Project Number: MQP-KL2-ABRV

# SONG RECOMMENDATION FOR AUTOMATIC PLAYLIST CONTINUATION

A Major Qualifying Report
Submitted to the Faculty
of
Worcester Polytechnic Institute

In partial fulfilment of the the requirements
for the Bachelor of Science Degree

by

Quinn Averill
Jackson Baker
Samuel Coache
Steven McAteer

Professor Kyumin Lee, Project Advisor
Professor Xiangnan Kong, Project Advisor

# Acknowledgements

We want to acknowledge the help of our two Major Qualifying Project Advisors, Dr. Xiangnan Kong, and Dr. Kyumin Lee without whose guidance this project would not have come to fruition with ease.

# Abstract

The goal of this project is to develop a recommender system that derives song recommendations from an implicit music dataset provided by the streaming service Spotify. We implemented current baseline systems and then two advancements over the baselines: Feature Enhanced Matrix Factorization and Non-Linear Matrix Factorization. To compare these systems, we took the predicted songs for a given playlist and calculated the performance score based on the accuracy of those results. We then compared the results from these NDCG scores to determine which system performed the best for the given Spotify dataset. Based off of the results, we were able to draw conclusions regarding the design process for an effective recommender system for music data.

# Table of Contents

# 1. Introduction

There is no doubt that music streaming services have forever changed the way we listen to and enjoy music. The days of purchasing physical copies of albums are dead, as the general public now has access to millions of songs via music streaming platforms like Spotify and LastFM. As streaming services become more prevalent and competitive, it is essential for them to cater to their customers by addressing one of the most widely criticized drawbacks of music recommender systems; that is creating a system that accurately allows for the user to find new music to enjoy. Streaming giants have detailed recommender systems ingrained into their applications dedicated to recommending new music to users - either when their playlist has finished (playlist continuation) or directly via weekly-generated playlists based on user's listening histories. While streaming services continue to be on the cutting-edge with regards to recommending new music to their customers, these services still have to focus on one of the significant disadvantages of having such an immense catalog of music: the long tail of less-popular albums available to users that go unlistened.

Music stores have limited shelf space, so they generally choose to carry products that guarantee positive revenue - which tends to be the highly popular albums. Because they are not limited on space, streaming services can hold entire discographies of artists that upload their music to their platforms. Below is a diagram showing the range of products and their popularity that the music industry may contain:

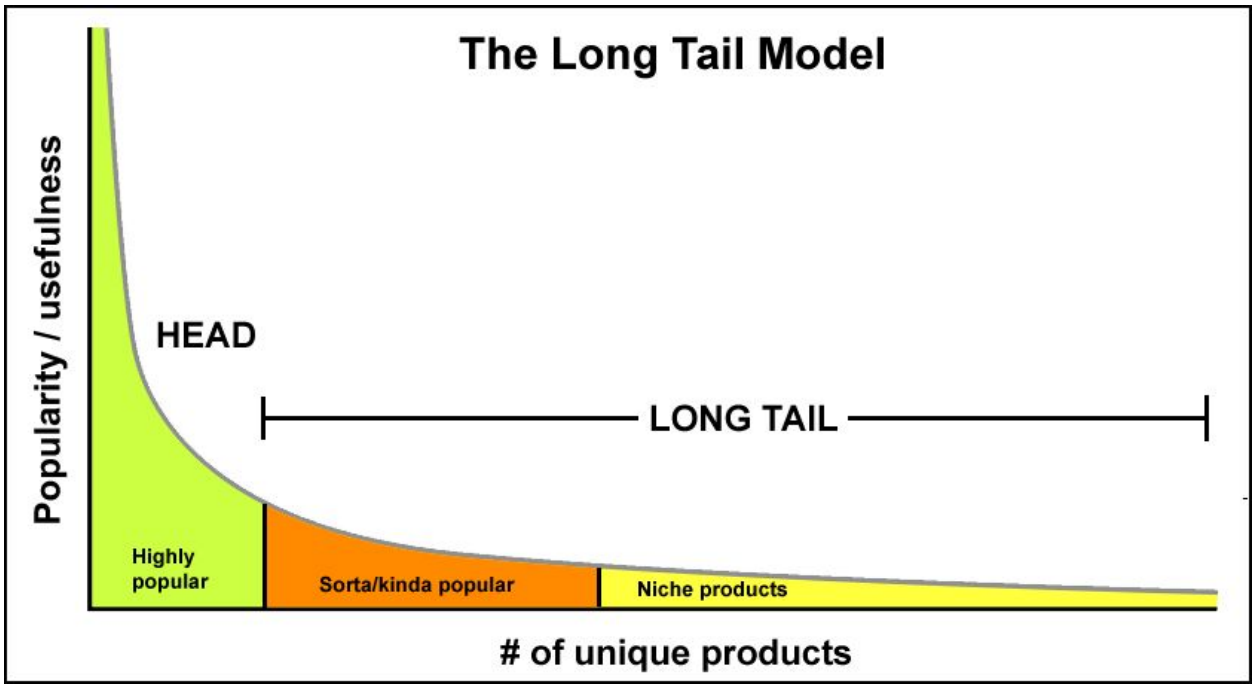


**Figure 1**: The long tail

If you wanted to discover new music in a store that had shelves to hold all of Spotify's available albums, how would you go about finding your new favorite song? A music store with limited shelf space carrying all of the highly popular albums (the head - *Figure 1*) would make it much easier to discover new music due to the smaller selection, coupled with the fact that the selection is entirely made up of successful artists. However, where does this leave the niche and somewhat-popular albums depicted in *Figure 1* above?

As technology rapidly advances, large music streaming companies have the ability and obligation to personalize the user's listening experience. One of the principal methods used for music-based personalization is accomplished by scraping large heaps of user data to identify trends in listening histories. Services like Spotify have a large

enough user base to implement these such tactics. They can use these massive data scrapes to recommend music to each user by examining the trends and similarities in their music tastes compared to other users with similar listening histories.

Recommender systems are a widely used personalization feature on media platforms. The purpose of a recommender system is to “turn data on users and their preferences into predictions of users’ possible future likes and interests.” (Lü et. al. 2009). One of the most fundamental approaches of a recommender is to find objects that other users have shown interest in that the target user may also like. The method of determining the similarity between items varies from system to system, but the underlying foundation stays relatively the same (Lü et. al. 2009).

Many companies have adapted their business models to use recommender systems too. For example, companies like Hulu and Netflix have implemented a “recommended for you” section, which contains shows or movies for the specified user based on the results of their internal recommender systems. By doing this, users will most likely find things that are more appealing to them, and watch shows they would not watch otherwise. By always having an algorithm determine new content for the user to enjoy, the platform becomes more appealing. In addition to that, it helps companies stay relevant to their users and engage their interests over time.

When it comes to music streaming services, finding new content is crucial for users. Many listeners do not know what music they precisely want to listen to and rely on radio-like services to pick songs for them. Spotify does this through “Daily Mixes”,

and a weekly playlist called “Discover Weekly.” These curated playlists are created based on the comparison between multiple user’s listening habits and saved songs.

Spotify also uses recommender systems to create continuous playback once a playlist ends. After all the songs in a playlist have played, Spotify continues to play songs that it thinks are similar to what was previously played in that playlist.

Recommender systems are entirely changing the music industry in multiple ways. Listeners can use them to refine their music taste by continually discovering new music. New and upcoming artists have a much better chance to be heard and discovered. Anyone can create music and put it on a streaming service, and from there, anyone can discover it. Listeners are moving further down the long tail of the niche music genres, which is creating a push on artists to develop newer sounds and make the music industry a more eclectic market.

