Music Recommender Systems

Rachel Zhu, Keat Chong Cheng, Timother Elgersma
{rzzhu, kc3cheng, trelgers}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Recommender systems have become essential for providing consumer satisfaction and ease. Music is a popular application of recommender systems and many people now rely on music recommendation services to find new songs. Finding the best way to make personalized recommendations is an ongoing process. In this report, we examine the two most common recommendation strategies: collaborative filtering and content-based filtering. We implemented a Logistic Matrix Factorization probabilistic model using user-song listening counts as implicit feedback for our collaborative filtering method. We based our content-based approach on vector representations of users and songs, and tested the content-based approach on three different similarity metrics: Euclidean Distance, Pearson Correlation, and Cosine Similarity. Using Mean Percentage Ranking (MPR) as our performance evaluation metric, our results show that all three similarity metrics perform equally on average. Additionally, we show that the collaborative filtering approach outperforms the content-based one.

Introduction

Music is universal and is an instrumental part of people's lives. It is used to express emotions, help relieve stress, elevate one's mood, and has many more use cases. IFPI's 2019 Music Listening report stated that people typically spent 18 hours a week listening to music (Moore 2019)and a previous study indicated that participants spent more time listening to music than watching television, reading books, or watching movies (Rentfrow and Gosling 2003). People are always looking to discover new music so we are looking to address the issue of helping users easily find music best fit to their tastes.

There exists a need for music recommendation services and a large percentage of people rely on them to explore new music. Spotify users have spent over 2.3 billion hours streaming the platform's Discover Weekly playlists in just 5 years and other music recommendation tools like Last.fm and Gnoosic have also grown in popularity (spo 2020). A good music recommendation system improves the user experience on streaming services and is able to quickly connect people with music they love. By bridging the gap between listeners and undiscovered music, these tools also help unknown artists get discovered by larger audiences and make it easy to introduce listeners to different genres of music. We want to explore 2 different methods and evaluate which approach makes for the best music recommendation system. This knowledge is also useful for a hybrid model, which makes use of both methods, to see which model should be given a larger weight to yield a stronger song recommendation system.

We want to find out how to introduce users to new songs that they enjoy listening to, and evaluate if it is more successful to suggest songs based on what other users with similar tastes listen to (collaborative filtering strategy), or suggest songs that have similar qualities to songs that the user likes (content-based filtering strategy). Both approaches try to answer the question of "closeness," but the data that is used differs. Collaborative filtering works on a large sample of (user, song) pairs, and content-based filtering requires an in-depth analysis of each song's acoustic features and metadata. Collaborative filtering is generally more successful, but suffers from the "cold-start" problem: When new songs appear, there is no historical data on the new songs available, so a collaborative filtering algorithm may perform poorly because of the lack of user scoring data for that particular song or song with similar characteristics (Liang, Zhan, and Ellis 2015). For our collaborative filtering approach, we implemented a Logistic Matrix Factorization probabilistic model using implicit user feedback based on the work by Johnson (Johnson 2014) but modified the definition of confidence. For our content-based approach, we followed the approach outlined by Li et al. in their paper about constructing user profiles for recommender systems and created song and user vectors using user listening counts and song audio features from the data set (Li and Kim 2004). We then use three different metrics to find similarity between the user profile vector and song vectors. We will experiment with Euclidean distance, Cosine distance, and Pearson Correlation. It should be noted that the performance of the algorithms heavily depend on the quality of the data sets. Because each recommendation approaches requires data set of a specific nature (e.g. collaborative filtering approach requires user listening history, content-based uses metadata or audio source) , the data set used for each approach will be different, therefore it will not be a completely equal and fair comparison.

To measure how the different approaches performed, we used Mean Percentage Ranking (MPR) as an evaluation metric. A lower MPR indicates better performance. We trained the models on a subset of the user listening history and then measured how our recommendations compare to the full user listening history. We found that the collaborative filtering model performed better than the content-based approach, with average MPRs of 48.65% and 56.65% respectively. There were no statistical differences in our content-based trials across three similarity metrics: Pearson Correlation, Cosine Similarity, and Euclidean Distance.

The relative strength of these models can be useful in the creation of a hybrid recommender, where one could apply more weight to the collaborative filtering model than the content-based model to yield better results. Since no similarity metric was shown to outperform any others, future content-based recommender systems using our dataset can use this observation and choose whichever is more convenient.

Through our research, we made several contributions to the field of recommenders.

- We implemented a sharded logistic matrix factorization algorithm.
- We found that Pearson Correlation, Cosine Similarity, and Euclidean Distance as similarity metrics for a content-based system all perform similarly on our dataset.
- We collected a 770,000 song dataset from Spotify's track analysis API.

Related Work

Most existing music recommendation systems use a content-based strategy, collaborative filtering strategy, or a hybrid of the two.

Content-Based Approach

Content-based music recommendation strategy depends on the description of music attained by extracting semantic information from and about music at different representation levels (e.g. audio signal, artist, song name, album cover) (Ricci, Rokach, and Shapira 2015). This approach recommends music by computing the similarity between songs through learning the music's feature vector.

There are two common types of information sources used in content-based approaches. One type of information source is musical metadata (e.g. human annotations, social tags obtained from collaborative tagging services such as Last.fm, and mined annotations from websites (Ricci, Rokach, and Shapira 2015). Another category of information comes from features extracted directly from audio sources (e.g. acoustic and musical features) by applying signal processing and analysis methods directly to audio (Ricci, Rokach, and Shapira 2015).

Musical Metadata In their paper, Bogdanov et al. proposed a content-based recommender system using editorial metadata from the Discogs.com music database. The approach provides artist recommendations starting from an arbitrary set of music (preference set) provided by the user to represent their music preference. The researcher created descriptive tag-based artist profiles containing information such as particular genres, styles, record labels, years of release activity, and countries. The artist profiles were then represented as vectors in a latent semantic space of reduced dimension, which reduces tag sparsity. The system recommends artists by applying a distance measure between the resulting artist vectors for the music in the user's preference set and the music within the entire music collection. The researcher found that the performance was comparable to the other metadata-based approaches (e.g. genre metadata, artist similarity based on Last.fm tags), including an industrial recommender (iTunes Genius) through subjective evaluation from 27 participants. Similarly, the technique described by Green et al. also recommended artists based on artist-to-artist similarity and similarity between artists in the database and a vector of keyword weights summarizing the user's favourite artists, using keywords from Wikipedia artist entries and social tags from Last.fm as information sources (Green et al. 2009). Besides that, Van et al. proposed a deep content-based music recommendation system by using a latent factor model (Van den Oord, Dieleman, and Schrauwen 2013). The researcher applied the weighted matrix factorization (WMF) algorithm to learn the latent factor representation of all users and songs in the Million Song Dataset that consists of play counts per song per user. Interestingly, the researcher also experimented with predicting latent factors from music audio using convolutional neural networks (CNN) when latent vectors cannot be learned from user usage data. They found that this approach outperforms a traditional approach of using bag-of-words representations of audio signals. Although the predicted latent factor produces a sensible music recommendation, the researcher found that a wide semantic gap between the corresponding audio signal and the characteristics of a song that affects user preference (e.g. popularity) (Van den Oord, Dieleman, and Schrauwen 2013).

