Bachelor Thesis in Statistics, 15 ECTS

Music Recommendations

Approximating user distributions to address the cold start problem

Johan Hammarstedt

LUND
UNIVERSITY

Department of Statistics
Lund University
Fall Semester, 2019

Supervisor: Krzysztof Podgórski

Abstract

In today's data driven society the world is at a point of information overload. As people rely on Google for information and other platforms such as Netflix and Spotify for entertainment, the need for relevant filtering of content has never been higher. As a result, recommendation systems have seen a great surge in demand. One can divide the space of recommendation algorithms into primarily two approaches. In the context of music,a collaborative based one where underlying correlations between users dictate the model, and the content based approach which examines the more specific relationship each user has to the songs. This paper aims to highlight the issues many collaborative models face when there is a lack uneven amount of interactions with the songs; this is usually the case for less popular or new items. To address this, a content based approach is suggested based on music feature data with the goal to distinguish unique user distributions based on song characteristics. After evaluating this method against a popularity based baseline model, there was a small but not significant difference in the error. This suggested that there are a lot of room for improvement in the approximation of user distributions, leading to the conclusion that with more elaborative methods one could most likely expand upon this research and build strong recommendations based on the idea of probabilistic user distributions.

Keywords— Content-Based Recommendations, Collaborative Filtering, Matrix Factorization, User-Feature distribution

Acknowledgements

I would like to thank my supervisor Krzysztof Podgórski for our interactive discussions and his commitment to help me complete this paper. I am also thankful to Spotify and the researchers behind the Million Song Dataset for their excellent API's and accessible data. Finally I would extend my gratitude to my parents and my grandpa for all their support throughout my academic career.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

In today's data driven society we are at a point of information overload. The availability of information has created an environment where haystacks are everywhere and user's barely know how to spot the needle. This has generated a surge of recommendation systems with the task to filter relevant content for the user. Tech-companies such as Amazon, Spotify, Netflix and Youtube are constantly developing algorithms to identify user behavior and create relevant content for each consumer. "Recommendation systems are defined as decision making strategy for users under complex information environments" [11]. This paper aim to elaborate on the algorithms used in today's industry, more specifically the two approaches known as *collaborative filtering* and *content based filtering* which will be discussed in detail in the upcoming sections. Collaborative filtering aims to use the ratings provided by a network of users to make predictions for what to recommend, refereed to as user-to-user. On the other hand, a content based approach looks at the specific user and the attributes of each item she prefers, i.e user-to-item [15]. Today, collaborative filtering is the most popular technology used and has seen a lot of success due to the power of user profiling and big platforms such as Spotify, having 124 million paid subscriptions and 271 million active monthly users, all with their own unique profiles [1] [3].

The authors of the article "Current challenges and visions in music recommender systems research" [16] identifies many important particularities with music recommendations, and the following are especially relevant for this paper:

1. "Consumption behavior": A lot of music is consumed passively as background music,

which we pay no attention to. This can cause a problem to identify actual preference and implicit feedback, leading to false assumption about the users taste.

2. "Listening context": A user can have very different taste for any given situation. On one hand creating a calm playlist for studying but then also a death-metal album for working out. User may also change their preferences over time.

1.2 The cold start problem

A well known obstacle with many recommendation systems is the *cold start problem*, where the lack of information makes it difficult to create proper assumptions of user preference [10] [16]. Collaborative models tend to have a strong bias towards music with a lot of user interaction and thus favor more popular music. In the article "A collaborative filtering approach to mitigate the new user cold start problem" [6], the authors identifies three types of cold start problems:

1. New Community: Difficulties obtaining data when first creating a recommendation system environment. With a few users and bad recommendations, keeping them becomes difficult and further weakens the models.

2. New User: The problem of limited information from new users is one of the more difficult problems, which in some cases can be addressed by asking a new user to provide information upon entering the platform.

3. New item: Newly created songs or not popular ones lack interactions from users and will therefore have a hard time being labeled.

Another complication of music recommendations is *sparsity*: "the inverse ratio between given and possible ratings" [16]. With a lot of content available and millions of users, individuals will only be able to rate a fraction of available songs. This also leads to more unreliable recommendations [12].

Nevertheless, in order to address the cold start problem, a lot of research has been done on *Ensemble Based Recommender Systems*. The idea is to combine collaborative models

and content models by different methods and thus address their respective weaknesses with the others strengths [16] [8]. There are many approaches to combine these models such as computing weighed averages between predictions or switching between models in a scenario based setting [4]. As this paper mainly will elaborate on the use of the models separately, combining them to a hybrid would be an intriguing topic for further research.

1.3 Goal and scope of this paper

In this paper I aim to do the following:

1. Give an overview of the state of the art collaborative filtering approaches being used in the industry today, then create and evaluate a collaborative model.
2. Attempt to address the cold start problem (mainly the new item) recommendation systems face by proposing a content based solution.
3. Introduce a method to objectively evaluate the content model for each user.

The collaborative filtering model is a technique utilizing Matrix Factorization and is a well known approach within recommendation systems. The goal is to predict user ratings given other user behavior by exploiting hidden correlations between users and their respective ratings. Further details of this are provided in later sections of this paper (see 3.1.1).

