could not be used for CBF. Thus, for the CBF part of this recommender, we decided to use euclidean distance (instead of a common CBF method like cosine similarity, suggested by Rethana [5]). to calculate the distance between other songs and the parent's top tracks, using their audio features from Spotify. From the LightFM's output list of artists (the number of artists is defaulted to 10 for convenience, since Spotify's API only allows pulling audio information for up to 100 songs at a time), we get the top 10 popular songs from each artist and compile a list of 100 songs. Then we calculate the euclidean distance between those songs and the parent's preferred songs, using the formula:

$$d_{p_q} = \sqrt{\sum_{i=1}^{11} (p_i - q_i)^2} \tag{2}$$

where p_i and q_i are the i$^{\text{th}}$ audio features from arbitrary song p and parent's song q, from 11 features: danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, and tempo. We get the minimum Euclidean distance for each song in the recommended list from all of the parent's songs, and rank the songs by their shortest distance from any parent's song. The final output is a list of songs ordered by ascending euclidean distance, to recommend songs that are most similar to the parent's preferred songs in terms of audio features to the user.

3.3 User-Parent Collaborative Filtering (CF) Recommender

With our user-parent recommender (Figure 5), we first request information from the user about their parent including their age, favored genres, and favorite artist. To build the user to parent recommender, we work primarily with the Last.fm data. Since we want to recommend to parents, we want to build similarity between the user and the parent so the playlist output would be enjoyed by both the user and the parent. We do this by extracting the users from the parents' age range from the Last.fm user profile dataset (user_id, age, country, etc). We pull these Last.fm users' listening history from the Last.fm listening history dataset (user_id, artist_id, artist_name, plays) which helps to provide a basis for the recommender. We then want to pull from the user's listening history, specifically user playlist data which we can use to approximate a listening history similar to the one found in the Last.fm listening history dataset. We generate a user-artist interaction matrix using the user playlist data such that it is analogous to the user-artist interaction matrix created.

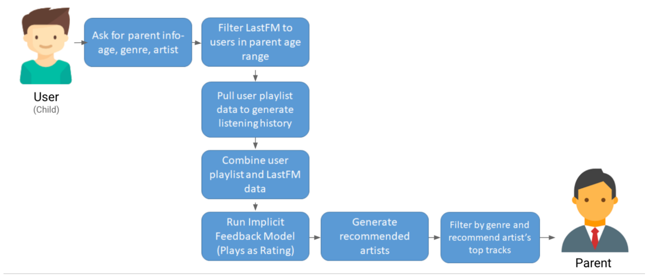

User
(Child)
Ask for parent info- age, genre, artist
Filter LastFM to users in parent age range
Pull user playlist data to generate listening history
Combine user playlist and LastFM data
Run Implicit Feedback Model (Plays as Rating)
Generate recommended artists
Filter by genre and recommend artist's top tracks
Parent

Figure 5: Workflow of User-Parent Recommender

After gathering the user's listening history, we can use the user's listening history to find similarities between the user and the users from the parents' age range in the Last.fm dataset. We also want to normalize the plays (frequency) column for the listening history so we have a common scale and there are less concerns about bias. We then move to create categories (assign numbers) for each unique user and artist in the dataset to clean the data and also transform the dataset so we can build the user-item interaction and item-user interaction matrix. The user-item interaction matrix has shape [num_user $*$ num_items] and the item-user matrix has shape [num_item $*$ num_user]. We implement this data transformation such that the [i^{th}, j^{th}] entry of a user-item interaction matrix is the rating of the ith user for the jth item and that is similarly applied to the item-user interaction matrix. We then combine both the Last.fm user-artist interaction matrix with the user-artist interaction matrix generated with the user's playlist data to have a user-artist interaction matrix ready for the collaborative filtering model.

We then use Implicit [1, 2] using alternating least squares to build similarity between the user's playlist and older users (users in the parent's age range) in the LastFM dataset. For our implicit feedback model, we use plays as our "rating." Once we fit the model, we can now build recommendations for the particular user, in this case the user who has requested a playlist for their parent, and a list of recommended artists is returned. In our initial responses collected by the user, we collected the parent's favored genres so we use that data to filter down the recommended artists. We use Spotipy [4] to collect the genre for each artist and we then filter the list of recommended artists down to artists that are represented by the favored genre of the parent. We can then use Spotify's top tracks to build a playlist from these filtered recommended artists to get a final list of songs that is outputted to a playlist on the user's Spotify account.