As detailed recommender systems become a mainstay of streaming services, it is essential that these services understand which system works best for their specific problem. Each streaming service has drastically different user bases, music inventories, and available features to enhance the precision of their recommendations. These services can only utilize the data they have available and design a system based around these factors. With multiple state-of-the-art approaches to recommending implicit music data, it can be a challenge to understand which base recommendation system makes the most sense for a given service. In general, the best recommendation systems must be effective in their recommendation, but also easily implementable and scalable (Falkman, 2016). With recommender systems becoming such an integral part

of the success of streaming services, it is paramount that the systems are well-designed and perform speedily and accurately. We first analyzed the current state of recommender algorithms that utilize implicit music data; we then implemented our system with a feature-enhanced recommender algorithm; finally, we designed evaluation metrics to determine the strengths and weaknesses of each system concerning the accuracy and overall performance of the recommendation.

**The contributions of this work are summarized as follows:**

- We designed two new recommender systems: Feature-Enhanced Matrix Factorization and Non-Linear Matrix Factorization.
- To test the scalability of each model, we used two different datasets containing real user-created Spotify playlists: one contained 100 playlists, while the other contained 1000.
- We evaluated the performance of our two advanced systems compared to the baseline systems via a universal test that calculated the accuracy of the predicted songs using the NDCG evaluation metric.
- To make the recommendation process as transparent as possible, we created a web application that acts as a demonstration of how our systems predict songs for a given playlist. The application is meant to concisely highlight the inner workings of a recommender model and display it in an easy-to-understand manner.

# 2. Related Work

It is essential to understand that when designing recommender systems, there is no designated ‘best’ approach to take. The most optimal recommender systems are those that are simple and transparent, and that understand and utilize the available data throughout the decision-making process while working towards the sought-after solution (Falkman, 2016). The best-designed recommender systems are all created with a unique solution in mind for a given problem; however, there are still common methods used as baselines for a variety of recommendation techniques. These methods differ substantially based on the available data and the goal of the system, and are generally modified as necessary to optimize recommendation. Below we will compare three of the most common baseline systems for constructing a recommendation and discuss the strengths and limitations of each system with regards to how they handle the challenges that music data presents. Additionally, we will highlight state-of-the-art systems designed for music recommendation.

The current baseline recommendation systems are collaborative filtering, content-based filtering, and matrix factorization. While all three approaches have certain advantages and disadvantages based on the data available, ultimately the dataset determines the viability of each respective approach. Collaborative filtering focuses on analyzing the similarities between users or items via data collected by a user’s behavior and preferences (Hu, et al. 2008). Comparatively, content-based approaches compare the description of an item versus a user’s preferences to find similarities to utilize in the

recommendation. By treating each playlist as a User, and each track as an Item, collaborative filtering was the best option for our system due to our lack of valuable metadata on our users. Content-based filtering is an advantageous approach for recommendation systems but tends to have difficulty making reasonable recommendations without explicit data pulled directly from users. Matrix factorization is a bit more confusing, as rather than merely treating playlists and tracks as users and items and analyzing the trends and similarities found from the data as seen in the collaborative filtering methods, matrix factorization relies heavily on calculus and linear algebra to identify hidden latent features (trends) within the dataset.

It is important to note that the results of the collaborative filtering recommender systems rely entirely on the data given to them. If the data contains a considerable number of users (in our case, playlists) with random items (songs of random genres in them), then the recommendations won't make sense, and the algorithm of the system is rendered useless. It is imperative that the systems are given accurate data, with as many item crossovers as possible in order for the system to perform its best. While the intake of inaccurate data does not cripple matrix factorization, it impacts the latent features identified because they are assigned no weight due to the inherent randomness of the input data.

To create a baseline of results for our further experimentation, we implemented the current most popular recommended systems of collaborative filtering: user based, item based, and matrix factorization.

## 2.1 Baseline Systems

### 2.1a User-Based

The first example of a functional memory-based collaborative filtering recommendation system can be accredited to Resnick (Resnick, et al. 1994), who modeled a user *u*'s preferences for an item *i* using a rating matrix *r(u,i)*. Resnick formulated that user's preferences do not change over time and that they should be expected to behave similarly throughout their lifetime (Resnick, 1994). With this in mind, Resnick created a system that would group users who had similar preferences based on explicit historical data to recommend products to these users in hopes that they would enjoy what each other already were known to like. He formulated the following equation, which results in a rating prediction $\tilde{r}$ (*u,i)* given a user and item.

To make the rating prediction, the system takes the sum of the average ratings across the rating matrix $\tilde{r}(u)$, a normalization factor $C_0$, and the sum of the similarity values generated using the preferences given users have for the item in question. Here, $N_k(u, i)$ represents the set of *k* most similar users to the input user *u*. This set of similar users changes in size based on the size of the dataset and is referred to as the "neighborhood". The similarity between the input user *u* and one of the users from the neighborhood *v* is performed in the function *sim(u, v).* Finally, the rating prediction is formulated and the system can determine whether or not the input user would enjoy the item in question.

User-based recommendation is a useful baseline technique, but relies heavily on a rating matrix - unfortunately, the user cannot “rate” music on streaming services, which makes incorporating collaborative filtering techniques in music recommendation so tricky. That being said, by making playlists act as users, we can create a rating matrix based on the existence of unique songs within playlists. Rather than comparing the similarities of user’s ratings of given items, we focus on whether or not similar playlists contain the same songs. The input playlist (playlist being recommended songs to) is matched to playlists that are the most similar to it, and then the tracks that appear most frequently in those playlists are recommended to the input playlist. The system fundamentally recommends items to users, based on comparisons made between their preference in items, and other user’s preferences.

### 2.1b Item Based

An item-based recommendation is similar to user-based collaborative filtering in that it focuses heavily on identifying similarities within the dataset; however, item-based recommendation looks for items that have been similarly rated by users who have used both items. If it is apparent that two items a and b have been similarly rated across the database of ratings, then it can be assumed that users who like the item are also inclined to like item b and vice-versa. Item-based collaborative filtering addresses one of the issues with user-based collaborative filtering - while the user-based system may be logically straightforward, it can become very expensive when dealing with more massive datasets (Falkman, 2016). By focusing on items, which generally appear in significantly smaller numbers compared to users in an item/user rating matrix, item-based

collaborative filtering performs more cost effective when compared to user-based collaborative filtering on large datasets. Understandably, item-based collaborative filtering seems like a viable system when dealing with datasets of music containing hundreds of thousands of unique songs. Unfortunately, due to the sheer mass of items compared to users within music datasets the performance strengths of item-based collaborative filtering are lost. One playlist may contain anywhere from twenty to over a hundred songs, and with a limited number of playlists compared to songs, it is much harder to isolate songs that appear across the database to find similarities in tracks.

Item-based collaborative filtering relies heavily on the number of times a song appears across the dataset. Since there are no ratings for music data, just whether or not a song exists in the playlist or not, the item-based collaborative filtering system designates those songs that appear in many playlists across the dataset as more valuable, in this way acting as the basis for the item-based collaborative filtering process. Firstly, every song in the input playlist (the given playlist to recommend songs to) is matched to songs that are similar to it from the dataset of unique songs, recording which is identified by the system as “similar tracks.” From there the input playlist takes in the tracks that appear most frequently within the “similar tracks” matrix. It takes a user’s preference in items and then recommends items to them based on other items that are similar to their preference.