As previously mentioned, it will be difficult for the above method to recommend songs that that many users have yet to interact with, known as the Cold start problems of new items. The main focus of this paper will therefore be on the content based approach and its evaluation. Content based methods do not require ratings of other users to recommend and instead extract features from the songs themselves. [16]. The paper suggests a probabilistic model based on identifying underlying user distributions. *The intuition is the rather simple idea that, given the set of song features, each user has their own distribution which would correspond to one's taste.* A user frequently listening to dance music would correspondingly see

the features clustering towards high scores in characteristic features of that genre, like dancability and energy. By approximating the feature distribution for songs that a user prefers to a multivariate Gaussian distribution (see 3.2.2=, and then computing the likelihood of other songs in the dataset, new songs with similar features would be recommended. Consequently, ignoring aspects such as popularity and other user preferences that are the main sources of bias when dealing with the new item problem.

Evaluating a music recommendation system presents many problems as it becomes difficult to objectively address each user's unique taste. A good recommendation for one might be poor for another. Furthermore, the content model will not output a predictive and comparable rating for each song but rather rank them in terms of likelihood that it fits the user distribution. This means that there are originally no true label to evaluate on. There are no natural translation between the likelihood of a song in a user distribution and the amount of times the user has listened to a given song. The paper therefore suggest an alternative approach where the goal is to compute the distance from each song to the user-distribution. Since the model will, given a certain amount of mixed songs, recommend strictly based on likelihood, one can compare how much the user's preferences matches the user distribution.

2. Spotify and EchoNest Datasets

There will be two data sets be used to build the models and make recommendations. The first is raw data obtained from Spotify's API. Due to user privacy, only public playlists are available here and no user data could be gathered. The dataset consists of 83 450 songs which are divided into 1453 public playlists. In addition to general info such as title, release date and artists, Spotify has also enabled a feature extraction for each song. This introduces the following variables, who are continuous if not otherwise mentioned. Further note that these values were later normalized from $[0, 1]$. [18]:

1. Danceability - how suitable a track is for dancing in range $[0, 1]$.
2. Acousticness - a confidence measure in range $[0.0, 1.0]$ of whether the track is acoustic.
3. Energy - represents a perceptual measure of intensity and activity in range $[0.0, 1.0]$.
4. Instrumentalness - predicts whether a track contains vocals or not from $[0.0, 1.0]$. Higher values indicate less presence of vocals.
5. Key - the key the track is in. Categorical using standard Pitch Class notation. -1 indicates no pitch detected.
6. Liveness - detects the presence of an audience in the recording, higher values indicate a higher probability that the track was performed live, range: $[0, 1]$.
7. Loudness - the overall loudness of a track in decibels (dB), usually in range $[-60, 0]$.
8. Mode - the modality (major or minor) of a track (binary), the type of scale from which its melodic content is derived.

9. Tempo - the overall estimated tempo of a track in beats per minute (BPM).

10. Valence - a measure in range $[0.0, 1.0]$ describing the musical positiveness conveyed by a track.

Due to the lack of user data in the Spotify dataset another one was obtained which is a subset of the *Million Song Dataset* (MSD) from the *Echo Taste Profile subset* [5] [19]. The MSD is a very extensive dataset and widely used within research on recommendation systems since it has real and authenticated user data. Since the processing of the full 280 GB dataset and even the full Echo subset would have required further data management, it is outside of the scope of this paper, and the final data gathered consists of $2 \cdot 10^6$ entries, containing 10000 unique songs and 76353 unique users. Each entry holds:

1. User ID

2. Listen Count

3. Song ID

4. Title

5. Release Date

6. Artist Name

Table 2.1: General data information

	Unique songs	Unique users	Features	Total Rows
EchoNest	10000	76353	User listen count	2000000
Spotify Set	83 450	No user data	Song features	83450

Table 2.1 displays the initial information obtained from each dataset. Using Spotify's query method in their web API, the feature extraction was made for the EchoNest subset as well, leaving 1550027 rows. Since the focus mainly will be on the new item problem, and not

the new user problem, users who had listened to fewer than ten songs were then removed in order to better estimate the user-feature-distributions, resulting in 1347089 entries. The EchoNest set will be used to build the models since it contains the user information needed. The recommendation library can then be expanded by the songs in the Spotify Set.

Figure 2.1 displays the distributions of the different features for users who have listened to more than ten songs. As can be observed, some are Gaussian-like and will fit the generated distribution properly while others will not. For this model binary and categorical values presents an issue, therefore the categorical *key* and the binary *mode* will be removed for the approximation.

