Modeling a Social Network for Music Recommendation

Using NetworkX and Matrix Factorization to Predict Songs and Users

Jacob Berkofsky

Applied Mathematics

Colgate University

19 December 2019

Abstract

In this paper, I will introduce an algorithm that analyzes and maps a social network for music recommendation. This project presents ways to model music data based on users of the same age group. The goal is to use a small data set to recommend music to users given their public play-lists and music data. I use simple mathematical models to compare user data in order to uncover the mystery of finding new music to listen to.

1 INTRODUCTION

As our decade comes to a close, we can look back at recommender systems as a luxury popularized thanks to our increasing use of the internet. The algorithms started with consumer giants such as Amazon and Facebook, but continued across all platforms, more recently penetrating into the music-streaming industry. First used by Facebook to match you with friends, and then by Netflix to recommend its users new shows based on what they had already watched, music streaming services now look to make finding new, enjoyable music easier than it ever has been.

The main difference between the success of the music industry and the success of companies such as Netflix, Facebook, Amazon, etc. is that the music industry has failed to connect their users with each other. Most of the algorithms focus on automatic semantic notation which compares songs based on their genre, how they sound, and their music structure [7]. Very few algorithms actually focus on user-centric recommendation; putting the user's and how the interact with other users at the forefront of the study [3]. This leaves us with a question: How can we model user networks to create a more-efficient way to recommend music?

1.1 SIMILAR ATTEMPTS

As noted in the Introduction section, there have been many attempts to quantitatively attain a simple and effective recommender system, but very few focus on the user's music profile in general. This project's goal is to use previous methods – used by other researchers to analyze data – in order to establish a small-scale user network system for modeling.

1.1.1 Deep Learning

The first attempts at creating an algorithm for music recommendation involved neural processing and temporal dynamics in order to create a profile for a user. These methods involve a hybrid approach to the algorithm; used are both neural studies, which involves the sequencing of a user's listening tendencies, as well as studying the science behind tunes, beats, and artist approaches [4]. It furthers the study with mathematics through the application of a user-rated score [6], which will be used later on via a combination of mathematical methods, code, and inferences.

1.1.2 Million Song Dataset Challenge

By far the most well-known user-centric algorithm came from a challenge called the *Million Song Dataset Challenge*. This method used provided Spotify data (a million songs) in order to create a recommender system with as little error as possible. These researchers created a "taste profile," essentially putting together music that was produced similarly based on a user's liked songs [1] . In fact, it used the beats of the song and even a listener's neural signals, just as the methods mentioned in 2.1, to calculate what people would think of when they heard songs with similar melodies, words, and beats [1] in order to minimize the Root Mean Squared Error [3]. Using RMSE (Root Mean Squared Error), humans have been able to judge how accurate these projects have been, by comparing the predicted value of the similarities of two songs to the observed value created via their algorithm [1] . This value is very accurate for comparing songs based on the user's created advanced taste profile, but it leaves much to be desired when recommending other user profiles.

2 METHODS

This section contains the several methods used on the path to creating the user network. Each one was considered throughout, and overall a combination of all methods was used in order to attain proper results.

2.1 Collaborative Filtering

The Million Song Dataset Challenge continued its journey toward minimizing error by implementing a technique that has been used most notably by Netflix, Amazon and Facebook. This technique is known as collaborative filtering, which is an approach to recommender systems that involves collaboration between a site and its users. Collaborative filtering ”relies only on past user behavior without requiring explicit creation of user profiles” [3]. It ”analyzes relationships between users and interdependencies among products to identify new user-item associations” [5]. Essentially, in Netflix's case, networks are created by surveying users for feedback based on shows or movies they have already watched, i.e. rating a show out of five stars. If multiple people have given similar feedback for that show/movie, then the company will recommend you a show or movie that those users have previously rated highly.