4 RESULTS

For the parent-user recommender, we evaluated LightFM, the CF part of the recommender, using AUC and NDCG. AUC, or Area Under the Curve, refers to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one . Generally speaking, a higher area under the curve means a more effective recommendation system. The LightFM model resulted in a training AUC of 0.988, and test AUC of 0.618 (Table 1). NDCG, or Normalized Discounted Cumulative Gain, is a measure of ranking the quality of a recommender system by the difference in actual rankings and predicted rankings. Unfortunately, we were unable to evaluate the entire hybrid recommender as the CBF part recommends songs instead of artists like the CF part does, but evaluating the CBF method using a sample user's top 10 songs resulted in an NDCG of 0.80.

When evaluating the user-parent recommender, we relied on both offline testing as well as user feedback studies. For offline testing, we used the metric AUC or area under the curve to evaluate the recommender. When evaluating the implicit feedback recommender, we found that the AUC would average around 0.8-0.82 which suggested that it was generally effective (Table 1).

Table 1: Performance of Our Recommenders

Recommender	Training/Test AUC	NDCG@10
Parent-User CF	0.988/0.618	
Parent-User CBF		0.80
User-Parent CF	/0.8-0.82	

To broaden our understanding beyond metrics we conducted user studies where we collected feedback from users about their experiences with our both recommenders. What we found was that the effectiveness of our recommendations was closely tied to the inherent similarity between users and their parents music preferences. When there was significant overlap in genre preference, both users and parents enjoyed the playlists they received. However, as the differences between parents and their children increased the enjoyment of our recommendations also somewhat decreased. The drop in enjoyment was especially noticeable when there was a complete difference in music preferences between the two parties. Our best example of this is an instance where the user's parents listened to singer/songwriter artists, but the user only listened to EDM and other genres of rave music. Since there was such a drastic difference in the genres and artists our recommender failed to find any commonalities or slight mutual interests. In these situations the recommender leans towards the interests of the parents, so at the very least the user can listen to the playlist around their parents even though they might not enjoy the music that was recommended. This means that at its worst, our recommender can accomplish our goal of giving users something to put on around their parents, but it cannot fulfill the goal of having both parties enjoy what is being played.

5 CONCLUSION

When first approaching the problem of music recommendations, we wanted to address disparities in music taste. We could see instances in our own lives where disparities in music taste might lead to conflict, especially with our own parents. This realization served as our motivation for developing bridgingthegapwithmusic.com, since automated recommendations could simplify the process of finding music for both younger and older audiences to enjoy. In our recommender, we attempt to solve this problem through two approaches: recommending music from 1) parents to users and 2) from users to parents.

(1) In our first approach, we find songs that parents would enjoy and recommend similar songs to their children. We do this

using Billboard rankings as well as parent Spotify history to generate a basis for what we expect parents to enjoy. We then request data about the children, like their favorite genre, to build similarity between the parent's songs and the user's music taste.

(2) In our second approach, we find songs that children enjoy and recommend similar songs to parents. We use the children's Spotify listening history and request information from the parent, like their age, to build similarity and offer recommendations.

When evaluating our recommenders with user feedback, we found that our algorithms performed well when there was some degree of overlap between users and their parents. As that overlap begins to disappear the enjoyment of our recommendations tends to decrease. Whenever the differences between a user and their parent become so extreme that there is little to no overlap, or algorithms struggle to produce recommendations that can be enjoyed by both parties. In general our recommender can consistently generate music that can be played around one's parents. Whether or not children and their parents can enjoy this music together largely depends on the overlap between their music preferences.