Audio Source One paper discusses how a traditional content-based recommender system that uses only traditional audio content features, such as Mel-Frequency cepstral coefficient (MFCC), does not perform well in predicting user preference (Zhong, Wang, and Jiao 2018). Zhong et al. proposed a CNN for content-based music recommendation. Each song in the data set is split into 20 segments and converted into images with Fourier transformation and coloured with the amplitude of sound. The researcher then applied deep CNNs on these music representations and recommended music using a neighbourhood-based approach by finding the nearest neighbour using a cosine distance measure.

Lastly, Lim et al. showed the potential of models using both audio sources and metadata as information sources to solve song-similarity tasks using data. The researcher experimented with three song representations, derived from either audio features or lyrical content. The researcher first used the learned audio and lyrics representation on robust extension to the Metric learning to rank algorithm (R-MLR), and then included 33 additional song descriptors (e.g. mean tim-

bre, pitch distribution, tempo, loudness, "danceability") with the existing audio or lyrics representation. When the additional 33 musical features are included, R-MLR does significantly better on average AUC score (Lim, Lanckriet, and McFee 2013).

Collaborative Filtering Based Approach

Collaborative filtering based recommender system predicts the user's future listening pattern by analyzing the user's past behavioural data. A user's past behavioural data can be either explicit or implicit feedback or a combination of the two. Explicit feedback is data provided by the user to express their sentiment towards songs (e.g. Spotify's like button). Implicit data is telemetry data collected by observing user actions (e.g. purchases, length of playing time, music streaming play counts) (Johnson 2014).

A significant category of collaborative filtering approaches is based on the latent factor model. It assumes that a low-dimensional latent representation of both users and songs exists, and the compatibility between a user and a song, modelled as their inner product in this latent space, predicts the user's sentiment of the song (Liang, Zhan, and Ellis 2015). The latent factor vector forms a compact description of the different facets of the user's tastes and the corresponding characteristics of music (Van den Oord, Dieleman, and Schrauwen 2013). Another approach is neighbour based. Neighbour based collaborative filtering models can be either user-based or song-based. User-oriented neighbourhood method methods search within the set of users to find similar users who have rated or interacted with songs and then recommend those songs that similar user prefers or interacted. Song-oriented neighbourhood methods search within the songs database to find new songs similar to ones that the user has either rated highly or frequently interacted with (Johnson 2014).

Explicit Feedback The most well-known example of the use of collaborative filtering with explicit feedback data is the Netflix Prize, an open competition to come out with a collaborative filtering model that beats the existing Netflix recommender system . Bell et al predicts user ratings by integrating models that focus on patterns at different scales. At a local scale, the researcher used a neighbourhood-based technique that infers ratings from observed ratings by similar users or observed ratings of similar items (Bell, Koren, and Volinsky 2007). The neighbourhood method is extended to allow item-item weights to vary by user and user-user weights to vary by item. At a regional scale the researcher learned latent factors that characterize all users and items using Singular Value Decomposition (SVD)-like matrix factorization (Bell, Koren, and Volinsky 2007). To fill in the unknown rating values, each unknown rating is estimated using the dot product of user factors with item factors (Bell, Koren, and Volinsky 2007). The combination of local and regional approaches perform favourably than other collaborative filtering approaches and the Netflix recommender system (Bell, Koren, and Volinsky 2007).

Implicit Feedback In contrast with the explicit feedback approach, Johnson proposed a collaborative filtering method with implicit feedback using data collected by Spotify (Johnson 2014). The researchers computed the probability that a user has chosen to listen a song by assuming a distribution of a logistic function parameterized by the sum of the inner product of user and song latent factor vectors and user and item biased. The bias terms are latent factors assigned to each song and users to offset variations in human behaviour and popularity bias of top songs. Similarly, Minh et al. provide a probabilistic framework for the implicit case where they model the probability of a user choosing an item as a distribution of a normalized exponential function (Mnih and Teh 2012). Minh et al. also utilized tree-structured label spaces to reduce the normalization cost.

Hybrid Recommendation System

There is evidence from a study that showed that content-based approaches might underperform when compared to collaborative filtering approaches (Green et al. 2009). However, content-based approaches can be used to supplement collaborative filtering based models by using the content model as a prior for collaborative filtering in cold-start scenarios (Bogdanov and Herrera 2012). A study that attempted to mitigate the cold start problem is a study by Liang et al.. Liang et al. incorporated content information into a collaborative filtering method by training a neural network on semantic tagging information as a content model and used it as a prior in a deep collaborative filtering model (Liang, Zhan, and Ellis 2015). Specifically, the researcher pre-trained a multi-layer neural network to predict semantic tags from vector quantized acoustic features, and the output of the neural network is treated as a latent factor that serves as a high level representation of the musical content. The resulting latent factor is then used as a prior for the song latent representation in the collaborative filtering model. They showed that the hybrid approach is comparatively better than just the collaborative filtering approach, especially in the cold-start case.

Methodology

We define

- $U = (u_1, ..., u_n)$: a set of n users
- $S = (s_1, ..., s_m)$: a set of m songs
- $R = (r_{us})_{n \times s}$: a user-song observation matrix where r_{us} denotes the number of times user u listens to song s
- $C = (c_{si})_{m \times f}$: a song-feature matrix where f denotes the number of audio features for a song track and c_{si} denotes audio feature i for song track s

The goal of both recommender systems is to find the top recommended songs that a user has not yet interacted with.

Collaborative Filtering: Logistic Matrix Factorization

We decided to employ Logistic Matrix Factorization (Logistic MF), a probabilistic model for matrix factorization with implicit feedback, based on the work by Johnson for our collaborative filtering approach. To do this, we factorized the

observation matrix R into two lower-dimensional latent representations of users and songs, i.e. $X_{n\times f}$ and $Y_{m\times f}$ where f is the number of latent factors. The rows of $X_{n\times f}$ are latent factor vectors representing a user's song profile, and the rows of $Y_{m\times f}$ are latent factor vectors encoding the inferred characteristics of a song. By assuming that the entries in R are independent and distributed according to a logistic function, we solve the optimal $X_{n\times f}$, $Y_{m\times f}$ by maximizing the log posterior using an alternating gradient ascent procedure.