Another example of this is the “People Who Bought This Item Also Bought” section on Amazon. By having their users rate products, the company can see that if one product is rated highly by two customers, chances are that if they are also observing another different, they will both also like it. In this project, we will rely on past user listening tendencies. We find this information based on a user's public Spotify playlists, found through the API. If one user has a song that they have saved in a play-list and another user has the same song, the two users will be interconnected in the user network model, further explained below.

2.2 Spotify API and Data Collection

API's, or *Application Programming Interfaces*, are online databases used by coders to access a company's information. This project uses the Spotify API in order to make a recommender system based on the shared songs of Spotify users. API's have become a dependable and highly used way of connecting application interfaces together; allowing authors to build software applications using other websites public data. It is code that gives programmers the ability to access the features of an operating program. In many cases, this data is used to monetize websites or applications, but it is also commonly used for research and teaching purposes; making it easier for people to understand public data. The Spotify API is used to access the public data of several college students, and examine these networks in several in order to recommend both similar users, as well as use artists that the similar users have in common in order to recommend songs they may not already have heard.

This project surveys 27 Spotify users, all of whom are current college students. As requested by those surveyed, the user's names will be left out of this study for anonymity's sake. This allowed for access to all of a user's public play-lists with just their username/ID code. With permission from Spotify, each user's data was collected with the following code in the Mac Terminal:

```
python3 spotifyxx.py username > username.txt
```

- `python3` represents the python program being used to code.
- `spotifyxx.py` represents the created python code used to access and print a user's playlists.
- `username > username.txt` specifies which user's data to collect, and then print it to a *.txt* file saved into a specific folder.

2.3 Python Coding

Due to the simplicity and ability to use many imported pieces of software together, it was determined Python was the best program to collect data for the recommender system. The Spotify API in Python used to access this information is called *Spotipy*. Spotipy creates an easy transition from the difficult format of Spotify's data to a usable, readable option for its users. This allowed for short, relatively simple code, and continuous access to any Spotify data with many options for code. This code was stripped of its spaces, and then placed into a Python dictionary for use later in NetworkX.

Following the code that retrieved the user data, it was vital to collect artists data. By replacing `spotifyxx.py username` with `pythonxx.py artistname` in the above Terminal operation, access was gained to all albums and songs by any given artist in the database. Although this information can be obtained with Spotipy, the vast amount of potential artists, even in a small sample size, created issues when storing. This code is best used when getting a specific rating for user feedback (deep learning) [6 − 7], or when trying to minimize error, such as in the Million Song Dataset Challenge [3]. It is less vital in user-centric analysis.

2.4 Network Theory and NetworkX

The idea of a user-centric recommender system is to analyze data and make conclusions based on each specific user, rather than songs in particular. In order to achieve this, both Schedl and Koren et al. suggest an expansion on Markov Chains [5]. Early recommender systems used Markov Chains to compute the probabilities of a user going from one song to another based on the previous path [1, 6]. Rather than using this less efficient approach, the Python input, called NetworkX, allows us to easily manipulate dictionaries to create networks of data. Once each user's data was stored into a dictionary, it was time to create a network of users. NetworkX has countless functions and ways of organizing data, allowing for easy manipulation of data in order to create models.

Networks contain nodes, which in this case are each user. A node is created if a user has data that was previously compiled by the starter Python code. Nodes are then connected by edges, showing their is a connection between two nodes. These edges may be weighted, creating a similar effect to a probability of going between two nodes in a Markov Chain [6]. NetworkX allows for simple traversing of dictionaries, creating nodes based on the key and edges based on the value. In the dictionaries created for each user, users are the keys, while songs are the values. In order to process the information needed to create edges, Python code was established to create vectors out of the dictionaries, and our entire user network represents a matrix of vectors, each one being a set of similar songs between two users.