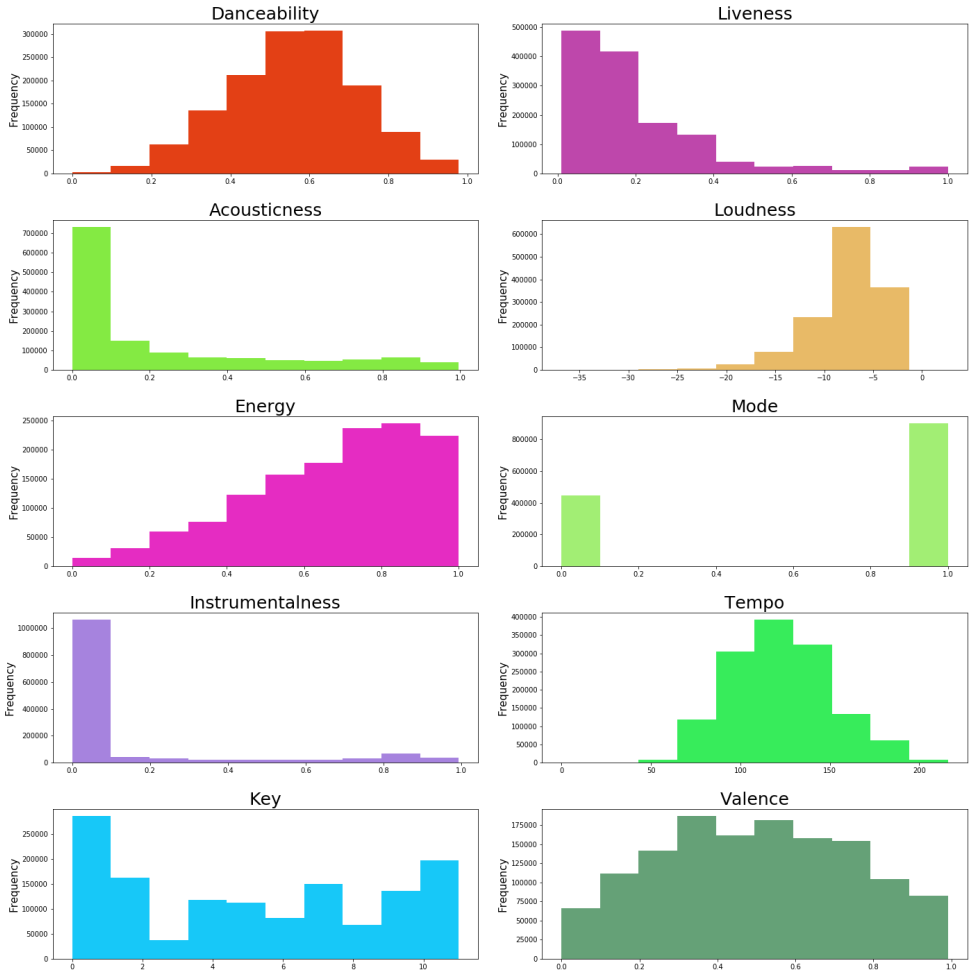

Figure 2.1: Distribution of full dataset with only users who has listened to more than ten songs

Furthermore, figure 2.2 displays the co-variances between the features for a sample user. It also becomes clear that a transformation is required to better approximate each user-feature distribution. Due to the limitations of non negative inputs in a *Box-Cox transformation* this paper suggests a *Yeo-Johnson transformation* [21] which is explained further in detail later.

Table 2.2 contains the most popular songs by users and how many percentage that accounts for in the aggregated listen count and table 2.4 is the cumulative percentage of n songs.