## 2.1c Matrix Factorization

Matrix factorization is significantly different than the other two collaborative filtering systems. To determine what songs the system recommends for the input playlist, matrix factorization uses calculus and linear algebra. It determines hidden features for each song and playlist and determines each track's importance in a playlist based on those features.

Matrix factorization utilizes two hidden feature matrices: for users and items. The calculated dot product of these two matrices creates a predicted rating matrix and the loss, or error, of each prediction, is calculated. The algorithm uses an optimizer to minimize the loss, so that the loss is as close to zero as possible. The loss of each prediction is calculated again for both matrices the same way as before, but this time, they are different because its gradient changed it; this updates the loss value for each iteration the algorithm loops through.

This process is explained more in *Section 3.2c Matrix Factorization*, but for now, know that it is an approach that decomposes the user-item matrix into two lower dimensionality rectangular matrices that when combined, create a mathematical approximation of the original matrix with previously empty spots in the playlist filled with songs that it thinks should be recommended.

## 2.2 State of the Art Systems

To get further insight into how to improve the baseline recommender system, our team researched current state-of-the-art systems in hopes of improving our recommendations. Generally, the state-of-the-art recommender systems for music data build off of the baseline systems mentioned above, adding layers of logic and enhancements based off of their unique dataset to improve recommendation scores. Recently, matrix factorization has become the standard algorithm for current recommender systems, as it consistently outperforms other systems when utilizing explicit music data and can easily be enhanced via implementing different combinations of the loss function and learning rates (Bodke et. al. 2015, Chen et. al. 2015). That being said, user-based and item-based systems are still viable when it comes to simpler recommender systems utilizing implicit datasets.

Simplicity is essential when designing a high-performance music recommender system, and many of the state-of-the-art approaches attempt to improve upon matrix factorization by making small changes to the base system and measuring the change in performance. Scientist Ishteva Kannan implemented a bounded matrix factorization system that imposes a lower and upper bound on every estimated rating. Adding these bounds to the songs score significantly improved recommendation accuracy when compared to using a binary rating matrix (wherein the song exists in the playlist or not), because of this addition songs could be weighted more heavily making it more apparent which songs were the most important to the playlist. The results of the experiment

proved that the proposed method outperformed the regular matrix factorization system. (Kannan et. al. 2014).

With regards to the quality of recommendation, deep learning has the potential to revolutionize the user experience for better customer satisfaction. Deep learning, unlike the baseline systems mentioned above, can accurately interpret and incorporate user intent when making predictions (Mu, 2018). Scientists at the University of Beijing implemented an item-based deep network structure for collaborative filtering, which aimed to identify hidden latent features within the dataset by utilizing a batch gradient descent on the rating matrix (Yong-Ping et. al. 2017). The hidden latent features that were identified due to the deep network structure positively affected the recommendation. Although the study was performed using explicit movie data, the inclusion of a gradient descent to identify a feature vector would be useful for implicit music data. The system outperformed the standard collaborative filtering algorithm, which was the comparison system (Yong-Ping et. al. 2017).

One of the most substantial issues with collaborative filtering is the sparsity of the initial rating matrix (Wang, et al., 2006). When there is not enough information on each user, the system lacks the insight that requires it to make accurate predictions. One way to solve this issue is by clustering - rather than using the totality of the rating matrix when calculating recommendation scores, the matrix would begin with only those items and users that possess a certain level of similarity to the input user or item (Frémal & Lecron, 2017). This method of clustering items and users is extremely effective, as it eliminates the need for countless comparisons between the input playlist and playlists

that contain no worthwhile tracks. Another method of solving the sparsity problem was implemented via a model that uses a user-tag matrix which represents the user's preference on each tag. This is then used to make the user-item matrix denser, allowing the system to gather additional information on each user when predicting the ratings. Experiments found that this method generates more precise prediction than general matrix factorization that suffers from the sparsity problem (Bu Sung Kim et. al., 2014).

When designing our system, we drew inspiration from the state-of-the-art approaches that tackle tweaking the standard matrix factorization system to produce better results. For example, in Kannan's paper on bounded matrix factorization, he emphasizes the importance of using a baseline system and changing it to include and upper and lower bound on every estimated rating. Although this change was not difficult to implement, it made considerable improvements to the standard performance of the baseline matrix factorization system. Throughout the systems mentioned above, it is evident that simplicity is key when designing recommendation systems. When designing our advanced systems, we kept this ideology in mind and subsequently made smaller changes to the baseline matrix factorization system rather than building our own system.

# 3. Our Approaches

## 3.1 Data

The data used in our research comes from a large dataset that Spotify released for their Million Playlist Dataset challenge in 2018. We obtained three slices of 10,000 playlists, which when broken down contained well over 300,000 unique tracks. We chose to use a set of 3000 playlists taken from our available dataset for our experiments. These playlists contained around 100,000 unique songs. The data is comprised of playlists that have been created by real Spotify users, so it contains 100% accurate real-world data (Chen, 2018). To persist our datasets, we utilized MongoDB Atlas's free non-relational database services. Each playlist's tracks were saved, along with basic information on each track such as title, artist, and album. Each track contains a unique Spotify ID which allowed us to append additional information to each song using the Spotify API. *Section 3.3a* goes into further detail regarding how this additional information was used.

### 3.1a Challenge

In recommender systems, the data used for the recommendations can be either explicit or implicit. Explicit data contains information deliberately entered by the user, such as a rating that they give the movie that they just watched. Implicit data is what is inferred by the user's interaction with the computer interface, such as how much of a

movie the user watched. Explicit data carries more useful information than implicit with regards to a recommender system, since the data comes directly from the user and tells us what they do and don't like. Since our experiment uses playlists that contains songs, our data is implicit with the playlist being the user, and the song being the item. Without this explicit data, we cannot answer the question *why* certain songs were included in playlists, or if the playlist favors one song over another, since each song carries the same weight by simply being included in the playlist - therefore all songs are treated equally (Schedl et. al. 2018). This inherent problem with implicit music data is one of the reasons well-designed recommender systems are so valuable to music streaming giants (Najafabadi et. al. 2017).

### 3.1b Square Dataset Smoothing

In order to keep the testing datasets easy to work with, we smoothed the data from Spotify's MPD into square datasets. Each dataset contains an equal number of playlists and tracks, making each of them extremely dense. For example, one of the datasets used for quick results was a 100 playlist by 100 track testing set; this made computation faster and also created less sparse matrices.

To acquire these dense, square datasets, we found the most popular songs in the entire dataset. Then, we sorted the playlists based on the number of popular songs that each one had. We took the top N playlists from this sorting and removed any songs in each playlist that were not in the N most popular songs. After the selection it has been narrowed down to N playlists containing only the top N most popular songs, therefore creating an N by N matrix that is as dense as possible.

We used two different values for N: 100 and 1000. The matrix containing 100 playlists by 100 tracks had a sparsity value of 0.367, and the matrix with 1000 playlist by 1000 tracks had a sparsity of 0.076. We tested all of our systems on both of these datasets and reported all results in *Section 4*. Results.

## 3.2 Baseline Techniques

The following section outlines the three different recommendation techniques our group implemented as a baseline to compare results to: item-based recommendation, user-based recommendation, and matrix factorization. The goal of each method is to recommend a list of K songs (where K is the number of songs to recommend) that the user may want to add to a given input playlist. In our programming, our “users” are Spotify playlists and our “items” are Spotify tracks, or songs.