2.5 Matrix Factorization

2.5.1 Creating the User Matrix

Each of the 27 users have a library of songs that represents a node in the network of college students. When iterating through each library, if one user represented by the set, U_i, has any similar songs with another user, U_j, then the two users will be connected, and the spot in the matrix represented by:

$$\dot{\mathbf{U}}_{ij} = \mathbf{U}_i \cap \mathbf{U}_j\,. \tag{1}$$

where i and j represent two different Spotify users [3]. $\mathbf{U}_i \cap \mathbf{U}_j$ determines the intersection of two libraries of songs. If two users have any intersection that is not $\emptyset$, then they have atleast one song in common, and an edge between the two user nodes is created. This will appear in the matrix as a vector, but it will later be granted a numeric value for the edge weight, or $len(U_{ij})$. $\mathbf{len(U_{ij})}$ represents the amount of shared songs that two users have together [2]. This is used to determine the edge weights that connect two users.

2.5.2 Creating an Artist Matrix

For each user in the matrix U_{ij} defined above, there belongs a Python dictionary of artists and songs. Each spot in the matrix, A_s represents a vector of all of the songs by a certain artist in the dictionary. As explained, there are infinitely many artists to create, but an artist network exists in which each place in A_s can be used for matrix factorization to achieve a specific score.

2.5.3 Factorization

After the matrices have been formed, we can take the vector **BothLike** = $\mathbf{U_{ij}}$ (1), and iterate through to determine a score [2], represented by the equation:

$$r = \mathbf{q_s^T} \mathbf{p}_u \,. \tag{2}$$

- Each song r is associated with an artist vector, q
- Each user u is associated with a user vector, p
- r represents the vector of scores given to each song in (2). This score allows us to see which artist we should look at to make the recommendation [2].

2.6 Cliques

The creation of a user network gives flexibility into finding the path between users. In order to further analyze the users and their edge weights, NetworkX gives functions for types of models. In this case, it became apparent to me through trial and error that looking for communities of the users would further the analysis by creating a simple way to map the probabilities of connections between users. Due to the vast number of connections and small sample size, finding strict communities appeared to not sufficiently portray the connections.

This led to the research of cliques in network theory. Just as in social communities, cliques are groups of nodes that stick together through highly connected edges. This does not necessarily mean that all nodes in a clique are highly connected with large edge weights, but it does mean that each node is highly connected to a node in the clique that all other nodes in the clique are also highly connected. To find the cliques using NetworkX the following code was compiled in Python:

```
list(nx.find_cliques(userGraph.to_undirected()))
```

- `nx` is all of the imported NetworkX functions through Python.
- `find_cliques()` is the specific NetworkX function for finding cliques in the already created network of users, called `userGraph`. These cliques are, for each node, u, the largest complete subgraph of userGraph that contains u.
- `to_undirected()` is the NetworkX function that turns a directed graph into one that has edges with no direction. In our case this is preferred, because if user u is connected to user v, then v is also connected to u by the same edge and edge weight.

3 RESULTS

This section will include both the figures created in Python, as well as the results. All is accompanied by analysis of the findings, and further explanation of how and when each previously discussed method is used.