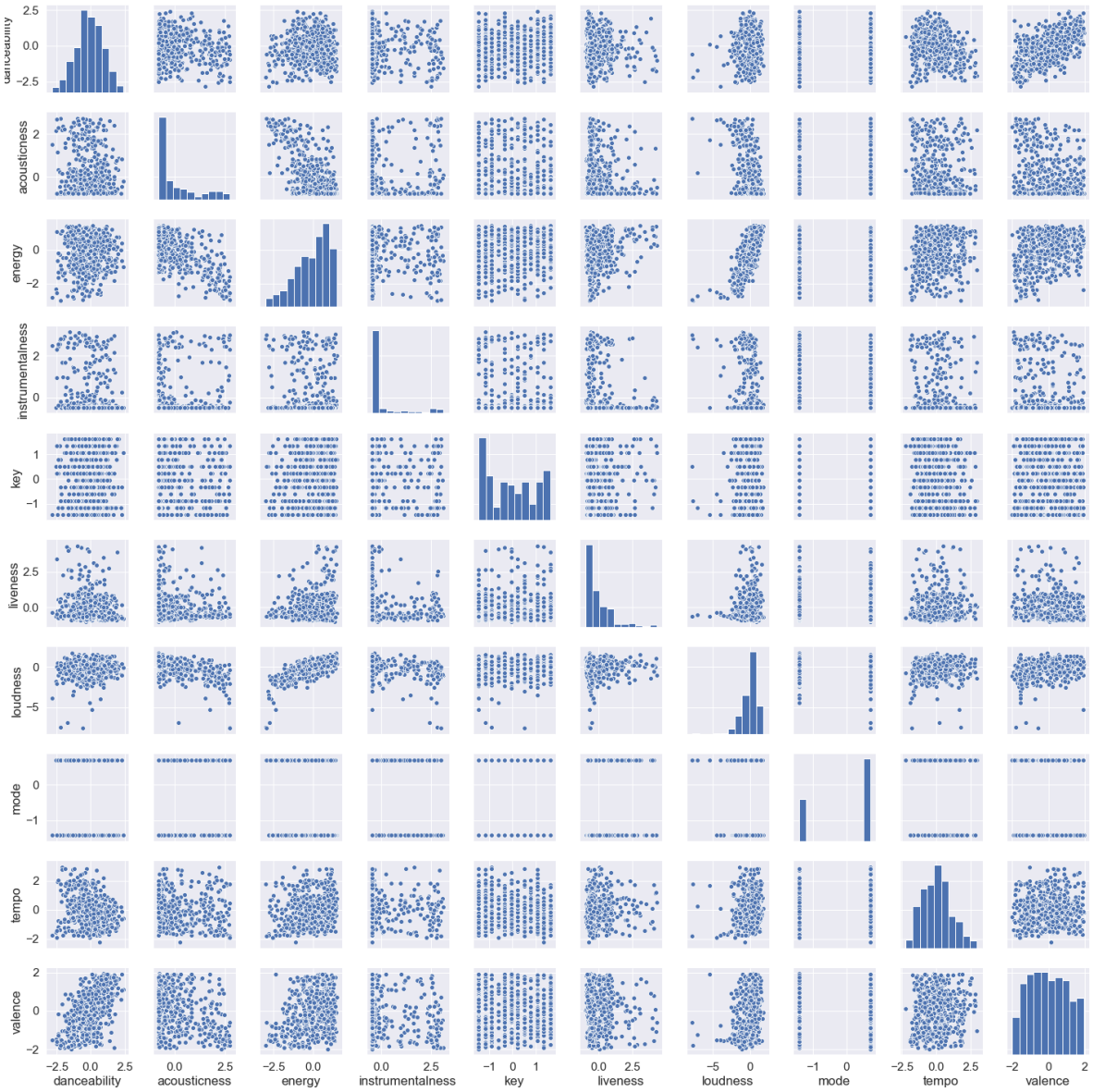

Figure 2.2: Raw sample user feature data

Table 2.2: Top-10 most listened songs by users in dataset

	Title	**Listen Count**	**Percentage of the aggregated listen count**
1	Undo	7032	0.453614
2	You're The One	6729	0.434068
3	Revelry	6145	0.396396
4	Secrets	5841	0.376786
5	Fireflies	4795	0.309312
6	Tive Sim	4548	0.293378
7	Use Somebody	3976	0.256480
8	Drop The World	3879	0.250223
9	Marry Me	3578	0.230806
10	Canada	3526	0.227452

Table 2.3: Cumulative Percentage

Songs	10	20	30	40	50
Cumulative Percentage	3.4	5.4	6.9	8.3	9.6

As displayed in Table 2.3, the first 50 songs account for 10% of the total 155027 for the full dataset, where the top 10 is 3.4%. Figure 2.3 then illustrates the distribution of the listen count. The x-axis displays how many times a single song has been listened to mapped to the y-axis by the amount of users. As seen roughly in Figure 2.3, the majority of listen counts are users who have listened to a song a fewer times. This is also confirmed by the listen count distribution in Table 2.4. This can present problems, especially for the collaborative model, since the recommendations in a environment could lead to overfitting and minimize the error by rating all songs as one.

Table 2.4: Distribution of Listen Counts

Listen Count	1	2	3	4	5	6	7	8	9	10	<10
Total (thousands)	890	242	110	63	71	35	23	17	12	14	71
Total (%)	57	16	7	4	4.5	2.2	1.5	1.1	0.8	0.9	4

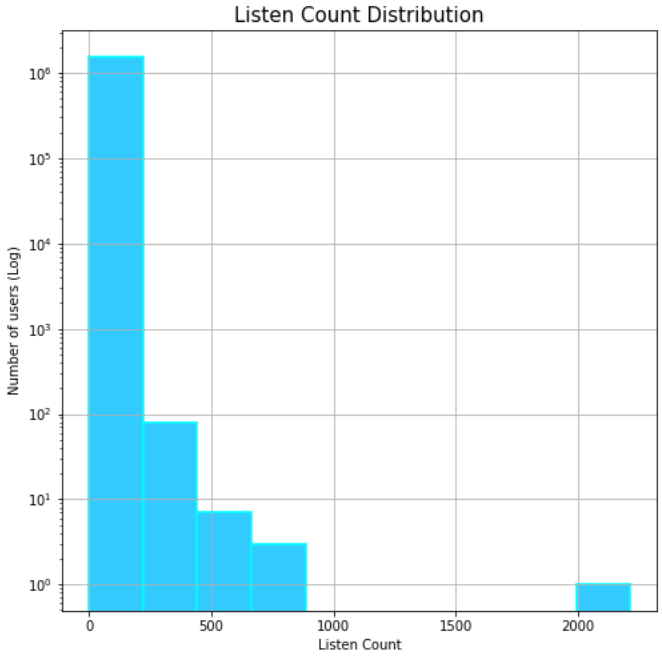

Figure 2.3: Listen Count

After removing the users with less than ten songs, this resulted in 1347089 entries with 42288 unique users. Table 2.5 displays the new distribution which is similar to Table 2.4 and thus will not change any other aspect of the analysis.

Table 2.5: Distribution of Listen Counts for users with >10 songs.

Listen Count	1	2	3	4	5	6	7	8	9	10	<10
Total (thousands)	772	212	97	56	60	30	20	15	11	12	608
Total (%)	57	15.8	7.2	4.1	4.4	2.2	1.5	1.1	.8	.9	4.4

3. Methodology

The methods used in this paper are divided into several subsections covering the statistical methods and the proposed implementation. The more detailed mathematical notations of optimization using stochastic gradient decent and the idea behind regularization can be found in the Appendix (see A).

3.1 Collaborative filtering - a model based method

Collaborative filtering usually falls into two types of techniques, Memory-Based and Model-Based. As the name suggests, Memory based stores a lot of information which it then maps to each user. This is effective but in many cases not scalable enough for a substantial amount of users. The model based technique implies that a generalized model will be implemented to predict good songs. The focus of the collaborative section will be on model based approaches and for further readings on memory based approaches the reader is referred to [11]. To better explain the intuition behind these methods, the listen count is being defined as rating.

3.1.1 Matrix factorization models - Singular Value Decomposition

As previously mentioned, many of the other techniques have difficulties in terms of scalability. Latent factor models or matrix factorization models uses a dimensionality reduction technique called matrix factorization to efficiently create a model which will be introduced below. The full approach is known as singular value decomposition (SVD). So given a matrix containing all users and all songs, we approximate the scalar product between these vectors and thus fill in the 'blanks' as displayed in Table 3.1.