The systems are initially given a matrix that contains a row for each playlist, and a column for each track. Each cell in the matrix contains a value of “1” if the playlist index contains the track index and a value of “0” if not; this matrix is referred to as the rating matrix. Each system uses this rating matrix to compare the users and items to a given playlist and return a list of similar tracks to it, ordered by greatest similarity to least. Tracks are then taken from these lists and are recommended to the given playlist, which is detailed more in *Section 4.1*.

### 3.2a User Based Collaborative Filtering

Our first approach implemented was user-based collaborative filtering. This method consists of comparing characteristics between playlists and tracks according to what tracks are placed in each playlist. For each input playlist, the algorithm creates a list of playlists that are similar to it using cosine similarity. It then takes the top N similar playlists, and creates a sum of how many times each track appears in those N playlists, and multiplies each count by its playlist similarity score as a weight. The algorithm lastly returns a list of these tracks ordered from most to least similar according to their sums. Determining the value of N is accomplished by trying it with each value between 1, and however many playlists are in the dataset - 1. The best value is then utilized for the recommendation algorithm according to which dataset it is being run against.

### 3.2b Item Based Collaborative Filtering

In this approach, it is the same process as user-based collaborative filtering, with the only difference being that each track is compared to one another instead of users. To make implementation of Item-Based Collaborative Filtering more efficient, the input rating matrix is transposed before the similarity computation in turn reducing computational overhead when comparing tracks.

For each input track, the algorithm first creates a list of tracks that are similar to it using cosine similarity. It then takes all the tracks in the input playlist and creates a sum for each of the top N tracks in its list of most similar tracks multiplied by its similarity as a weight. Finally, the algorithm returns the list of these tracks ordered by their sums from

greatest to least. The value for N was found the same way as in *Section 3.2a User Based Collaborative Filtering* above.

## 3.2c Matrix Factorization

Matrix factorization factorizes the rating matrix into two hidden feature matrices: one for the users and one for the items. The user hidden feature matrix contains a row for each user and a column for each hidden feature. The item hidden feature matrix is structured the same way, but a row for each item. For the remainder of this section, the user hidden feature matrix is referred to as U, and the item hidden feature matrix as V.

The algorithm starts by first initializing U and V to contain random decimals between 0 and 1. The initial rating matrix given to the system is also multiplied by a constant alpha, allowing more variations to be made when predicting the rating for a user/item combination. Then, for a defined number of times (steps), the following process is repeated (this process is also known as Alternating Least Squares (Yehuda, 2009)):

1. The dot product of U and V is calculated, creating the predicted ratings matrix.
2. The loss (error) of each prediction is found by taking the squared difference between the user/item combo in the ratings matrix from the corresponding combo in the predicted ratings matrix. The formula is below:

$$e_{ui} = (ratingsMatrix_{ui} - U_u^T V_i)^2$$

3. To minimize this loss, the algorithm uses an optimizer so that the loss is as close to 0 as possible. It does this by calculating the gradient (derivative) of the error

function above, and then taking a small step, □ (learning rate), in the opposite direction of the gradient. The system loops over each vector in the item hidden feature matrix and updates the row using a method known as Stochastic Gradient Descent (Artificial Intelligence - All in One, 2016). Below is the formula for each update:

$$V_i = V_i + 2\gamma * e_i U$$

4. The loss of each prediction is calculated again the same way as in step 2, but this time, V is different because it was changed by it's gradient; this updates the loss.
5. For each user in U, it's gradient is calculated just like in step 3:

$$U_u = U_u + 2\gamma * e_u V$$

These five steps repeat until the gradient of the loss is low enough to yield good recommendations. The algorithm knows to stop after a set parameter called steps, which is found by trial and error. If it is too large, then the model will be overfitted, and the predictions will match the actual ratings too well. If the algorithm is not stepped through enough, then the model is under fitted and predictions are accurate. It is imperative that this constant is at the optimal spot so that the predictions are close to the actual ratings, but not spot on to the point where there are predictions that do not exist in the original rating matrix.

In the equations above, there is a value □. This is known as the learning rate, and is what nudges the value along the gradient to bring it closer to zero, or the actual value. The values for the number of hidden features, the number of steps, alpha, and

the learning rate are found by testing a range of values for each parameter, and finding what combination of all four works the best.

For a given playlist that is being recommended to, the dot product of U and V (after all the steps have run through) is calculated, and the playlist's row of tracks are then ordered from greatest to least, where the value in the cell indicates how well the track fits the given input playlist. Finally, the program returns the list, and the top K songs are then recommended for the playlist.

## 3.3 Baseline Advancements

For the contribution to the current research in the field of recommender systems, the team implemented two alternative systems in an attempt to yield better results. In doing so, the inner workings of matrix factorization dissected, and the team gained a better understanding of what happens inside the process. The first system implemented is called Feature Based Matrix Factorization, and the second is called Non-Linear Matrix Factorization; outlined below.

### 3.3a Feature Enhanced Matrix Factorization

This method uses the standard matrix factorization as a base, and appends three Spotify API features to each item vector. For each step in the matrix factorization process, the last three latent features in the item to feature matrix are kept constant since they are known features of the track (this would be completed right after the third step in *Section 3.2c Matrix Factorization*). This allows matrix factorization to continue its normal procedure, while adding known information for a given track to the calculation. A

constant, C, was multiplied by each Spotify feature in the item hidden features matrix to make them have a prominent role when predicting songs to recommend. The anticipation is that these three features pulled from Spotify's API will assist the normal matrix factorization and increase the performance.

The three features added were all normalized to be values between 0 and 1. The first was Valence, which determines the overall positiveness of the song, with 0 being not positive, and 1 being extremely positive. The second was danceability, which determines if the song is easy to dance to or not. The last was energy, which tells the intensity and activity of the song. (Spotify, 2018)

### 3.3b Non-Linear Matrix Factorization

This form of matrix factorization uses a nonlinear function to predict if the track would be a good recommendation to the playlist or not. After the dot product of the user hidden features matrix U and the item hidden features matrix V is taken to create the prediction matrix p, the matrix is then normalized across each user using the equation below to make all values between 0 and 1. This equation was taken from the Python library Pytorch (Pytorch, 2018)

$$p_{ui} = \frac{p_{ui}}{max(||p_{ui}||_2, \varepsilon)}$$

Where $p_{ui}$ is the predicted rating for the track in the user row, and $\varepsilon$ is a small value (1e-12) that avoids dividing by zero [WAT]. This normalization bounds the estimates, giving an upper and lower bound that creates a range for the predictions.

Then the sigmoid of each normalized predicted rating was taken to further specify its importance in the user / item combo. The classic sigmoid equation was altered a bit in order to encompass only values between 0 and 1. Here is that equation, along with its graph to visualize what it is doing to the prediction matrix:

$$p_{ui} = \frac{1}{1+e^{(-40p_{ui}+20)}}$$

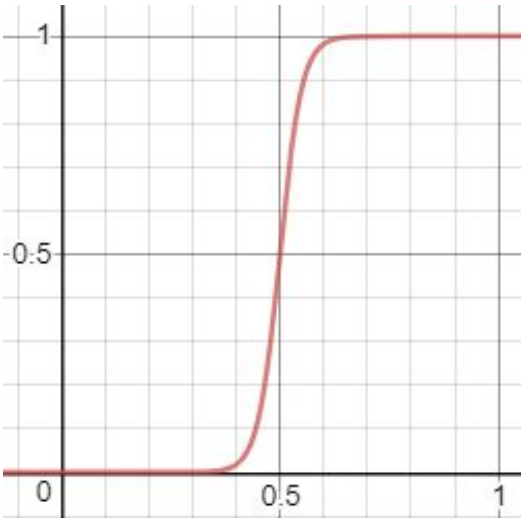


**Figure 2**: Sigmoid graph

This nonlinear approach allows predictions closer to 1 to be greater, where predictions closer to 0 will be lower. This further increases the separation between the 1's and the 0's, allowing the system to solidify what it thinks should be recommended to a playlist and not.