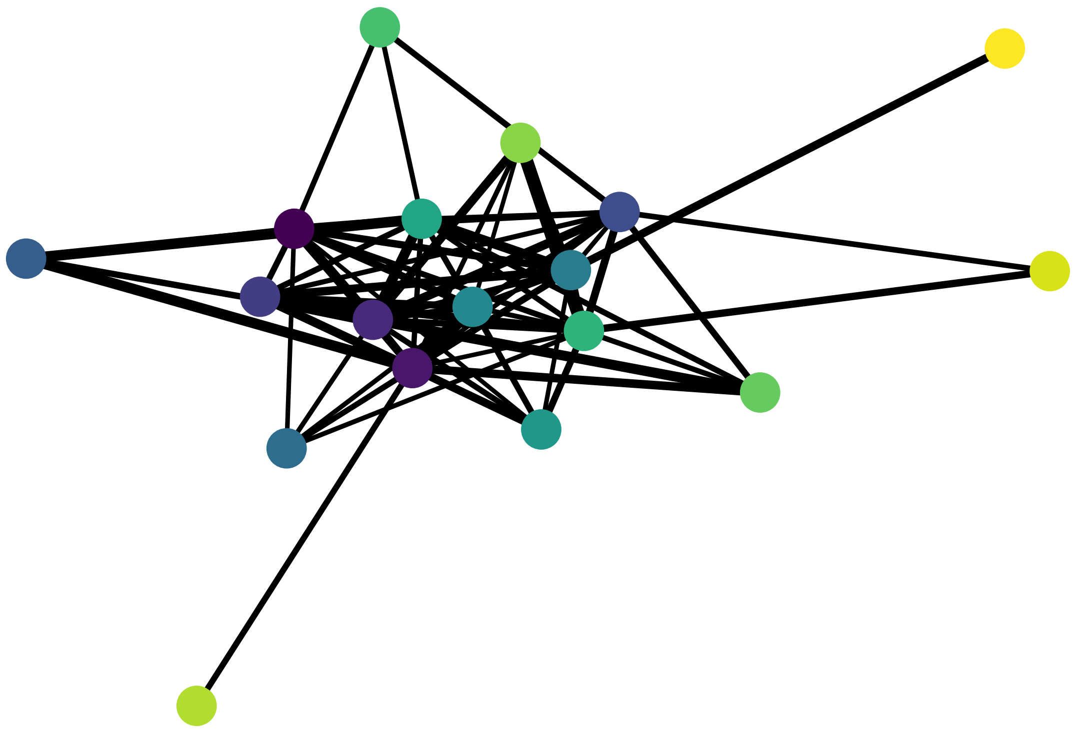
3.1 Network Analysis

After creation of the user network, our models were chaotic and unclear. In order to further quantify the data, a minimum limit was added in order to encourage the program to not include several nodes and edges in the network. This limit was set as the aforementioned $len(BothLike)$, or the length of the intersection between two user's Spotify libraries. In order to get better views of the model, and analyze more about our network, this minimum value will be changed throughout, as shown in the code below with easy manipulation of the value set for the variable X:

```
BothLike = set(userDict) & set(otherUserDict)
if (len(BothLike) > X):
  userGraph.add_edge(file, OTHERfile, weight = len(BothLike))
```

- `set(userDict)` is the vector of songs saved by a given user, say u. In this case it is stored in the program as a Python dictionary.

Figure 1: This represents 23/27 people studied. Each user shown has at least 80 songs in common with at least one other user.



- `set(otherUserDict)` is the vector of songs saved by a different user, say v. In this case it is also stored in the program as a Python dictionary.
- The `&` command in Python creates the intersection between `userDict` and `otherUserDict`.
- `add_edge` is the NetworkX function to add edges to a directed graph called `userGraph`.
- `file` and `OTHERfile` are the names of college students surveyed that will be added to the graph as unnamed nodes.
- The edge weight between the two users is created with `weight=len(BothLike)`.