Table 3.1: User-Song Matrix (R)

	S1	S2	S3	S4
User 1	3	1		3
User 2	1		4	1
User 3	3	1	1	3
User 4		3	5	4

Matrix R displayed in Table 3.1 is what one wish to compute. Realistically this matrix is missing values since most users have not listened to all songs. The goal is to predict these missing entries and recommend those with high scores to respective user. The problem is that matrix R quickly gets very large, for the data set consisting of 10000 songs and 76353 users it would be a 76353×10000 matrix. This is dealt with by matrix factorization. The idea is to exploit the fact that there exists high correlations between the rows and columns in the matrix.

Now we introduce a user matrix U and an song matrix S. The user matrix is of size $N_u \times N_h$ where N_u is the amount of users and N_h is the amount of so called hidden features we choose (the idea behind these features will be explained in the next paragraph). The song matrix S is similar and of dimension $N_s \times N_h$ where N_s is the number of songs. The goal of singular value decomposition is to compute:

$$U \cdot S^T \approx R \tag{3.1}$$

In words, finding the optimal matrix multiplication between two feature matrices that approximates the whole user-song matrix R the best. This is done by finding hidden features that exploits the correlations as previously mentioned. These hidden features are not intuitively easy to explain since they are generated to find the best fit and might be the genre of the song, the popularity or something uninterpretable. [15].

Table 3.2: Factor Matrix example - Songs

	Hidden Feature 1 (e.g. Genre)	Hidden Feature 2 (e.g. key)	...	Hidden feature N_h
Song 1	0.4	0.6	...	0.5
Song 2	0.2	0.8	...	0.2
...	...	...	...	..
...	...	...	...	..
Song N_s	0.5	0.3	...	0.3

The user factor matrix looks similar with users $\{1, 2, ..., N_u\}$ instead of songs. In the graph above each song j has its own vector that suggests how well the song j possesses these features. In the user matrix, corresponding interpretation for user i would be the level of interest the user has shown in songs with high values for given features [15]. An example would be that a user has high values of hidden features that could corresponds to Pop or Dance music, meaning he tends to rate those songs high.

To quote from C.Aggrawal's book on recommender systems [4], the inner dot product of matrices U and S, $u_i \cdot s_j$ "captures the interaction between user i and item j, the overall interest of the user in characteristics of the item." The vectors u_i and s_j thus both has dimension $1 \times N_h$. From 3.1 it follows that rating $r_{i,j}$ can be approximated by the dot product of the jth song vector and ith user vector in their respective matrix. So the basic prediction for the rating $r_{i,j}$ on song j by user i would be:

$$r_{i,j} = u_i^T \cdot s_j. \tag{3.2}$$

In the models used for this paper we will also incorporate baseline predictors as suggested by [4] on their model for Netflix recommendations. The baseline predictors are meant to capture systematic biases, like some users giving too high ratings. However, our ratings are based on listens which are less likely to be biased in such a way (or a user could listen to an extensive amount of music compared to the rest and boost a song too much). To explain these baselines used for dealing with bias, let μ be the average rating for all songs, v_i and w_j

be the observed standard deviation of user i and song j from the average. Then the baseline prediction $b_{i,j}$ for an unknown rating $r_{i,j}$ is:

$$b_{i,j} = \mu + v_i + w_j \tag{3.3}$$

Hence, to find the baseline for ex Happy with Pharell Williams one would do as follows: If the average rating (listen count) for all songs was 3, but given the popularity of the song users listen to it on average 1.5 times more. Then user X barely listen to music at all, on average he only listens to any song 1 time, 2 less than average. The baseline would therefore be $b_{i,j} = 3 + 1.5 - 2$.

The updated rating for song j by user i will accordingly be computed by:

$$\hat{r}_{i,j} = b_{i,j} + u_i^T s_j = \mu + v_i + w_j + u_i^T s_j. \tag{3.4}$$

We wish to estimate v_i, w_j, u_j, s_i, using a regularized (see A.2) squared error as our cost function, as recommended by [4]. We first define a set with the known rating pairs (i, j) as η. So: $\eta = \{(i, j)|r_{i,j} \text{ is known}\}$. Then we minimize the cost function:

$$\min_{b^*,u^*,s^*} \sum_{i,j\in\eta} (r_{j,i} - b_{i,j} - u_j^T s_i)^2 + \lambda(v_i^2 + w_j^2 + ||s_j||^2 + ||u_i||^2). \tag{3.5}$$

The minimization is done by stochastic gradient decent [A.2.1]. A very successful SGD approached created by Funk [17] is used to update the parameters:

- $w_j := w_j + \gamma(e_{i,j} - \lambda w_j)$
- $v_i := v_i + \gamma(e_{i,j} - \lambda v_i)$
- $u_i := u_i + \gamma(e_{i,j} \cdot s_j - \lambda u_i)$
- $s_j := s_j + \gamma(e_{i,j} \cdot u_i - \lambda s_j)$

The error term is defined by computing the prediction $\hat{r}_{i,j}$ and then comparing it to the true rating $r_{i,j}$ which is possible since we are optimizing a training set with the true ratings known in η [15]:

$$e_{i,j} = r_{i,j} - \hat{r}_{i,j}, (i, j) \in \eta. \tag{3.6}$$

3.2 Content Based filtering methods

Content based models are using the analysis between user item rather than user to user. The approach suggested in this paper originates from the idea that each user has a corresponding feature distribution which, independent of popularity and previous ratings, can be used to make more individual recommendations. The features are transformed by Yeo-Johnson transformation with the goal to better fit the approximated multivariate normal distribution for each user and then the likelihood of each song is computed to recommend the songs fitting the distribution the best.Below follows the methodology necessary to perform these computations. The last section contains the proposed method to evaluate these results.