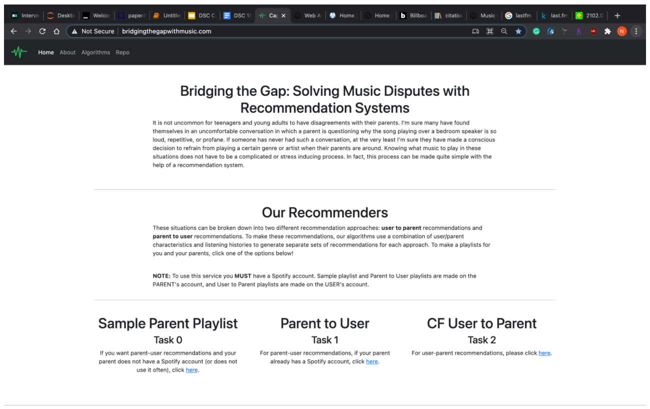
In the future, our project could be expanded to include more comprehensive features rather than relying principally on collaborative filtering techniques. One potential pathway forward could be to emphasize sonic characteristics more. We include these characteristics partially in our parent-user hybrid model, but we do not use them anywhere else in this project. Using these characteristics might help to lessen the impact of inherent genre preference differences by looking for more abstract and unnoticeable similarities between the music preferences of children and their parents. An entirely different way forward could be to look at the situation from a different perspective entirely and try to nudge the music preferences of children and their parents together closer over time. This could function in a way where both parties are given new playlists every week that slowly come closer together in genre or sonic characteristics, literally bridging the gap between parents and children over time. Regardless of what is done next, it is clear that there is an abundance of potential improvements that have yet to be tested or explored in our niche of music recommendations.

6 USER INTERFACE

The UI for bridgingthegapwithmusic.com is relatively simple. The homepage has a short mission statement followed up by a brief description of our recommendation algorithms. Links to each of our recommendation algorithms sit at the bottom of the page. There are other tabs in the navigation bar at the top of the page that link to more information about the project (Figure 6).

When a user clicks on one of the recommendation algorithms they are taken to a short form. This is where we ask the users for the limited amount of parent information that our algorithms need. Once the users submit their information, they are taken to another page where they wait for their playlist to be generated (Figure 7).

Once the algorithms are finished generating recommended songs, they are added to a playlist created on the user's Spotify account. The playlist is named after the corresponding algorithm that was



Bridging the Gap: Solving Music Disputes with Recommendation Systems

It is not uncommon for teenagers and young adults to have disagreements with their parents. I'm sure many have found themselves in an uncomfortable conversation in which a parent is questioning why the song playing over a bedroom speaker is so loud, repetitive, or profane. If someone has never had such a conversation, at the very least I'm sure they have made a conscious decision to refrain from playing a certain genre or artist when their parents are around. Knowing what music to play in these situations does not have to be a complicated or stress inducing process. In fact, this process can be made quite simple with the help of a recommendation system.

Our Recommenders

These situations can be broken down into two different recommendation approaches: **user to parent** recommendations and **parent to user** recommendations. To make these recommendations, our algorithms use a combination of user/parent characteristics and listening histories to generate separate sets of recommendations for each approach. To make a playlists for you and your parents, click one of the options below!

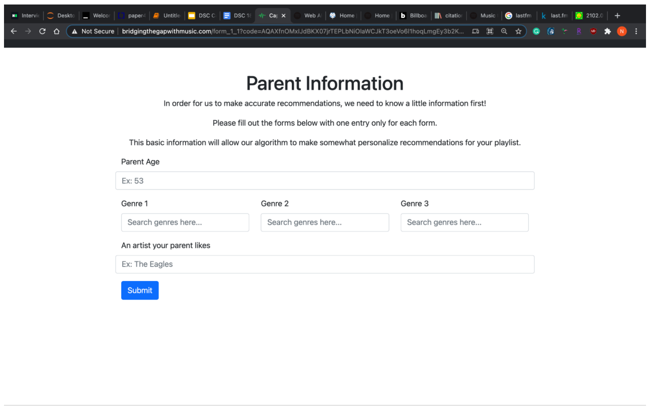
NOTE: To use this service you **MUST** have a Spotify account. Sample playlist and Parent to User playlists are made on the PARENT's account, and User to Parent playlists are made on the USER's account.

Sample Parent Playlist
Task 0
If you want parent-user recommendations and your parent does not have a Spotify account (or does not use it often), click here.

Parent to User
Task 1
For parent-user recommendations, if your parent already has a Spotify account, click here.