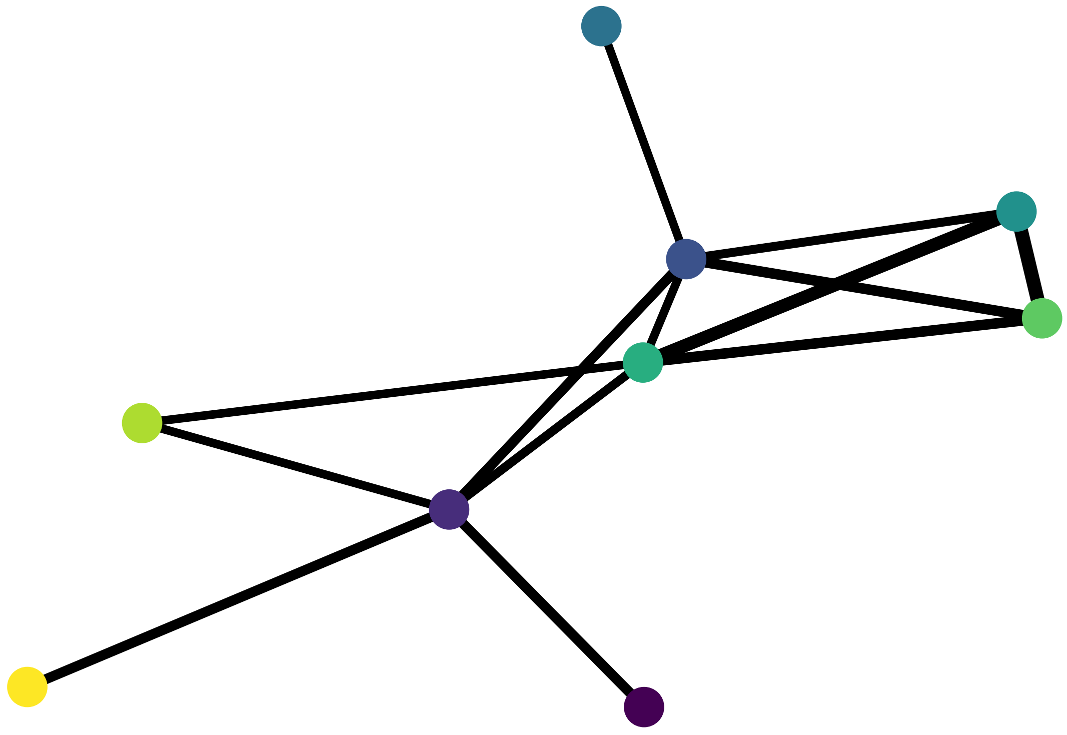
3.1.1 Large Scale Network

For clarity's sake, the limit set to encompass the maximum amount of people was 80. As mentioned, figure 1 is hard to read, as it represents most of the entire surveyed library of college students. This means each user is connected to all other users through some path through the network. It is hard to conclude much from this figure, but the colors of the nodes hold large importance. It is not randomness, in fact, notice the cooler colors are closer to the middle and the warmer ones appear to stay near the perimeter of the network. This occurs because of the amount of edges that each user has. The cooler and darker the color assigned to the node, the more connections this user has with other users throughout the community. This does not take into account edge weight, just if the two users have greater than 80 songs in common. From this we see that not many users are extremely interconnected (purple), and we can see that many do not have a strong correlation with more than a couple users. It is inferred that as we shrink our range of focus in order to sharpen our results, these users will be left out, as they will not have enough shared songs to contribute to the recommender system.

3.1.2 Medium Scale Network

Figure 2 shows us a better glimpse into what is actually happening in our network. Here we can see clear paths between nodes. Also, notice that in Figure 1 some lines are thicker than others. This means that they have more songs in common (higher edge weight) than the ones connected by thinner lines. In the code from section 3.1, the edge weight is set. As we increase the minimum inclusion value, or X, the lines will all be thicker, as each user has a higher edge weight between nodes, as they would not be included otherwise. This is why it is more difficult to see the difference in the thickness of the edges between nodes in Figure 2. Furthermore, it should be noted that the nodes shown in Figure 2 are not the same color as they were when included in Figure 1. This occurs because the centrality of the nodes and there ability to connect multiple nodes has changed with the increase in minimum value. Nonetheless, our cliques are beginning to form. We even start to see small communities that are highly interconnected. The four blue and green nodes in the upper right hand corner forming a trapezoidal-like shape are all connected with each other. They likely have many of the

Figure 2: This represents 9/27 people studied. Each user shown has at least 150 songs in common with at least one other user.