3.2.1 The Yeo-Johnson Transformation

Examining the initial data it was clear that some transformation was necessary to better allow for normality. The *Yeo-Johnson transformation* was suggested by In-Kwon Yeo and Richard A. Johnson as a way to deal with the limitations of the Box-Cox transformation, as it only allows for strictly positive numbers [22]. The transformation was chosen as a way to deal with potential skewness and allow for better analysis. The transformations on a feature vector X with parameter λ is given by:

$$\psi(\lambda, x) = \begin{cases} ((x+1)^{\lambda} - 1)/\lambda, & \text{if } \lambda \neq 0, x \geq 0 \\ \log(x+1), & \text{if } \lambda = 0, x \geq 0 \\ -\left[(-x+1)^{2-\lambda} - 1)\right]/(2-\lambda), & \text{if } \lambda \neq 2, x < 0 \\ -\log(-x+1), & \text{if } \lambda = 2, x < 0 \end{cases} \tag{3.7}$$

The transformation is used to better allow for normality in the feature data and lambda is a hyper-parameter used for selecting the set of transformation on X. The optimal lambda (λ) for each feature is determined by the following method [2]:

$$L(X; \lambda, \sigma) = \frac{N}{2}\log(\hat{\sigma}^2) + (\lambda - 1)\sum_i \text{ sign } (x_i)\log(|x_i| + 1).$$

3.2.2 Multivariate Gaussian Distribution

The multivariate normal distribution is said to be defined when the random variable $Z = [Z_1, ..., Z_N]^T$ with mean $\mu \in R^n$ and the co-variance matrix $\Sigma \in S^n_{++}$ have the probability density function:

$$p(z; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right)$$

This is written as $Z \sim \mathcal{N}(\mu, \Sigma)$. The co-variance matrix Σ, for any random vector Z is defined as:

$$\Sigma = E[(Z-\mu)(Z-\mu)^T] = E[ZZ^T] - \mu\mu^T.$$

The co-variance Σ for the multivariate Gaussian distribution, it is required to be *positive definite*, meaning that all its corresponding eigenvalues are positive.

For the k given features, each user-feature distribution will be approximated with a $k \times 1$ vector μ_i and $k \times k$ matrix Σ_i. Since songs has different listen counts, both the co-variance matrix and μ_i will be weighted accordingly. Then, the likelihood of each song is calculated for each user and the top ten recommendations will be given by the ten with the highest likelihood, given that the user has not listened to the song previously.

3.3 Proposed evaluation method

The following definitions are introduced below to be referred to in this section:

- Let N_u be the total amount of users and $user_i$ denote the i-th user where $i = \{0, 1, ...N_u\}$.
- Let N_s be the total amount of songs and $song_j$ refer to the j-th song where $j = \{0, 1..., N_s\}$.
- Let k be the number of features evaluated.
- μ_i, Σ_i: The $k \times 1$ weighted average vector and the $k \times k$ weighted co-variance matrix for $user_i$.

- $\mathcal{S}_i$: The set of songs in the full dataset which $user_i$ has listened too.
- $\mathcal{L}_{i,j}$: The number of times $user_i$ has listened to $song_j$ assuming that $song_j \in \mathcal{S}_i$.
    1. $\mathcal{L}_i$: total listen count for $user_i \in \mathcal{S}_i$.
       $\mathcal{L}_i = \sum_{j=0}^{\mathcal{S}_i} \mathcal{L}_{i,j}$, $j \in \mathcal{S}_i$
- $\mathcal{F}_j$: The $k \times 1$ vector containing the corresponding k features for that song, so each $\mathcal{F}_j$ consists of k numerical values representing the characteristics of that song.
- $\mathcal{P}_i$: Refers to the overall listen count by all users in $\mathcal{S}_i$. $\mathcal{P}_{i,j}$ is then the overall listen count for song j by all users in $\mathcal{S}_i$.

Due to the difficulties of measuring model performance for a user on songs he/she has not given any information on, the evaluation will be performed on $\mathcal{S}_i$.

For $user_i$, the idea is to compute the *Mahalanobis squared distance* ($\mathcal{M}^2$) from each song vector $\mathcal{F}_j$ to μ_i, thus the center of $user_i$'s approximated user distribution. Then the aim is to examine the distribution of these distances in relation to the listening preferences for $user_i$. Mahalanobis Squared Distance is defined as follows: Let $\mathcal{F}_j$ be a $k \times 1$-dimensional vector corresponding to a song with k- features. Let μ_i, Σ_i be the mean vector and co-variance matrix for user i between k features. Then the Mahalanobis squared distance between $\mathcal{F}_j$ and μ_i is defined as [14]:

$$\mathcal{M}^2 = (\mathcal{F}_j - \mu_i)^T \Sigma^{-1} (\mathcal{F}_j - \mu_i). \tag{3.8}$$

As Σ^{-1} is the inverse of the co-variance matrix it is assumed that the matrix is invertible and positive definite. As a result, it is then possible to show that $M^2 \sim \chi^2(k)$, for k degrees of freedom corresponding to the number of song features[1].