## 3.4 Web Application

To present our system and components as well as demonstrate a recommendation utilizing our design, we designed a web application demonstration to educate a user on the system by displaying an example of a given playlist and the output songs recommended to it. The demonstration was created as a Node.JS web application using core Javascript libraries and with Bootstrap added for styling; the D3.js library was used to create the graphs. To protect the project database instead of the demonstration hosts a small subset of the data required to show our results without exposing the rest of the data to malicious automated enumeration. Therefore, the final demonstration is an isolated demo designed with the purpose of displaying and explaining the process in a streamlined method.

To create the demo the first step was the selection of a highly perfect playlist from the dataset to provide the most visibly exciting results. Once the playlist from the dataset was chosen we performed evaluations with the different implementations of the algorithm using this playlist as the given input. With these results, we then developed a series of graphs to show the correlated data in a visually appealing and interactive way.

The informational component of the demonstration explains the equations used and system in a brief overarching view of the project. The purpose of the demonstration is to allow someone to go from start to finish of the application and see step by step the process for recommending songs with the data displayed in the graphs described in the next paragraph.

The data visualizations components of the demonstration is an NCDG precision graph and two radar graphs. The NDCG precision graph, Figure 3 below, is a line graph displaying the Average NDCG precision of Normal Matrix Factorization, Feature Matrix Factorization, and Nonlinear Matrix Factorization versus K Values; the K value represents the number of songs recommended to the input playlist.

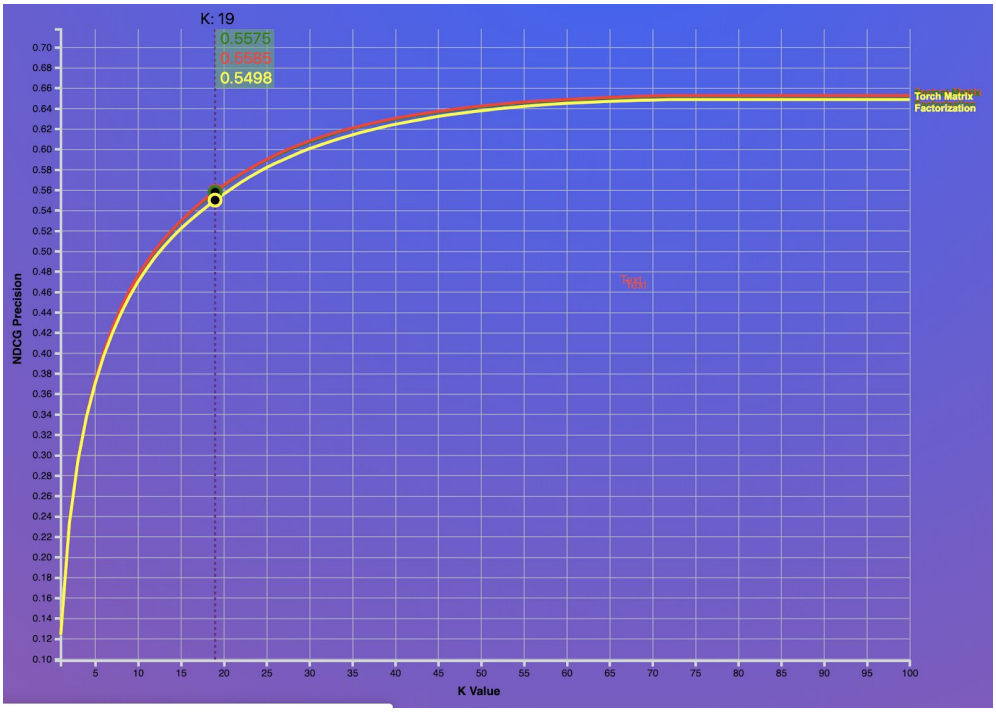


**Figure 3**: NDCG Precision Evaluation Graph

The two radar graphs display similar data but from different sources. Each axis of the chart represents a different feature supplied by the Spotify API for each song; Energy, Acoustic, Instrumental, Dance, Tempo, Valence, Speech, and Loudness. The first graph, Figure 4, displays the average of these features for the complete dataset and the input playlist to see how it compares to the available pool of songs. The second, Figure 5, displays the same average playlist features in comparison to the features of

the ten recommended songs, by selecting a song from the section underneath the values update to see how accurate the song recommendation was.

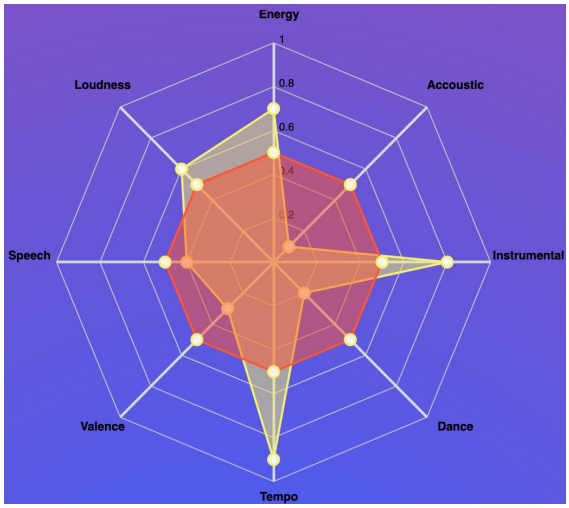


**Figure 4**: Dataset vs. Playlist Feature Radar Graph

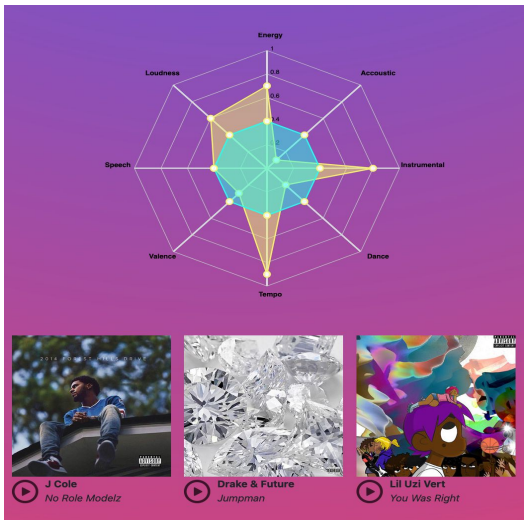


**Figure 5**: Playlist vs. Recommended Songs Feature Radar Graph

# 4. Results

## 4.1 Evaluation Techniques

To find the parameters for each system trial and error was used that lead to the most significant outcome. Every result in this paper uses these parameters in each system, giving us the best case scenario result for each system.

### 4.1a Scoring

One-fifth of the input playlist tracks is removed at the start of the algorithm to be used to evaluate the ability of each recommender system. Before each system runs the algorithm, we record the overlap between the recommended songs and the removed tracks for every playlist in the dataset. Then, after each run terminates, the overlap between the recommended songs and the removed tracks is again recorded. The algorithm then runs, and the system recommends K songs to each input playlist in the dataset, and then uses initially removed fifth of the playlist to evaluate the accuracy these K songs. The average calculated score for each playlist is used to determine the overall score of the recommender system. We sample this average multiple times, and then each run's average is taken to make a more precise total score.