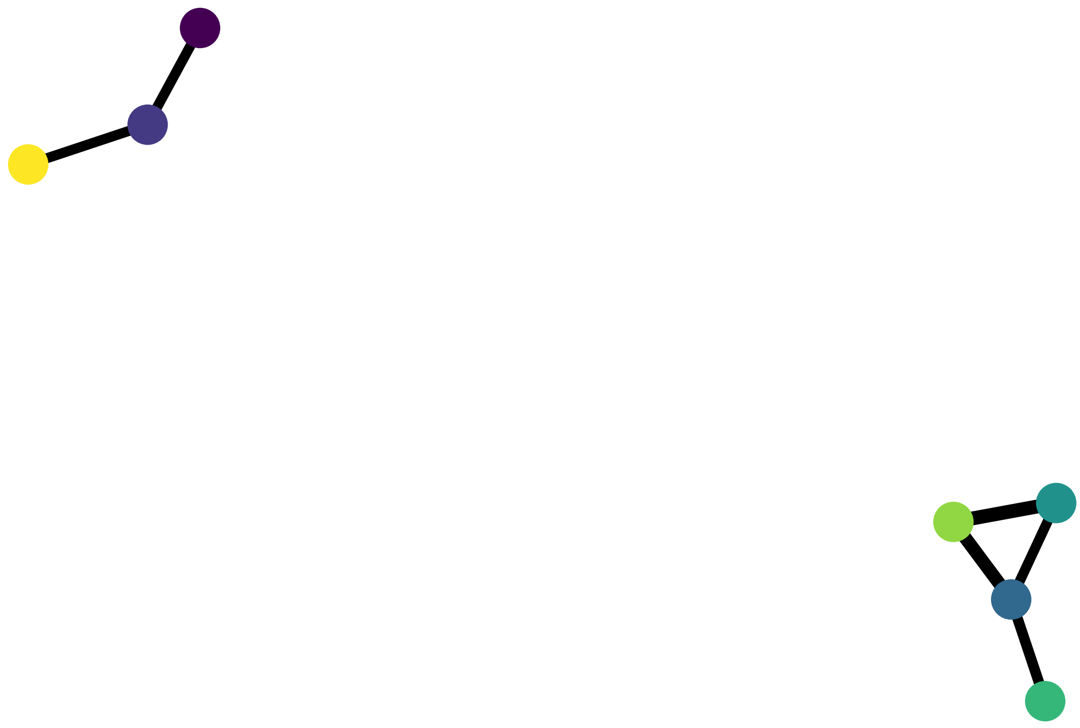


same songs, thus their music pages and songs would likely be good recommendations for one another. In order to find for sure, we should continue to increase the inclusion value, and find the cliques that are starting to form.

3.1.3 Small Scale Network

Figure 3 is a snapshot of this project's full results. These users have so much in common, they are in the same cliques of users. This is how we can recommend both the similar users, and then use matrix factorization to show which songs should be recommended. These are the closest we get to a user recommendation that is possible with such a relatively small library of songs. As we increase X our data gets more and more specific.

Figure 3: This represents 7/27 people studied. Each user shown has at least 180 songs in common with at least one other user.



3.2 Clique Analysis for User Recommendation

The clique represented in the top left corner of Figure 3 contains three users that are not all connected to one another. The yellow user and violet user are each connected to the blue user, but it is important to see that they do not contain an edge between one another. Although the blue user would be a great recommendation to follow for each of the two users, it is is the secondary connection here that is important. Their libraries may not necessarily appear similar, but if both of these users have over 180 songs in common with the central blue user, then their taste in music is probably very similar. Cliques present us with the ability to not only compare two users head to head, but gives us groups of users that are connected through other users.

The clique represented in the bottom right corner of Figure 3 shows even more important information. Unlike the other clique, we see a triangle of users all connected and one user on a branch. The user that only has one connection can be treated very

Table 1: Connections Between Users

User	**JOE**	**DAN**	**DAVE**	**JOHN**
Color	Yellow-Green	Aqua	Grey-Blue	Green
Primary Connection(s)	Dan, Dave	Joe, Dave	Joe, Dan, John	Dave
Secondary Connections	John	John	$\emptyset$	Joe, Dan

similar to the two users mentioned previously from the other clique. This user will have a primary user to recommend, and then two other secondary users that have less of the same songs, but a lot of similar music. Furthermore, the difference in edge weight between users in this clique is much more evident than in the clique of three users. The lime green user has two very thick edges, one with the aqua user and then light blue user. This shows us that this user has a large amount of songs in common with the other two users, and – although unnecessary for a small sample size – if we continued to raise the inclusion value, X, then it is likely the network would still contain these three nodes. In a larger sample size this is quite vital; if the pool of users is larger, there will be more cliques forming, causing less disparity in user connection. If the edge weight is so great, these three users will likely continue to be highly correlated even when the amount of users surveyed is increased.