Let $l_{(u,s)}$ denote the event that user u prefers to listen to song s. We model the probability of this event with a logistic function parameterized by the sum of the inner product $\langle x_u, y_s\rangle$, the user biases β_u, and song biases β_s.

$$Pr(l_{(u,s)}|x_u, y_s, \beta_u, \beta_s) = \frac{exp(x_u y_s^T + \beta_u + \beta_s)}{1 + exp(x_u y_s^T + \beta_u + \beta_s)}$$

The bias terms β_u, β_s are latent factors associated with each user $u \in U$ and each song $s \in S$ to offset the variability in behaviour across both users and songs. An example of variability in users is that some users consume a wide variety of music, while others only listen to a specific music genre. Similarly, some songs in a specific genre are popular and will have a higher likelihood of being listened to by many users, while other songs that are less popular will only be listened to by a small subset of specific users.

Denote the non-zero entries of R as positive observations and zero entries of R as negative observations. Our premise is that if user u has listened to song s a lot (greater values of r_{us}), then this indicates that they prefer this song. Contrarily, low listening counts indicate that the u does not prefer s because they chose not to continue listening to the song after hearing it. Negative observations ($r_{us} = 0$) suggest that u does not prefer s. However, it could also be that the user likes the song, but has only recently encountered it. For negative observations, we are less confident in whether or not the user prefers the song since we have no explicit sentiment (like/dislike) from the user about the song. Therefore, we are more confident in entries with high listening count since we can infer with confidence that the user prefers the song, while entries with low or zero listening count lead to lower confidence in inferring that the user prefers the song because we have insufficient evidence about the user's interaction with the song. We measure these levels of confidence in r_{us} by introducing a set of variables, c_{us}. We define $c_{us} = 1+\alpha r_{us}$ where α is a tuning hyper-parameter. As the value of r_{us} increases, our confidence in observing positive observations in R increases and α denotes the rate of increase in confidence. Increasing α places more weights on positive observations while decreasing α places more weight on the negative observations.

Assuming that all entries in R are independent, we derive the likelihood of our observations R given the parameters $X_{n\times f}$, $Y_{m\times f}$ and biases β as:

$$L(R|X, Y, \beta,) = \Pi_{u,s} Pr(l_{u,s}|x_u, y_s, \beta_u, \beta_s)^{1+\alpha r_{us}} (1 - l_{u,s}|x_u, y_s, \beta_u, \beta_s)$$

Moreover, assuming that rows of X and Y are uncorrelated and have a common variance, we assume Gaussian priors with zero mean for latent matrices X, Y, which helps to regularize the model and avoid overfitting.

$$Pr(X|\sigma^2) = \Pi_u N(x_u|0, \sigma_u^2 I)$$
$$Pr(Y|\sigma^2) = \Pi_s N(y_s|0, \sigma_s^2 I)$$

With the priors and the likelihood function above, we derive the log posterior and replace constants (e.g. $\frac{1}{\sigma_u^2}$, $\frac{1}{\sigma_s^2}$) with a common scaling parameter λ:

$$\begin{aligned}\log Pr(X,Y,\beta|R,\sigma^2) = &\textstyle\sum_{u,s}(1+\alpha r_{u,s})(x_u y_s^T + \beta_u + \beta_s) \\ &- (2+\alpha r_{u,s})(\log(1+exp(x_u y_s^T + \beta_u + \beta_s))) \\ &- (\tfrac{\lambda}{2}\|x_u\|^2 + \tfrac{\lambda}{2}\|y_s\|^2)\end{aligned}$$

Our objective is to obtain an estimate of X, Y, β that maximizes the above log posterior, i.e.:

$$\arg\max X, Y, \beta (logPr(X, Y, \beta|R, \sigma^2))$$

We will perform an alternating gradient ascent procedure to find a local maximum to this optimization problem. The alternating gradient ascent procedure is as follows: In each iteration, we first fix the user vectors X and biases β, then take a step towards the gradients of the song vectors Y and biases β. In the next iteration, we fix the song vector Y and biases β, then take a step towards the gradients of the user vectors X and biases β. We repeat the above procedure until we reach convergence. The gradients for the user vectors, song vectors, and biases are as follows:

$$g_{x_u} = \tfrac{\partial}{\partial x_u} = \textstyle\sum_i (1+\alpha r_{ui}) y_i - \frac{y_i (2+\alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_s)}{1+exp(x_u y_i^T + \beta_u + \beta_s)} - \lambda x_u$$

$$g_{y_s} = \tfrac{\partial}{\partial y_s} = \textstyle\sum_i (1+\alpha r_{is}) x_i - \frac{x_i (2+\alpha r_{is}) exp(x_i y_s^T + \beta_u + \beta_s)}{1+exp(x_i y_s^T + \beta_u + \beta_s)} - \lambda y_s$$

$$g_{\beta_u} = \tfrac{\partial}{\partial \beta_u} = \textstyle\sum_i (1+\alpha r_{ui}) - \frac{(2+\alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_s)}{1+exp(x_u y_i^T + \beta_u + \beta_s)}$$

$$g_{\beta_s} = \tfrac{\partial}{\partial \beta_s} = \textstyle\sum_i (1+\alpha r_{is}) - \frac{(2+\alpha r_{is}) exp(x_i y_s^T + \beta_u + \beta_s)}{1+exp(x_i y_s^T + \beta_u + \beta_s)}$$

Each iteration is linear in the number of users $|U|$ and songs $|S|$. Linear time complexity is computationally heavy, given that our dataset is large. Therefore, we take an approach suggested by Johnson, where we sample fewer zero entries in R and decrease α in response to approximate the full loss function. Moreover, we further reduce our computation time using AdaGrad, an algorithm that adjusts the gradient step size adaptively at each iteration and it was found to reduce the number of iterations required to converge by Li et al. An example of an AdaGrad update at each iteration is as follows: Let γ be the initial step size, x_u^t denote the value of user vector x_u at iteration i, $g_{x_u}^i$ denote the gradient of user vector x_u in with respect to x_u at iteration i. At iteration i, the AdaGrad update to x_u is:

$$x_u^t = x_u^{t-1} + \frac{\gamma g_u^{t-1}}{\sqrt{\sum_{t'=1}^{t-1} (g_u^{t'})^2}}$$

We choose a Bayesian probabilistic approach (which encodes the probability that a user listens to a song) over binary-based matrix factorization (which encodes the user

songs' preference in binaries) such as Implicit Matrix Factorization because probability encoding provides additional insights into the degree to which a user prefers a given song. There is a lot of literature on probabilistic matrix factorization, such as Mnih et al.'s 2008 paper and Gopalan et al.'s 2013 paper. We choose Logistic MF over other probabilistic frameworks because the algorithm has been evaluated using a data set consisting of user listening behaviour, and the algorithm has achieved excellent performance (Johnson 2014). Therefore, assuming R's entries are distributed according to a logistic function will yield a higher chance of success. Moreover, Logistic MF has an additional advantage that accounts for the variation in user and item behaviour.

Content-based Filtering

For our content-based approach, we calculate the user-song similarity by first experimenting with various similarity metrics and we recommend songs that are the most similar to the user based on the best performing similarity metric.