By using that $\mathcal{M}^2 \sim \chi^2_k$ we compute the decentiles for that distribution and construct intervals to check the total listen count for that user in each decentile interval. By definition, the distribution of the songs from our likelihood model will be 0.1 in each decentile. Let $\mathcal{L}_{i,m}$

[1]The actual derivation is out of scope of this paper, see [20]

be the listen count for $user_i$ in decentile m. The squared error for $user_i$ in decentile m is then then defined as:

$$SE_{i,m}^{Model} = \left[0.1 - \frac{\mathcal{L}_{i,m}}{\mathcal{L}_i}\right]^2 \tag{3.9}$$

The mean squared error for $user_i$ and the total for all users then becomes:

$$MSE_i^{Model} = \frac{1}{10}\sum_{m=1}^{10} SE_{i,m}^{Model}, m = \{1, ..., 10\} \tag{3.10}$$

$$\implies MSE^{Model} = \frac{1}{N_u}\sum_{i=0}^{N_u} MSE_i^{Model} \tag{3.11}$$

For comparison, a popularity model is also introduced. Defined as follows:

$$SE_{i,m}^{Popularity} = \left[0.1 - \frac{\mathcal{P}_{i,m}}{\mathcal{P}_i}\right]^2 \tag{3.12}$$

$MSE_i^{Popularity}$ and $MSE^{Popularity}$ are determined similarly as (3.12) and (3.13).

For example, examining 8 features. Given the song vectors $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ for $user_i$ with a respective listen count of $10, 12, 20$. mean vector μ_i and co-variance matrix Σ_i. The Public Count for the songs are $(1020, 950, 2000)$. All $\mathcal{M}^2$ values were then calculated to $6.55, 6.6, 3.8$. For a chi square distribution with 8 degrees of freedom: both $\mathcal{F}_1$ and $\mathcal{F}_2$ belongs to P_{50} and $\mathcal{F}_3$ is in P_{20}. Thus:

$$SE_{i,2}^{Model} = (0.1 - \frac{20}{10 + 12 + 20 + rest})^2 \approx 0.09$$
$$SE_{i,5}^{Model} = (0.1 - \frac{22}{10 + 12 + 20 + rest})^2 \approx 0.11$$
$$\implies MSE_i^{Model} = \frac{1}{10}\left[0.09 + 0.11 + \sum_{m=1}^{10} SE_{i,m}^{Model}\right], m \neq \{2, 5\}$$

Where the $rest$ denotes the remaining listen count for $user_i$.

The step-by-step evaluation is done as follows:

1. Compute all decentiles for χ^2_k.
2. For each user:
   (a) Randomly split the dataset into a training set of 80% and a test set of 20%.
   (b) Approximate the multivariate Gaussian user distribution (μ_i, Σ_i) on the training set.
   (c) Compute the *Mahalanobis distance* for each song vector $\mathcal{F}_j$ in the test set.
   (d) For all decentiles, determine $\mathcal{L}_{i,m}$ and $\mathcal{P}_{i,m}$
   (e) Calculate SE_i^{Model} and $SE_i^{Popularity}$
3. Calculate MSE^{Model} and $MSE^{Popularity}$

Given that the whole dataset is transformed to approximately be normally distributed, the following key point is made regarding the aim of this method:

> *The goal off the evaluation method is to test if each user distribution is distinct enough to make personalized recommendations based on feature data alone.*

4. Evaluation and Results

4.1 Collaborative filtering Results

To create meaningful recommendations the listen count was scaled down to 'ratings' from $0 - 10$.The initial scaling was just performed to merge values larger than 10. So the rating for $song_j$ by $user_i$ would be computed:

$$\mathcal{R}_{i,j} = \min(\mathcal{L}_{i,j}, 10)$$

Table 4.1 contains the after training the model on first a sample of 100000 entries with default values and a grid-search on number of epochs, hidden features,the overall learning rate and overall regularization term. The optimal parameters where then used to train the full dataset which is also displayed below.

Table 4.1: SVD results

		Hidden Features	Epochs	LR	RT	MSE	MAE
Sample	Default	100	20	0.005	0.02	6.243	1.816
	Gridsearch	100	25	0.03	0.6	6.23	1.815
Full	Gridsearch	100	25	0.03	0.6	4.8	1.4

LR: Learning Rate, RT: Regularization term

Examining the sample results, one can see that the after grid-search, the model did slightly better after training for 5 more epochs with a different learning rate and regularization term.

Given more data on the full training set the model improves further with the optimal grid-search parameters from the sample. The final precision and recall scores for a five fold cross-validated model was:

$$PR_{All} = \frac{\text{Recommended items that are relevant}}{\text{Recommended items}} = 0.845$$

$$RC_{All} = \frac{\text{Recommended items that are relevant}}{\text{Relevant items}} = 0.594$$

However, given the uneven distribution of the data these results might be misleading. Table 4.2 shows the best results for the full model. As seen the highest scores are all ones which matches are assumptions given the highly skewed data. The common trend is that all songs have all been 'rated' by a large amount of users with an average P_i of 368.9 and I_i of 85. Although some might be more of lucky guesses with a low P_i and also a not very active user corresponding to a low I_i value.