3.2.1 User Recommendation Results

This section will show a table representing the proposed social network for a user based on clique and network analysis. Aliases will be used for clarity's sake. This table will focus specifically on the previously analyzed clique in the bottom right corner of Figure 3.

Based on Table 1, we can see who the central user in the clique is, Dave. Dave has over 180 songs in common with each of Joe, Dan, and John. The first thing we will look at for user recommendation is the primary connections. These users have a lot in common, and their tastes in music are likely very similar; they must like many of the same genres/artists if they are included in the same clique. Next, as Dave has multiple primary connections and no secondary connections, it is imperative to see which of those connections is the strongest. Dave has a much stronger connection with Joe than either Dan or John, as evidenced by the very thick edge, and thus large edge weight. These two users would still be represented even if we raised the inclusion value to 250. Thus Joe will be the first user recommended for Dave. If Joe has a primary connection with Dan, and Dave has primary connections with both Dan and John, then the second user recommended would be Dan due to the secondary edge weight connecting Joe and Dan, which would hold even if the inclusion value was raised to 240. The connection with John would only hold until the value reaches 200, meaning he is the next to be recommended.

In order to properly understand the relationship with users that have secondary connections with each other, John's recommended users will be analyzed further. John only has a primary connection with Dave, so Dave will be the first to be recommended to John. Dave then has connections with both Joe and Dan, but Dave has a stronger connection to Joe, so Joe will be the next up in line, followed by Dan. Even if these Joe and Dan do not have a quantifiable connection with Joe at this inclusion value, their secondary connection through Dave, and a strong one at that, suggests that their libraries may be similar. Another possibility is that Dave has a broad range of music taste and a thus a large library, and John fits in with a genre of music that Joe and Dan do not listen too much.

Additionally, another possibility of the lack of primary connections for John within the clique may be explained by a slightly smaller library. If John has fewer songs, then it will be more difficult for him to get users to recommend, as he will have fewer songs in common with all involved. Nevertheless, due to the extremely high edge weight between Dave and Joe, as well as between Joe and Dan, John figures to have highly similar music to all involved in the clique, even if their connection may be weaker than a primary connection. It can be concluded that the music is similar enough to use for successful user recommendation. The final recommendations based on our sample size for this specific clique are shown in Table 2.

Table 2: Predicted User Recommendations

	JOE	**DAN**	**DAVE**	**JOHN**
Primary Recommendation	Dave	Joe	Joe	Dave
Secondary Recommendation	Dan	Dave	Dan	Joe
Tertiary Recommendation	John	John	John	Dan

3.2.2 Song Recommendation Results

Specific songs to recommend are purposely left out of these results. There is not enough data to firmly recommend specific songs to each user, even within the clique. If this were to be done, the user's libraries that are recommended to follow would be factored with an artist matrix. For each song in the recommended library that is not included in (1), the union between two user's song vector, the artist of the song would be factored between the two matrices. If the artist name is contained in both, but the song is not, then the spot in the artist matrix, A_s, defined in 2.5.2 will contain an initial base value, say $r_0 > 0$. If several of the recommended users to follow also have this song, the value will be multiplied based on the amount of appearances in the recommended users' libraries. If there is no cross-reference, then A_s will be $\emptyset$. Once completed, each song that has a value will be added to a vector q_s which will be multiplied with the vector p_u, which stands for the user u in the clique vector formed by all of the songs of each member in the clique. Using (2), each song is rewarded a score $r > 0$. The songs with the highest score would in turn be recommended to the given user being observed.