We need to transform songs and users into vectors of the same subspaces using the user's listening count in R, and song tracks' audio features from C to compute the similarity between users and songs. We will take a similar approach outlined as outlined by Li et al. to construct a feature-based user profile vector by aggregating each audio feature of song tracks that the user has previously listened to. Therefore, the user profile vector represents a user's song preferences, and we denote the user profile vector for user u as

$$\vec{u} = \begin{bmatrix} a_1 \\ \vdots \\ a_f \end{bmatrix}$$

Recall that f denotes the number of audio features for a song track. Each entry a_i in $\vec{u}$ corresponds to an audio feature of a song tracks i and is calculated using audio features i of songs tracks in user u listening history as follows:

$$a_i = \text{Avg}_{s \in S}(c_{si} | \text{count}(s) > \text{threshold})$$

Since a high listening count for a song indicates the user's fondness towards the song, we define a threshold variable which filters out songs with low play counts when constructing the user profile vector. For example, suppose we want to calculate a_i of a particular audio feature i for user u, who has a listening history of songs A, B, C, with listening counts 3, 4, 5 respectively, and the value of the threshold is 4. We will only consider songs with listening counts greater than or equal to the threshold value (i.e. B, C) in constructing u's user profile. If attribute i for songs B, C has values 10, 20 respectively, then the value of a_i is 15, the average of the value for B and C.

Since the similarity metric's performance depends on the dataset's distribution and nature, we shall experiment with three popular similarity metrics to compute the similarity between the user and song vectors: Euclidean distance, Cosine distance, and Pearson Correlation. For each user profile vector $\vec{u}$ and song vector $\vec{s}$, the formula of each similarity metric is as follows:

$$\text{Euclidean}(\vec{u}, \vec{s}) = \sum_i (u_i - p_i)^2$$

$$\text{Pearson}(\vec{u}, \vec{s}) = \frac{\sum_i (u_i - \bar{u})(s_i - \bar{s})}{\sqrt{\sum_i (u_i - \bar{u})^2} \sqrt{\sum_i (s_i - \bar{s})^2}}$$

$$\text{Cosine}(\vec{u}, \vec{s}) = \frac{\sum_i u_i s_i}{\sqrt{\sum_i (u_i)^2} \sqrt{\sum_i (s_i)^2}}$$

Cosine similarity is similar to Pearson correlation similarity except that Pearson correlation is centred. Both have a range of $[-1, 1]$, observe the vector direction, and ignore the magnitude of the vectors. Unlike the Cosine and Pearson correlation similarity metrics, Euclidean distance is sensitive to the magnitude of vectors. It calculates the distance between the two vectors and is unbounded. Our data has been normalized before computing the similarity, so Euclidean's sensitivity to magnitude will not pose issues in our dataset (Vintch 2020).

After having the user profile vector and song vectors for each song in the dataset, we compute the similarity of each user profile vector with every song in the dataset. Then, we recommend songs by selecting songs in order of proximity with the user vector profile.

Dataset

We used the Echo Nest taste profile subset and the Echo Nest track ID list from The Million Song Dataset. The taste profile subset consists of 48 million `(user_id, song_id, play_count)` triples for one million distinct users. Each user has between 10 and 4400 triples recorded, with a median of 27. The Echo Nest taste profile gives us the information required to build one million listener profiles. To translate song IDs to a useful (song title, artist name) tuple, we use the track ID list. It contains one million songs, with a median of 4 songs per artist. However, we also need information about the content of each song for content-based filtering. We augmented the Echo Nest track ID list with the Spotify API to get this data, using a two step process for each record.

1. Search `api.spotify.com/v1/search` for the song title and artist name to get a song ID. If a search fails, try again with a pair of brackets removed because some songs from the dataset included a descriptor such as (LP Version) in the title, which may not exist on Spotify.
2. To get the features object for the song, we pass that song ID to `api.spotify.com/v1/audio-analysis`

The features object contains:

- A list of sections roughly corresponding to verses, choruses, and bridges. Each section has attributes for time information, loudness, tempo, key, mode (either major or minor) and time signature. We used these values to compute the median and variance of the duration, loudness, tempo, and key for each segment.
- A list of segments, where each segment contains a roughly consistent sound. Each segment has attributes for time information, loudness, and 12-item vectors for pitches (corresponding to the 12 pitch classes) and timbre. We used

these values to compute the median and variance for the duration, maximum loudness, starting loudness, and ending loudness. We also computed the median and variance for each term of the pitches and timbre vectors.

- A list of tatums. A tatum is the smallest distinguishable sound in a piece of music– a beat is made up of multiple tatums. We calculated the median and variance of tatum lengths.
- Some general track information. We kept loudness, tempo, time signature, key, and mode.

In total, this is 36 features for each song. Some songs didn't have a match on Spotify's database or, once matched, the audio analysis was missing some basic features. This second case occurred frequently in very short songs. We trimmed down the taste profile subset so it only contained songs for which we could obtain all the information from Spotify. This resulted in 770 000 records in the song list, and 39 million records in the taste profile subset. This dataset works well for collaborative filtering, because we have exactly what we need to build a user profile, and enough song metadata to identify the songs. Unfortunately, the dataset does not provide information on when each song is listened to by the user. The timestamp would be useful for collaborative filtering because we can emphasize more weights on recently listened songs than older songs as a user profile may change slightly over time. It also works for content filtering, because we have an in-depth audio analysis of many songs, and we can use user histories as test sets to validate our approach.

Results

For our collaborative filtering approach, we will apply Logistic MF to decompose R into two matrices $X_{n\times f}$, $Y_{m\times f}$ and learn an local optimum solution that maximizes the log posterior. For our content-based approach, we will first create a user vector profile from user listening history and audio features of sound tracks by averaging each audio feature of song tracks that the user has previously listened to. We then experiment with three similarity metrics i.e. Euclidean distance, Cosine distance, and Pearson Correlation and use the similarity metric with the lowest MPR score to compute the user-similarity metric. We will train our models with 90% of our dataset and use the other 10% as a test set, T. Johnson used this ratio in his experiment and obtained good results (Johnson 2014). We will create a training matrix, R_t, which takes R and replaces 10% of randomly selected entries, r_{us}, with the value 0 indicating that user u has not listened to song s. The test set, T, is a dictionary of users u and a list of the corresponding songs s from the removed entries. By taking away r_{us} data points, we now have full (user, song) pairs to test our model on. We will evaluate our recommender models by comparing the returned recommended values to the actual values of the removed r_{us} entries. Since this result can vary depending on which songs we remove for each user, we will use cross-validation to decrease variance and bias. To do this, we form ten test sets for a 10-fold cross validation by randomly sorting the entries in matrix R into ten equally sized partitions. For each partition, we will use it as the test set once and form a training matrix, R_t, with the rest of the values to train our model on. We then take the average value of the evaluated user satisfaction across the ten experiments.