To calculate a score for the system's recommendation the output is measured using Normalized Discounted Cumulative Gain (NDCG) as the evaluation metric. This metric allowed us to see if the songs recommended matched the fifth that was removed,

as well as see if the order of the recommended songs were correct. This evaluation is crucial as it tells us if ranked the most similar songs in the correct order (Busa-Fekete et. al. 2012). It is also normalized, so it always returns a value between 0 and 1.

### 4.1b Presentation

We evaluated the systems using different values of K to see how it affects the results of each system. We tested values of K from 5 to 15, because that is around the average number of songs recommended to a playlist at a time in music recommender systems. Having results for each K value allowed us to graph the score versus K to see how the score of a system changes as more songs are recommended. These charts are in both Sections 4.2 Baseline Results and 4.3 Feature Enhanced Matrix Factorization Results. The Spotify playlist recommendation feature in their desktop app recommends 10 songs at a time, so we mainly used K = 10 when comparing systems using a single value.

### 4.1c Comparisons

To compare each system against the other systems, each ran its algorithm using the same split playlists, allowing us to compare all scores to each other honestly. Because of this, it yields the most accurate results to determine how the systems compare to each other. The rest of the evaluation process continued, and the final NDCG results were compared and graphed against each other.

## 4.2 Baseline Results

After running the three baseline recommender systems against all playlists in the mpd_square_100 dataset 100 times, it was found that the matrix factorization system outperformed both user and item collaborative filtering, with item and user based yielding fairly similar results. To compare the systems numerically, a K value (number of songs recommended) of 10 was used since that is how many songs Spotify recommends to playlist in a given time. Here is a graph showing the results of this test:

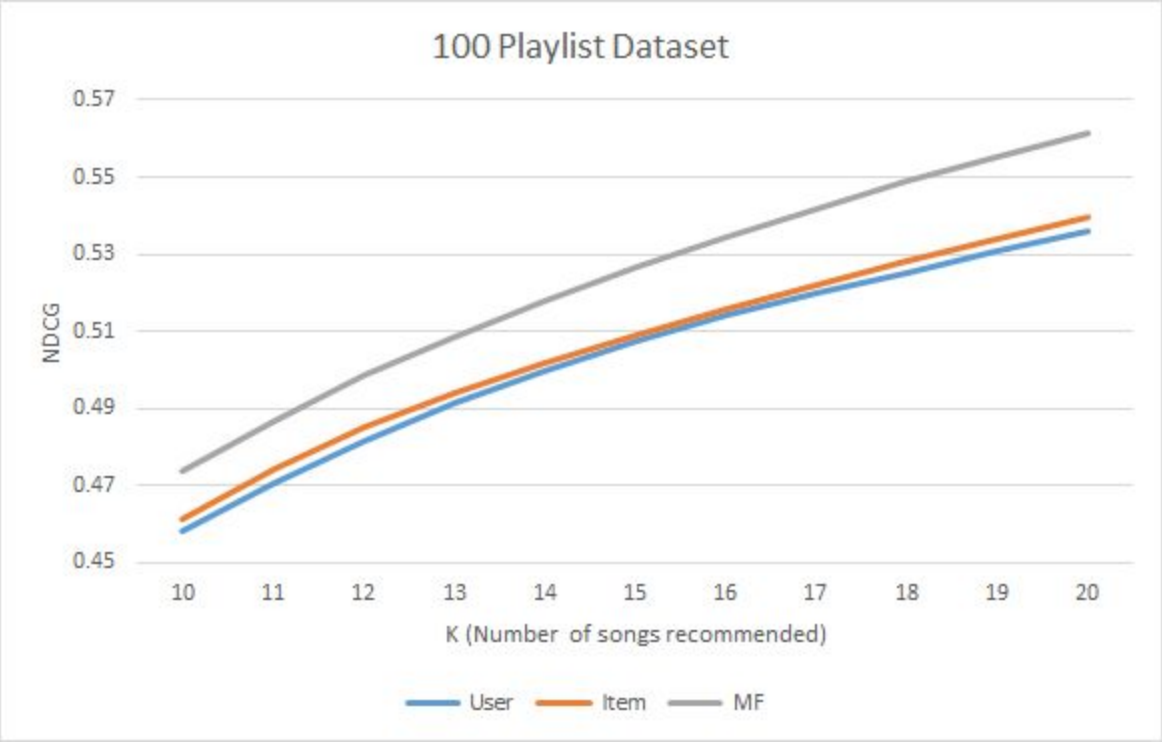


**Figure 6**: 100 Playlist NDCG scores

Matrix factorization scored an average of 0.474 at K = 10, Item based collaborative filtering scored an average of 0.461, and user based scored 0.458. These

results were expected, due to our prior research in the state of the art recommender systems.

After running the three baseline recommender systems against all 1000 playlists in the mpd_square_1000 dataset 100 times, it was again found that matrix factorization had a better average score than both item and user collaborative filtering. Here is the graph showing the results of this test:

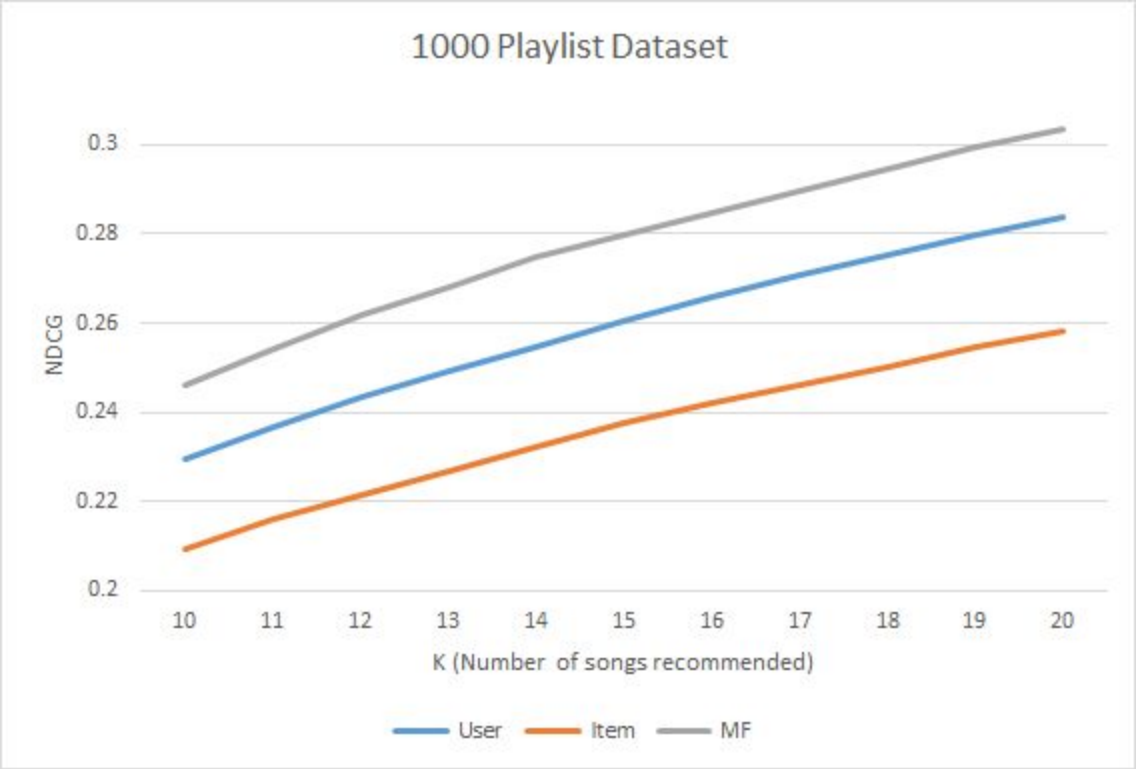


**Figure 7**: 1000 Playlist NDCG scores

Matrix factorization scored an average of 0.246 at K = 10, Item based collaborative filtering scored an average of 0.209, and user based scored 0.229.