CF User to Parent
Task 2
For user-parent recommendations, please click here.

Figure 6: Home Page - Description of Recommender and Tasks


Not Secure | bridgingthegapwithmusic.com/form_1_1?code=AQAXfnOMxIJdBKX07jrTEPLbNiOlaWCJkT3oeVo6l1hoqLmgEy3b2K...
Submit

Parent Information

In order for us to make accurate recommendations, we need to know a little information first!

Please fill out the forms below with one entry only for each form.

This basic information will allow our algorithm to make somewhat personalize recommendations for your playlist.

Parent Age
Ex: 53

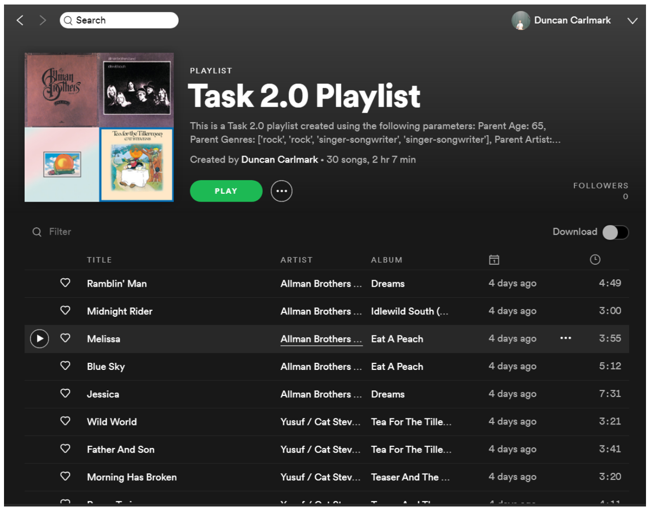
Genre 1
Search genres here...

Genre 2
Search genres here...

Genre 3
Search genres here...

An artist your parent likes
Ex: The Eagles

Figure 7: Requesting Information from User



Task 2.0 Playlist

This is a Task 2.0 playlist created using the following parameters: Parent Age: 65, Parent Genres: ['rock', 'rock', 'singer-songwriter', 'singer-songwriter'], Parent Artist:...

Created by Duncan Carlmark • 30 songs, 2 hr 7 min

TITLE	ARTIST	ALBUM		
Ramblin' Man	Allman Brothers ...	Dreams	4 days ago	4:49
Midnight Rider	Allman Brothers ...	Idlewild South (...	4 days ago	3:00
Melissa	Allman Brothers ...	Eat A Peach	4 days ago	3:55
Blue Sky	Allman Brothers ...	Eat A Peach	4 days ago	5:12
Jessica	Allman Brothers ...	Dreams	4 days ago	7:31
Wild World	Yusuf / Cat Stev...	Tea For The Tille...	4 days ago	3:21
Father And Son	Yusuf / Cat Stev...	Tea For The Tille...	4 days ago	3:41
Morning Has Broken	Yusuf / Cat Stev...	Teaser And The ...	4 days ago	3:20

Figure 8: A finalized playlist on Spotify

used and the description states the features that were used to generate it (Figure 8).

REFERENCES

[1] Ben Frederickson. 2017. *Implicit.* https://github.com/benfred/implicit

[2] Viktor Kohler. 2017. *ALS Implicit Collaborative Filtering.* Medium. Retrieved March 12, 2021 from https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

[3] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys '15)*, Vol. 1448. CEUR Workshop Proceedings, 14–21. http://ceur-ws.org/Vol-1448/paper4.pdf

[4] Paul Lamere. 2014. *Spotipy.* https://github.com/plamere/spotipy

[5] Mark Rethana. 2018. *Building a Song Recommendation System using Cosine Similarity and Euclidean Distance.* Medium. Retrieved March 12, 2021 from https://medium.com/@mark.rethana/building-a-song-recommendation-system-using-cosine-similarity-and-euclidian-distance-748fdfc832fd

[6] Seth Stephens-Davidowitz. 2018. *The Songs That Bind.* New York Times. Retrieved March 12, 2021 from https://www.nytimes.com/2018/02/10/opinion/sunday/favorite-songs.html