Our evaluation metric takes a list of songs in the recommended order for each user u in T. The list for user u will consist of songs s for every $(u, s) \in T$. We will validate the results by comparing how highly recommended the song is (its order in the list) to how much we predict the user will like the song (higher listen count, r_{us}, indicates higher satisfaction). We use Mean Percentage Ranking (MPR) to evaluate user satisfaction with the ordered list of recommended songs.

MPR is defined as

$$\text{MPR} = \frac{\sum_{us\in T} r_{us} * \text{rank}_{us}}{\sum_{us\in T} r_{us}}$$

We only look at $M_{rank}[user, song]$ (u, s) from our test set, T, which we can extract by iterating over every $u \in T$ and forming (u, s) pairs for every s in the corresponding list of songs. rank_{us} represents the percentile ranking of song s in the returned recommended list of songs for user u. So $\text{rank}_{us} = 0\%$ for the most highly recommended song and $\text{rank}_{us} = 100\%$ for the least recommended song. We do not want to evaluate songs that the user has not listened to since we cannot determine if they would prefer it or not. When $r_{us} = 0$, its term in the MPR sum is 0 so MPR ignores songs that the user has not listened to. If a song is listened to a lot (r_{us} is high), then this is a good indicator that the user likes this song. Then we want our model to have highly recommended this song (rank_{us} is small). If a song is not listened to much (r_{us} is low), then this indicates that the user does not like the song as much, so we want our model to give it a lower recommendation (rank_{us} is greater). Thus, we want to minimize our value by having higher rank values correspond to low r_{us} values and vice versa. Ultimately, a lower MPR indicates higher user satisfaction. Randomly recommended songs should average to an MPR of 50%, so a well-performing recommender system should have an MPR lower than 50%.

We chose to use MPR as an evaluation metric since it was also used by Johnson, who achieved an MPR value of 6-25% in his experiments. Additionally, MPR will ignore songs that we cannot evaluate and factors in false positives (recommending songs that the user doesn't like). We decided it was not as important to consider false negatives (not recommending songs that the user would potentially like) since there are likely an abundant amount of these songs that exist and it will not affect the user's satisfaction because they are only concerned with the songs that are currently being recommended to them.

For our collaborative filtering approach, our definition of c_{us} differs from the definition in Johnson's paper ($c_{us} = \alpha r_{us}$) because our definition allows us to place some degree of confidence for every user-song pair in R, even for negative observations (Johnson 2014). We cannot give 0 confidence for unknown songs because there is still a chance that the user may like it. Also in Johnson's approach, he tested multiple numbers of latent factors, f. In his graph compar-

ing the number of latent factors to the resulting MPR value, MPR tapers off around 40 latent factors so we will choose to use this number.

We have a total of three hyperparameters for our Logistic MF algorithm, namely α in confidence levels c_{us}, scaling parameter γ, and AdaGrad's initial step size λ. α denotes the rate of increase in confidence. Increasing α places more weights on positive observations ($r_{us} \neq 0$) while decreasing α places more weight on the negative observations. We will tune α by cross-validation, starting with $\alpha = 40$ as suggested by Hu et al. (Hu, Koren, and Volinsky 2008). According to Johnson, choosing α to balance the positive and negative observations yields the lowest MPR (Johnson 2014), hence we will tune α in the direction of balancing positive and negative observations. γ is the initial step size of AdaGrad algorithm. We will set the value as 1.0, as used by Johnson. This hyperparameter does not need to be tuned as the step size will be changed adaptively at each iteration in the AdaGrad. λ is the scaling parameter which regularizes the model. We will use $\lambda = 0.6$ as suggested by Johnson.

For our content-based approach, we decided to evaluate 3 different ways of calculating similarity. We are interested to see which similarity metric namely Euclidean distance, Cosine distance, and Pearson Correlation will yield a better performing recommender system. We will use the best performing similarity metric in our final comparison against our collaborative filtering approach. When creating a user profile vector, we use a $threshold$ variable to filter our songs that we do not want to include in our user profile. In the paper by Li and Kim, this value is fixed at a constant number (Li and Kim 2004). Instead, we chose to set the threshold as the median number of listening counts in user u's listening history so that our algorithm will consider on average half of the more preferred songs for user u. We want to account for the variance in user listening counts (e.g. users who obsessively listen to songs will have higher listening counts for preferred songs compared to other users). We expect that this will construct a more accurate user u profile vector.

Implementation

Our project code, as outlined in Figure 1, is available on Github.[1]

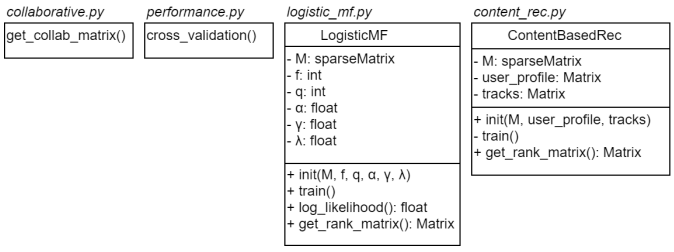

Figure 1: Our project UML. We created a Python object for each recommendation approach and called them from `performance.py:cross_validation()`.

[1] github.com/timothy-e/music-recommender

Collaborative Filtering First, we construct the listening count matrix, M (an $n \times m$ sparse matrix), using the Echo Nest triples dataset (each line in the dataset contains a listening count for a (user, song) pair for n users and m songs). We then construct a `LogisticMF` object with M and model parameters f, q, α, λ, and γ, where f is the number of latent factors and q is the number of iterations. α, λ, and γ were defined in the Methodology section.

Next, we train the model using the method detailed in the Methodology section. Each iteration step is a complex process that requires a lot of memory and involves large matrix multiplication and operations on every item in M. To overcome this limitation, we divide M into square shards, compute the value for each shard, and then combine the shards to complete the iteration. Each step of the iteration develops U, an $n \times f$ matrix where each row is a user vector made up of f latent factors, and S, an $m \times f$ matrix where each row is a song vector made up of f latent factors. We then create a new $n \times m$ matrix $M' = U \times S^T$. M' is our expected listening count for every (user, song) pair. We generate M' row-by-row because it is too large to be hold in memory.

The final step is to convert the listening counts for each user to its rank value defined in the MPR description in the Methodology section. The ranked list contains -1s for each positive observation in the given listening count matrix, M. The other values are remapped to a range $[0, 1]$, where 0 represents the most recommended song and 1 represents the least recommended song. This process naturally can be done row-sequentially as each row represents a single user.

Content-Based Filtering We define a helper function `get_user_profile_matrix()` to create a user profile matrix given the user's listening history and the metadata for each song. We compute the median play count for each user by performing a group by operation on user_id with Pandas. For each user, we filter out all the songs in the user's listening history that have a lower play count than the user's median play count. We then calculate the mean value of each attribute in the remaining set of songs to produce the user profile vector. The function returns an $n \times f$ Pandas data frame where n is the number of users and f is the number of song audio features.