Since matrix factorization yielded the greatest results for both datasets, it will be used as the baseline to compare against the baseline advancement systems.

## 4.3 Feature Enhanced Matrix Factorization Results

To evenly compare the results of feature enhanced matrix factorization with the regular matrix factorization, feature enhanced was given the exact same parameters as regular matrix factorization. For the 100 playlist dataset, the number of latent features was 300, and for the 1000 playlist dataset it was 15. It was found that for feature enhanced to achieve the best results, the feature constant C had to make the spotify features weigh equally with the other latent features. If it was increased or decreased to make them more or less important, then the results lowered.

For the 100 playlist dataset, the two systems achieved the exact same scores. This is thought to be because of the high amount of latent features (300) required to make regular matrix factorization yield the highest results. This caused the Spotify features to not make a big enough impact on the system. As Zafari & Moser suggest, “...the sparsity of the user-item ratings matrix makes it difficult to learn the user preferences over feature values” (Zafari & Moser, 2017). Adding features values does not give enough information regarding whether or not a user will like the recommended songs. In our scenario, even when using the best feature constant, C, the Spotify features did nothing to help improve matrix factorization results. There is no need to show a graph for this experiment, for both lines overlap and it shows no added information.

For the 1000 playlist dataset, feature enhanced matrix factorization achieved worse results than the regular matrix factorization. At K = 10, it scored an NDCG of

0.241 (regular matrix factorization scored a 0.246) . Here is a graph showing the comparison:

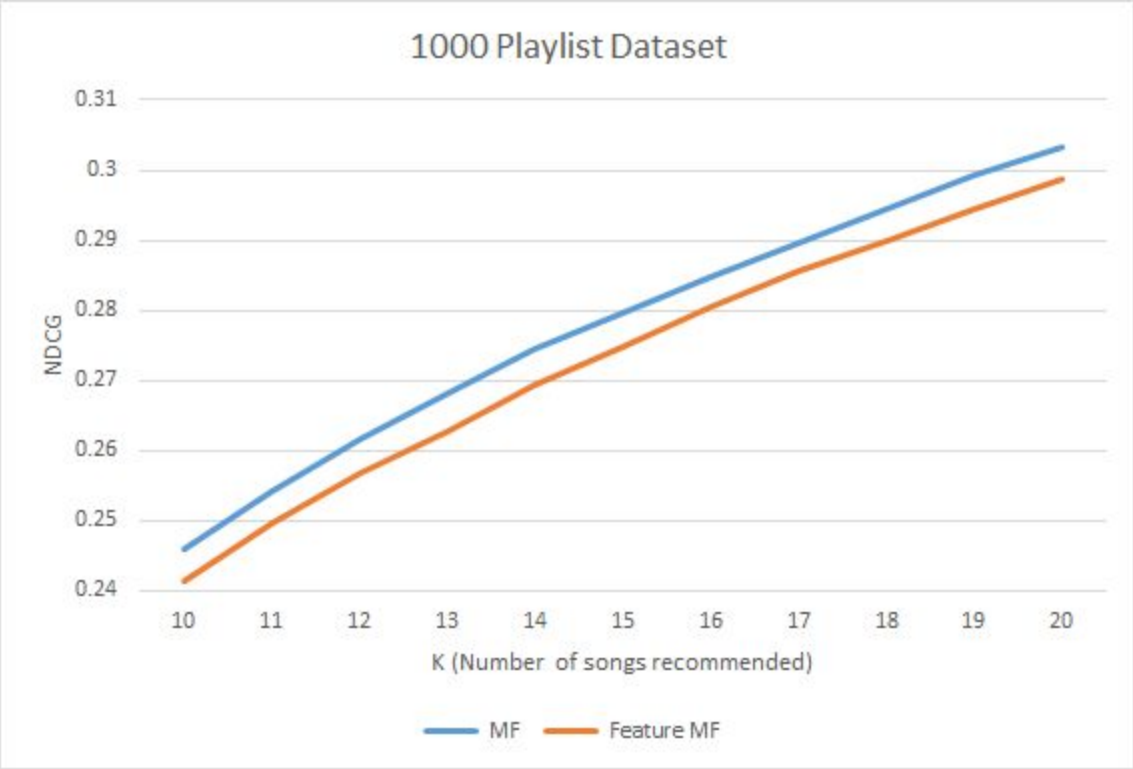


**Figure 8**: 1000 Playlist NDCG scores

Since the number of latent features was only 15, having 3 of those features be the Spotify features caused the average score to lower when compared to normal matrix factorization. This just further shows that the added Spotify features do not provide assistance to the matrix factorization system.

## 4.4 Nonlinear Matrix Factorization Results

For the 100 playlist dataset, the nonlinear matrix factorization did not beat the regular matrix factorization in NDCG scores. At K = 10, it scored an NDCG score of 0.470 (regular matrix factorization scored 0.474). Here is the graph comparing the two systems:

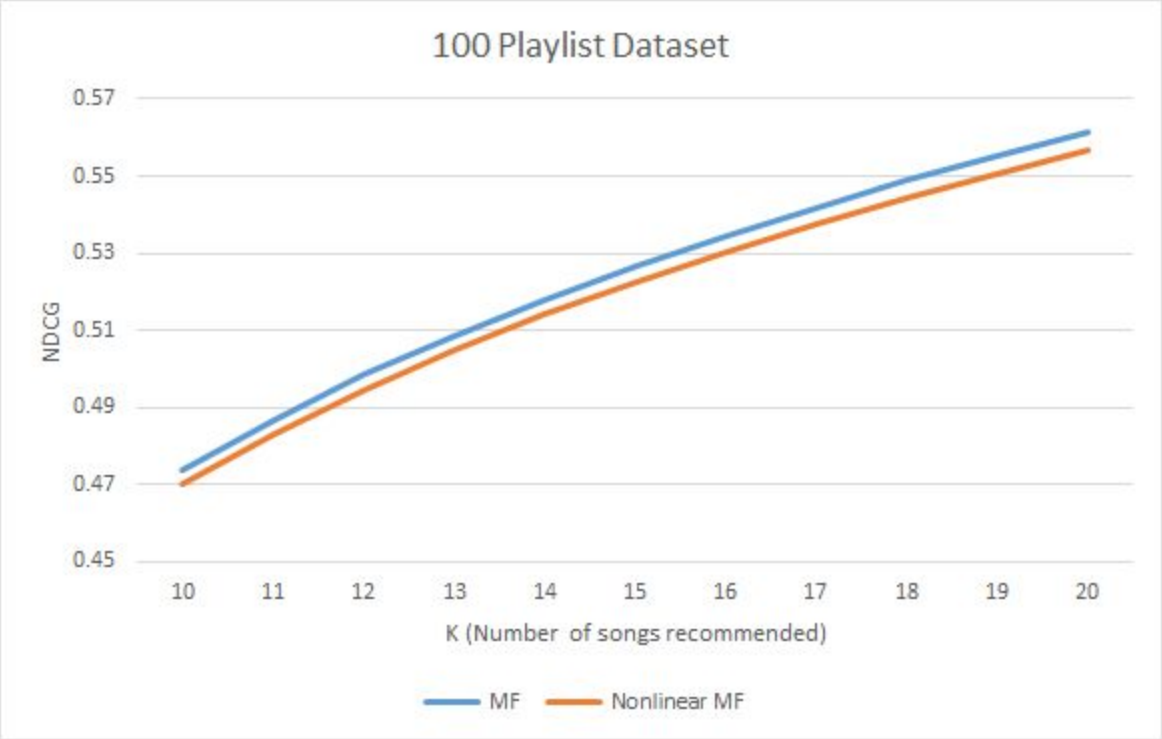


**Figure 9**: 100 Playlist NDCG scores

For the 1000 playlist datasets, the results of the nonlinear matrix factorization were much closer to the regular matrix factorization. At K = 10, it scored a 0.245 (where regular matrix factorization scored 0.246). Here is a graph comparing the systems:


**Figure 10**: 1000 Playlist NDCG scores

At each K value, the results are extremely similar, showing that with more data in the dataset, the nonlinear approach does not change the results by much. Overall it was slightly worse, but not as much as the 100 playlist dataset.