3.3 Conclusions

Although our results can only be translated directly into information for a few of the users surveyed, this can all be translated to a larger scale. Each user displayed in Figure 1 was added to a clique. These cliques have as few as two users, and as many as six. The reason that our results cannot be confirmed for those not included in Figure 3 is due to the small sample size of the survey in general. As discussed in section 3.2, there is great importance to the edge weight between the users, as it shows how many songs they have in common. The inclusion value used to eliminate users from the study can and should be adapted based on the amount of users. If we were to only study five users data, it is likely our inclusion value would have been very low, likely causing our data to be skewed, with no real conclusions. Each user would probably be recommended to each other due to the small amount of users (this would be similar having one clique for all users). Our data becomes more viable with the more users; the more users surveyed, the greater our library of songs, and the stronger each edge between nodes becomes. We could continue to raise our inclusion value, making each clique and connection more exclusive, resulting in better recommendations all around.

It should be noted that as the library of users and songs grows, the recommended users found from Section 3.2.1 would very likely change. Potentially, we could even see entirely different cliques forming, if a user has more in common with other users that are not currently surveyed, which is highly likely. The user recommendation in Table 2 would likely not hold, as stronger connections would grow elsewhere.It can be concluded that such a small sample size was never going to create a recommendation with full confidence, yet the algorithm still works with some success. Nonetheless, this is not a bad thing. Many inferences can be made with strong certainty based on the results. The larger the sample size, the better the recommendations this algorithm would secure, and confidence in the predicted user recommendations would continue to grow. Cliques would continue to be added, and the users that were eliminated in this study when the inclusion value was raised would likely be added to newly formed cliques. All of the users displayed in Figure 1 would have different cliques, and the network would grow and begin to look entirely different. In fact, if the sample size was raised to 100 college students, we would likely see an exponential raise in connections: primary, secondary, and tertiary. The inclusion value could be raised by hundreds, and the cliques studied would show stronger similarities between the users' libraries.

These results can be looked at as a microcosm of college students in general. Not just one type of student was surveyed, and no generalizations were made about the type of music that each student listens to. The small sample size can be compared to that of a small college or even a high school. The small amount of students would encourage the formation of cliques between students who have similar tastes in extracurricular activities, courses, music, etc., just as the cliques were formed in the analyzed network. If the sample size is raised, even further analysis could be done. Let's compare this larger sample size to a big school. It is unlikely that small cliques would be formed, but there still may be some that exist. In fact, it is much more likely that larger student communities would be formed, as there are more people with more connections than would be observed at the smaller school. These communities could be studied for the centrality of the users contained (similar to the warm colors discussed in section 3.1.1) as a method of determining who has the broadest/narrowest taste in music, or even to create further user recommendations past the tertiary level. Even though generalizations were able to be made with the sample size of 27 students, a larger amount of students included in the research would result in the potential for better, more accurate, and more general results.

4 BIBLIOGRAPHY

References

[1] Aiolli, Fabio. Efficient top-n recommendation for very large scale binary rated datasets, Proceedings of the 7th ACM conference on Recommender systems, October 12-16, 2013, Hong Kong, China

[2] Bell, Robert. Volinsky, Chris. Yehuda, Koren, Matrix Factorization Techniques for Recommender Systems, IEE Computer Society, August 2009.

[3] Thierry Bertin-Mahieux, Brian McFee, Daniel P.W. Ellis , Gert R.G. Lanckriet, The million song dataset challenge, Proceedings of the 21st international conference companion on World Wide Web, April 16-20, 2012, Lyon, France

[4] Ke Ji , Runyuan Sun , Wenhao Shu , Xiang Li, Next-song recommendation with temporal dynamics, Knowledge-Based Systems, v.88 n.C, p.134-143, November 2015

[5] Sergio Oramas , Oriol Nieto , Mohamed Sordo , Xavier Serra, A Deep Multimodal Approach for Cold-start Music Recommendation, Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems, August 27-27, 2017, Como, Italy

[6] Schedl, Markus. "Deep Learning in Music Recommendation Systems." Front. Appl. Math. Stat. (2019).

[7] Xinxi Wang , Ye Wang, Improving Content-based and Hybrid Music Recommendation using Deep Learning, Proceedings of the ACM International Conference on Multimedia, November 03-07, 2014, Orlando, Florida, USA