For the content-based recommender, we define an object `ContentBasedRec` that computes and stores the similarity matrix for each of the three similarity measures (Euclidean distance, Cosine similarity, Pearson correlation). We compute the similarity matrix with `sklearn.metrics.pairwise_distances()`. This method can be configured to breakdown the pairwise matrix into j even slices to compute in parallel or reduce concurrent memory usage. We use the class function `get_rank_matrix()` to compute and retrieve the rank of each song for a particular user. The returned rank matrix is $n \times m$ in size. Since the matrix is too large to be stored in memory, we generate and return the matrix row by row to prevent heap memory error like our collaborative filtering implementation of `get_rank_matrix()`.

Performance Measures We use the original listening count matrix, M, returned from `get_collab_matrix()`

and the rank matrix, M_{rank}, returned from `get_rank_matrix()` to calculate MPR, our evaluation metric. $M_{rank}[user, song]$ holds the corresponding rank value from the MPR equation for the entry $M[user, song]$. There is no rank for non-zero entries in M because we do not recommend songs that the user has already listened to previously. The non-zero entries are indicated by -1 in M_{rank} and we ignore them in our MPR calculation.

To perform 10-fold cross-validation, we randomly partition the entries in M into 10 test sets. The partition is done by first calling `scipy.sparse.find()` which returns three separate arrays (i, j, v) of length k, representing the position and values of the k nonzero entries in M. That is, for a nonzero entry e, $i[e]$ gives the user, $j[e]$ gives the song, and $v[e]$ holds the listening count which is equal to $M[i[e], j[e]]$. We build an array holding the indices for each e (storing the numbers from 0 to $k-1$) to represent all nonzero entries. This array is then randomly shuffled and divided equally into 10 subsets using `np.array_split()` to create the 10 test sets.

For each test set, we build the corresponding training set, M_{train}, by first initializing M_{train} to be a copy of M. Then for every entry e_t in the test set, we use the position arrays i, j to set the corresponding entry value in M_{train} to zero (i.e. $M_{train}[i[e_t], j[e_t]]) = 0$) to indicate that the user has not listened to this song.

Next, we train M_{train} on each algorithm. For each method, we use the rank matrix from `get_rank_matrix()` to calculate the MPR. The MPR for each approach is stored and then averaged after iterating over all of the test sets.

Technical Challenges and Optimizations After directly implementing our mathematical formulae, we found that the intermediate results were far too large to be stored in memory. It's feasible to store an $n \times f$ user vector matrix U and an $m \times f$ song vector matrix S, but a full $n \times m$ matrix exceeds the RAM limits of the most expensive options on Google Cloud.

To optimize the code for space efficiency, we use sparse matrices to store the initial observation matrix M for collaborative filtering. Constructing M requires the index of every user and song as we encounter it. Storing these `(id, index)` pairs in a dictionary ($\mathcal{O}(1)$ retrieval for each n) resulted in dramatically faster matrix construction (from minutes down to seconds) than searching through a list of unique users/songs for these indices as they appear ($\mathcal{O}(n)$ retrieval for each n).

To calculate the next step at each iteration, we use the same sharding strategy as other logistic matrix factorization approaches (Das et al. 2007), (Johnson 2014). For each iteration, we break U and S into smaller chunks, compute the partial derivative summation, and sum to complete the computation. We use the same approach for calculating log likelihood without running out of memory.

To find the optimal size for each partition, we plotted the iteration run time for each configuration. We ran the iteration 8 times for each width and height and discarded the two fastest and the two slowest runs, to account for variation in the runtimes. Figure 2 shows that setting our partition size to 300×300 optimizes the run time.

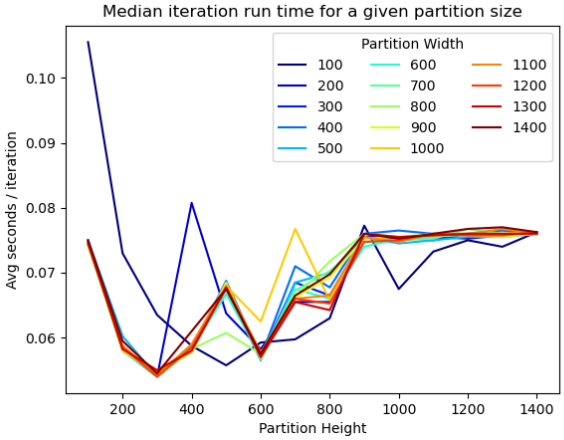

Figure 2: Comparison of median iteration run time for different partition sizes. The entire (user, song) listening count matrix cannot be stored in memory at once so we need to break the matrix into chunks. To compute the prediction matrix efficiently, we want chunks that result in the shortest run time.

The final step of the output requires multiplying U and S together to get a full $n \times m$ prediction matrix M'. We re-implemented matrix multiplication to calculate and yield one row of the prediction matrix at a time because the matrix is too large to be stored in memory. Every subsequent function that uses this matrix M' must also be a generator, working on the result row-by-row.

Performance

Based on our hyperparameter tuning for our collaborative filtering approach, we found that setting α to 0.05 yielded the lowest MPR (see Figure 3). As α increases, we give more weight to the nonzero entries in M (i.e. we have more confidence in songs that the users has already listened to). Larger values of α result in an overfitted model that prefers only the songs that the user has listened to. We want to avoid this as the purpose of the recommender system is to find new songs that the user has not heard before. We found that lower values of α result in a better balance between giving preferences to songs which the user has shown interest in (high listening count) while still having confidence in songs that the user has not listened to yet.

We evaluated our collaborative filtering model on latent factors from 10 to 70 using the MPR evaluation metric (see Figure 4). The MPR is high at the beginning because the model is underfitted. It does not have enough complexity to capture all the useful information in the dataset. It starts to perform better as the number of latent factors grows but starts to overfit after 40 when the number of latent factors becomes too large. Having so many latent factors leads the model to capture too many characteristics that are only

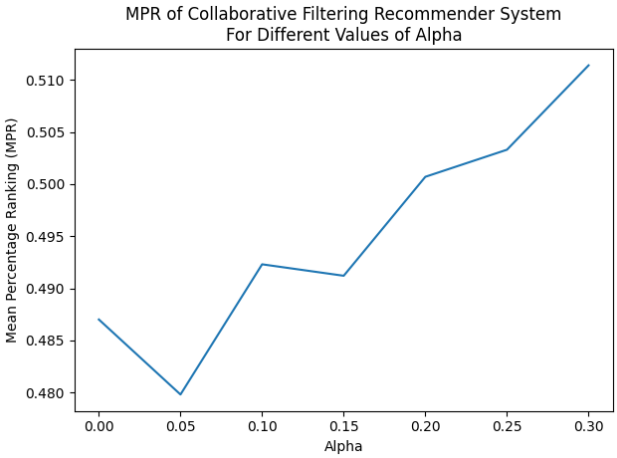

Figure 3: Performance evaluation of our collaborative filtering approach on different values of alpha using MPR (Mean Percentage Ranking). Lower MPR values indicate better performance. Alpha (α) is a variable in the collaborative filtering algorithm which represents the "confidence" in the observations on the user's song history.

present in the training set. We use 40 latent factors in our final algorithm as it resulted in the lowest MPR score.