# 5. Discussion

As *Section 4* shows, there are many factors that affect the performance of each system. In Figure 3, the NDCG values show that the user-based and item-based recommender systems clearly trail behind matrix factorization recommender. User and item yield poor performance due to the nature of each algorithm; the former only focuses on the user factors and compares them, while the latter only focuses on the items. The predictions from matrix factorization are more accurate than those from the prior two methods because it utilizes both the user and item factors during its algorithm.

When comparing the 3 different matrix factorization methods from *Section 4,* the results are similar. For matrix factorization and feature-based matrix factorization, the NDCG values are exactly the same. This shows that adding additional static feature values from Spotify's API to each individual track did not improve performance. A weight constant was multiplied to these features to increase their importance over the other hidden features, however the system performed best when this constant treated the known features the exact same as the hidden features. The nonlinear matrix factorization method also did not perform better than the standard matrix factorization method; it yields considerably lower performance when compared to the other two matrix factorization methods.

After implementing these two variations of matrix factorization, we realized that adding more data and changing parts of the standard matrix factorization algorithm did not affect the performance as intended. We learned that the simplicity that matrix

factorization provides makes the prediction results more accurate since there are less complications within the algorithm.

# 6. Conclusion

Although the proposed advanced recommender systems did not improve performance when recommending songs, they both displayed the importance of simplicity within a recommendation algorithm. While the advanced systems we designed utilized more of the available data when calculating their recommendations, ultimately it was clear that when it comes to recommending music from an implicit dataset, simplicity is key. Recommender systems should be as scalable, efficient, and transparent as possible - and due to the inherent difficulties that come with music data, it was apparent that the more we focused on attempting to improve the performance via analyzing individual track features, the more unreliable the recommendations would become. The matrix factorization recommender system consistently produces the best performance with regards to Spotify music data.

# 7. References

*Advanced Intelligent Systems (ISIS)* (pp. 980–984). IEEE. https://doi.org/10.1109/SCIS-ISIS.2014.7044855

Artificial Intelligence - All in One. (2016, April 13). *Finding the Latent Factors | Stanford University* [Video File]. Retrieved from www.youtube.com/watch?v=GGWBMg0i9d4.

Bodke, D., Girase, S., & Mukhopadhyay, D. (2015). Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey, [Abstract]. *Procedia Computer Science*, 49, 136-146. Retrieved March 3, 2019, from https://doi.org/10.1016/j.procs.2015.04.237.

Bu Sung Kim, Heera Kim, Jaedong Lee, & Jee-Hyong Lee. (2014). Improving a recommender system by collective matrix factorization with tag information. In *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on*

Busa-Fekete, R., Szarvas, G., Élteto, T., & Kégl, B. (2012). An apple-to-apple comparison of Learning-to-rank algorithms in terms of Normalized Discounted Cumulative Gain. In ECAI 2012 - 20th European Conference on Artificial Intelligence : Preference Learning: Problems and Applications in AI Workshop (Vol. 242). Ios Press.

Chen, L., Chen, G., & Wang, F. (2015). Recommender systems based on user reviews: the state of the art. User Modeling and User-Adapted Interaction, 25(2), 99–154. https://doi.org/10.1007/s11257-015-9155-5

Chen, C. (2018, June 06). Introducing The Million Playlist Dataset and RecSys Challenge 2018. Retrieved from https://labs.spotify.com/2018/05/30/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/

Falkman, G. (2018). A Scalable Recommender System for Automatic Playlist Continuation Master Degree Project : Report.

Frémal, S., & Lecron, F. (2017). Weighting strategies for a recommender system using item clustering based on genres. *Expert Systems with Applications,77*, 105-113. doi:10.1016/j.eswa.2017.01.031 https://www.sciencedirect.com/science/article/pii/S0957417417300404?via%3Dihub

Kannan, R., Ishteva, M., & Park, H. (2014). Bounded matrix factorization for recommender system. Knowledge and Information Systems, 39(3), 491–511. https://doi.org/10.1007/s10115-013-0710-2

Lü, L., Matúš, M., Chi Ho, Y., Yi-Cheng, Z., Zi-Ke, Z., & Tao, Z. (2009). Recommender Systems. Physics Reports, 519(1), 1-49. Retrieved January 10, 2019

Mu, R. (2018). A Survey of Recommender Systems Based on Deep Learning. *IEEE Access,6*, 69009-69022. doi:10.1109/access.2018.2880197 https://ieeexplore.ieee.org/document/8529185

Najafabadi, M., Mahrin, M., Chuprat, S., & Sarkan, H. (2017). Improving the accuracy of collaborative filtering recommendations using clustering and association rules mining on implicit data. Computers in Human Behavior, 67, 113–128. https://doi.org/10.1016/j.chb.2016.11.010

Pytorch. (n.d.). Torch.nn Documentation. Retrieved November 2, 2018, from https://pytorch.org/docs/stable/nn.html

Resnick, P. (n.d.). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of the 1994 ACM Conference on Computer Supported Systems* (pp. 175-186). doi:10.1145/192844.192905

Schedl, M., Zamani, H., Chen, CW. et al. Int J Multimed Info Retr (2018) 7: 95. https://doi.org/10.1007/s13735-018-0154-2

Spotify. (n.d.). Spotify Web API. Retrieved October 1, 2018, from https://developer.spotify.com/documentation/web-api/

Wang, J., De Vries, A.P., and Reinders, M.J.T. “Unifying User-Based and Item-Based Collaborative Filtering Approaches by Similarity Fusion.” Vol. 2006. N.p., 2006. 501–508. Print.

Yehuda, K. (2009). Matrix Factorization Techniques for Recommender Systems. IEEE Computer,42(8), 30-37. Retrieved February 02, 2019.

Yifan Hu, Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. In 2008 Eighth IEEE International Conference on Data Mining (pp. 263–272). IEEE. https://doi.org/10.1109/ICDM.2008.22

Yong-Ping, D., Chang-Qing, Y., & Jing-Xuan, L. (2017). A new item-based deep network structure using a restricted Boltzmann machine for collaborative filtering. Frontiers of Information Technology & Electronic Engineering, 18(5), 658–666. https://doi.org/10.1631/FITEE.1601732

Zafari, F., & Moser, I. (2017). Modelling socially-influenced conditional preferences over feature values in recommender systems based on factorised collaborative filtering. *Expert Systems with Applications,87*, 98-117. doi:10.1016/j.eswa.2017.05.058