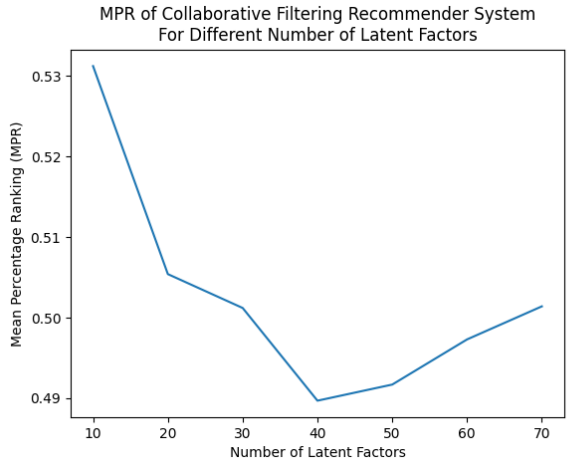

Figure 4: Performance evaluation of our collaborative filtering approach on a range of latent factors (f) using MPR (Mean Percentage Ranking). Lower MPR values indicate better performance.

From our 10-fold cross-validation results on the different algorithms (see Figure 5), the collaborative filtering method performed the best averaging an MPR of 48.65%. The content-based approaches yielded an average MPR of 56.65%. As expected, the collaborative filtering method performed better than the content-based methods. This supports the evidence favouring collaborative filtering over content-based filtering from previous studies (Liang, Zhan, and Ellis 2015) and explains why popular services like Amazon, Youtube, Netflix, and Spotify primarily emphasizes collaborative filtering approaches as part of their hybrid recommendation systems. From multiple trials, we found that the three different distance-measuring algorithms for calculating similarity in our content-based approach performed similarly (typically ranging from 56% to 57%). There was no predictable pattern of any similarity measure outperforming the others. Additionally, conducting an ANOVA test showed that there was no statistically significant evidence that one similarity measure was better.

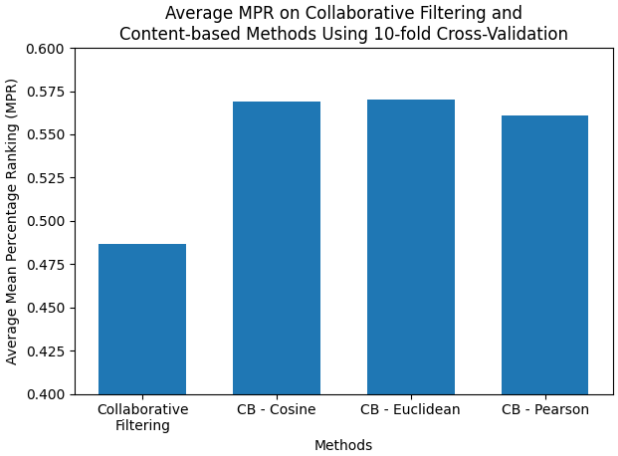

Figure 5: Comparison of collaborative filtering (CF) and content-based (CB) approaches using MPR (Mean Percentage Ranking) as the performance metric. Lower MPR values indicate better performance. Our content-based method was tested with three different algorithms for calculating similarity between song and user vectors: Cosine similarity, Euclidean distance, and Pearson correlation.

Lessons

We learned a lot from this project. The biggest surprise was the significance of choosing efficient data structures and algorithms when working with a huge dataset. We found that our full matrices required terabytes of RAM, but our computations performed significantly better after converting them into a sparse matrices, so it's important to keep matrices sparse for as long as possible. Reducing the algorithmic complexity by an order of magnitude results in a runtime that's hours shorter. Writing code that handles a such a large dataset is very different from working on projects for class assignments that are of a much smaller scale. Working with a dataset containing a million songs forced us to evaluate how our implementation decisions affect the performance since it resulted in evident drawbacks.

We also learned that some optimizations are great on paper, but don't work out in real applications. We attempted to annotate a complex function with Numba, a library that produces LLVM optimized machine code (Lam, Pitrou, and Seibert 2015), to run our code more efficiently. The func-

tion ran five times slower with Numba than without. It's important to always verify what needs to be optimized and to measure the optimization results.

Finally, we learned that the size and quality of the dataset are crucial for a successful recommender. Despite using the same algorithm for our collaborative filtering approach with minor adjustments, we achieved sub-par results compared to the experiment run by Johnson. Not only was our dataset significantly smaller, it contained lots of users but limited song history (users had listen counts for a median of 27 songs) whereas his dataset had less users but was rich with song interactions.

Discussion

The main objectives of our research are to find out:

1. Which recommender strategy (collaborative filtering or content-based) predicts user preferences better.
2. The impact of different similarity metrics on the performance of the content-based recommender.

We evaluated our models using the MPR evaluation metric. Figure 4 compares the average MPRs for the content-based method with three different similarity metrics and the collaborative filtering method.

A good recommender aims to achieve a lower MPR score which indicates that the user was recommended songs they were predicted to like (high listening counts) more often.

Our results empirically show that our collaborative filtering model (MPR of 48.65%) outperforms our content-based recommender system on all three similarity metrics (MPRs ranging from 56% to 57%). We can deduce that in the music recommender domain, using user-user similarity to recommend songs is a better strategy than using user-song similarity. Our finding is in line with our expectations and previous studies which support collaborative filtering over content-based approaches (Liang, Zhan, and Ellis 2015). Moreover, our collaborative filtering model does not suffer from the "cold-start" problem explained earlier because we trained our model on a relatively large dataset that includes user listening histories.

Note that both recommender approaches use different types of data; collaborative filtering uses user listening history while content-based filtering uses audio metadata. Because the quality of these two different sets of training data may differ, our comparison may not be fair. Our MPR values are also higher than expected which may be because our dataset does not have enough listening history for users on average. In our dataset, users listen to a median of 27 songs. As we have 770,000 songs in the dataset, this results in a very sparse matrix. Because of this limited song history, we did not have a lot of information to form our user profiles off of. With more data, we would be able to make more personalized song recommendations resulting in lower MPR values. Additionally, the sparsity of the matrix led the recommender to suggest many songs from our test set that the user had not listened to. These songs were not included in the MPR because the listening count is 0. However, they lowered the rank of other songs that were counted thus resulting in a higher overall MPR score.

Our experiment also showed that there was no statistically significant evidence that any similarity metric outperformed the others. However, the performance of the similarity metrics depends on the location and the clustering of users and songs in their vector space, so our finding cannot be generalized to every dataset.

Our collaborative filtering model is based on the work by Johnson (Johnson 2014). Due to computational limitations, we did not perform a grid search to tune all the hyperparameters simultaneously so our values may not result in the most optimal performance. In Johnson's paper, he was able to attain much lower MPR values with the same algorithm. This difference in performance is likely due to our different datasets. Johnson's dataset has 50,000 users and 10,000 items. In our dataset, we have one million users — 20 times more than his dataset — and 770,000 items of interest. His dataset has 5 users for every item whereas we have ~ 1.3 users for every item. By limiting his items to only the top 10,000 artists, he has a larger ratio of users to items and his dataset is denser than ours. Note that we compared user-song interactions while he examined user-artist interactions. This would result in larger listening counts in his dataset which in turn affected his hyperparameter values which explains why we got different values. In addition, his paper is based on data from Spotify users over a period of two years (2011-2013). As previously mentioned, users in our dataset listened to a median of 27 songs. In 2014, Spotify reported that the average user listens to 30 unique artists in a single week (Iqbal 2020). Obviously, his dataset has more information about user preferences and less sparsity which would help his model make much more accurate predictions. This explains how he was able to attain an MPR of less than 15% with the same collaborative filtering model.

According to Figure 4, we also found that Logistic MF performed better with fewer latent factors which supports the finding in Johnson's paper. Increasing the number of latent factors may not yield performance improvements because of the curse of dimensionality. As we increase the number of latent factors, the number of columns in our user and song matrices also increase. So we require more training samples in order to learn an optimal representation of the user and songs which eventually leads to overfitting the training data.

Using less latent factors is also beneficial because the recommender can recommend songs faster and use less memory. Some collaborative filtering models recommend songs by performing a nearest neighbour search over the song vector space in real-time. Since the memory used by these search algorithms is directly proportional to both the number of songs in the database and the dimensionality of the latent factor vectors, reducing the number of latent factors can significantly reduce the size of the data structure.

Conclusion

For many people, music is a huge part of their lives, and they spend hours curating playlists and listening to new songs. Good song recommendations can make or break a platform. We compared two commonly used recommendation strategies, collaborative filtering and content-based fil-

tering, to find which performs better. We also compared our content-based model on three different similarity metrics: Pearson Correlation, Cosine Similarity, and Euclidean Distance. When testing our recommender models, we excluded a subset of each user's song history, ran the algorithm, and evaluated the recommender based on the ranking of the excluded songs in our recommendation list and how much the user is expected to enjoy the recommendation (their listening count for the excluded songs). We quantified the quality of our recommendations by applying Mean Percentage Ranking (MPR), where a lower MPR indicates a better performing recommender.

We found that collaborative filtering was more effective than content-based filtering. Collaborative filtering yielded an average MPR of 48.65%, while content-based averaged 56.65% across the three similarity metrics, with no large performance difference between the similarity metrics.

In the future, we would use a grid search to tune our hyperparameters in Logistic MF simultaneously, as we did not have the computational time to try every combination on our full dataset. While we did find a local minimum for each parameter, it's possible that a different combination would yield a better result. It would also be interesting to repeat our analysis on different song features. While constructing the song feature dataset, we selected seemingly useful audio features or features that are not strongly correlated with other audio features. Some features that we excluded may provide useful information and have a positive impact on the MPR. We'd also like to try implementing a hybrid model using ensemble learning on our two recommender models and observe how different weights affect the result. A hybrid model benefits from the strengths of both models and lessens the effect of the "cold start" problem on a collaborative filtering model because the content-based model performs well even under minimal song interactions. Johnson's experiment on recommending artists only considered interactions with the 10,000 most popular artists (Johnson 2014). It would be interesting to apply a similar approach where we provide recommendations that only consider the most popular songs.

There are many exciting fields to apply recommendations to, and many directions to research further. This research is very important too: a great recommender system has huge benefits for both consumers and corporations.

References

[Bell, Koren, and Volinsky 2007] Bell, R.; Koren, Y.; and Volinsky, C. 2007. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 95–104.

[Bogdanov and Herrera 2012] Bogdanov, D., and Herrera, P. 2012. Taking advantage of editorial metadata to recommend music. In *9th International Symposium on Computer Music Modeling and Retrieval (CMMR)*, 618–632. Citeseer.

[Das et al. 2007] Das, A. S.; Datar, M.; Garg, A.; and Rajaram, S. 2007. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, 271–280. New York, NY, USA: Association for Computing Machinery.

[Green et al. 2009] Green, S. J.; Lamere, P.; Alexander, J.; Maillet, F.; Kirk, S.; Holt, J.; Bourque, J.; and Mak, X.-W. 2009. Generating transparent, steerable recommendations from textual descriptions of items. In *Proceedings of the third ACM conference on Recommender systems*, 281–284.

[Hu, Koren, and Volinsky 2008] Hu, Y.; Koren, Y.; and Volinsky, C. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, 263–272. Ieee.

[Iqbal 2020] Iqbal, M. 2020. Spotify usage and revenue statistics (2020).

[Johnson 2014] Johnson, C. C. 2014. Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems* 27:78.

[Lam, Pitrou, and Seibert 2015] Lam, S. K.; Pitrou, A.; and Seibert, S. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15. New York, NY, USA: Association for Computing Machinery.

[Li and Kim 2004] Li, Q., and Kim, B. M. 2004. Constructing user profiles for collaborative recommender system. In *Asia-Pacific web conference*, 100–110. Springer.

[Liang, Zhan, and Ellis 2015] Liang, D.; Zhan, M.; and Ellis, D. P. 2015. Content-aware collaborative music recommendation using pre-trained neural networks. In *ISMIR*, 295–301.

[Lim, Lanckriet, and McFee 2013] Lim, D.; Lanckriet, G.; and McFee, B. 2013. Robust structural metric learning. In *International conference on machine learning*, 615–623.

[Mnih and Teh 2012] Mnih, A., and Teh, Y. W. 2012. Learning label trees for probabilistic modelling of implicit feedback. In *Advances in Neural Information Processing Systems*, 2816–2824.

[Moore 2019] Moore, F. 2019. Music listening 2019.

[Rentfrow and Gosling 2003] Rentfrow, P. J., and Gosling, S. D. 2003. The do re mi's of everyday life: The structure and personality correlates of music preferences. *Journal of Personality and Social Psychology* 84(6).

[Ricci, Rokach, and Shapira 2015] Ricci, F.; Rokach, L.; and Shapira, B. 2015. Recommender systems: introduction and

challenges. In *Recommender systems handbook*. Springer. 1–34.

[spo 2020] 2020. Spotify users have spent over 2.3 billion hours streaming discover weekly playlists since 2015.

[Van den Oord, Dieleman, and Schrauwen 2013] Van den Oord, A.; Dieleman, S.; and Schrauwen, B. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*, 2643–2651.

[Vintch 2020] Vintch, B. 2020. What similarity metric should you use for your recommendation system?

[Zhong, Wang, and Jiao 2018] Zhong, G.; Wang, H.; and Jiao, W. 2018. Musiccnns: A new benchmark on content-based music recommendation. In *International Conference on Neural Information Processing*, 394–405. Springer.