Table 4.2: Best Predictions for SVD model

	User ID	Song ID	TR	Pr	I_i	P_i	Error
1	54a3...	spotify:track:0kXeKglrzFU3w5J9nQWX0N	1.0	1.0	113	1360	0.0
2	3cec...	spotify:track:56mAv6TVHL4QXcD4B4Ezvg	1.0	1.0	178	468	0.0
3	4a25...	spotify:track:7BukHDlNfu4Ql7NATH6YIz	1.0	1.0	42	61	0.0
4	c6dd...	spotify:track:57PqYPpY9sB8IQlRlnDVnN	1.0	1.0	145	171	0.0
5	9442...	spotify:track:1pJS4rS12iA5MryQSAP2kQ	1.0	1.0	70	268	0.0
6	90ee...	spotify:track:5UWwZ5lm5PKu6eKsHAGxOk	1.0	1.0	22	814	0.0
7	67db...	spotify:track:2GGnAVuaaKklRJ67QZClzW	1.0	1.0	21	81	0.0
8	b075...	spotify:track:6Y6f7LSvHxUA61ItYiSMKE	1.0	1.0	118	118	0.0
9	c766...	spotify:track:029O4HWI1pVXLfFdfQd1Jb	1.0	1.0	130	104	0.0
10	72e5...	spotify:track:3gNynXzWWUBDm9u4FLywaC	1.0	1.0	11	244	0.0

I_i: the set of songs listened to by user i

P_i: The set of all users who have listened to the song

TR: True Rating, Pr: Prediction

Looking further into one of the top predictions, the left histogram in in Figure 4.1, one can see that the majority of listenings are one, which then gives the model a lot of incentive to rate this song accordingly.

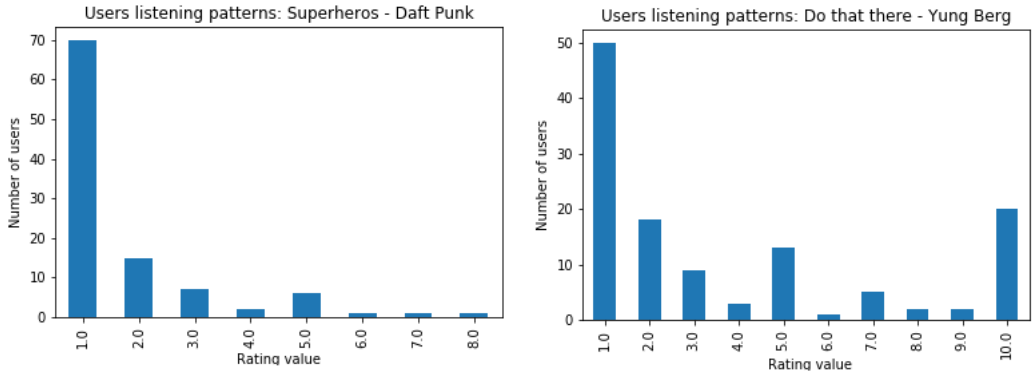

Figure 4.1: Listening pattern prediction

Examining the worst predictions in table 4.3 the trend is also very clear as the maximal error will inevitable be when a low prediction corresponds to a high true rating and vice versa. All songs displayed here also have a high average $\bar{P}$ of 1020. The average $\bar{I}$ of 69 is also lower than for the best predictions, implying less information and thus harder to predict. Since most songs in the dataset are only listened to once we will examine one of the songs that was predicted a high score.

Table 4.3: Worst predictions for SVD model

	User ID	Song ID	TR	Pr	I_i	P_i	Error
1	87f7...	spotify:track:1NhPKVLsHhFUHIOZ32QnS2	10.0	1.0	53	3634	9.0
2	b8f6...	spotify:track:2vO1wr5wIEHqQmY4jWbuhi	10.0	1.0	158	397	9.0
3	06c4...	spotify:track:1R6lhY5PqoxQJU5hsMDvjg	10.0	1.0	130	488	9.0
4	3da9...	spotify:track:2cOCunrzyHVpTrTwSIKRbt	1.0	10.0	65	195	9.0
5	567b...	spotify:track:2CR62nxWDw8ZRmJLCpm5PD	1.0	10.0	50	122	9.0
6	e647...	spotify:track:3MJdjsfekFs4kh04g2l6Zg	10.0	1.0	12	1518	9.0
7	2a1f...	spotify:track:43OjswTQMkuvQQEP38Roxl	10.0	1.0	32	64	9.0
8	4e6d...	spotify:track:11LmqTE2naFULdEP94AUBa	10.0	1.0	138	378	9.0
9	3604...	spotify:track:05NMSR0sSrCZxTHDRV415A	10.0	1.0	39	2637	9.0
10	8f8c8...	spotify:track:31I3Rt1bPa2LrE74DdNizO	10.0	1.0	14	335	9.0

I_i: the set of songs listened to by user i
P_i: The set of all users who have listened to the song
TR: True Rating, Pr: Prediction

The right histogram in 4.1 shows the song corresponding to index 5 in table 4.3. Again, the majority has listened to the song once but here a lot of users has also listened to the song more than 10 times and thus the model makes the prediction on the wrong side of the spectrum.

4.2 Content Based Results

The figures in Appendix B displays the features after the Yeo-Johnson transformation. Several features are improved slightly to better fit a Gaussian distribution but many of them, especially instrumentalness and acousticness (see B.2) are still facing problems due to its non-symmetric properties.

Figure 4.2 shows the same user with respective feature scatter plot but now with the transformed values. As mentioned in the data exploration section (see 2), *key* and *mode* were removed to better allow for the Gaussian approximation. Furthermore, all values that account for the users top ten most listened songs are plotted in terms of their listen count. In addition, the top ten recommendations from the model are marked green in each plot. The recommendations tend to cluster around the center together with the top songs which indicate that the likelihood estimation will give recommendations that most likely will correspond to the user distribution and by assumption then be good recommendations. Acousticness and instrumentallness still have difficulties to fit the distributions but the recommendations still lay relatively close to them to the center.

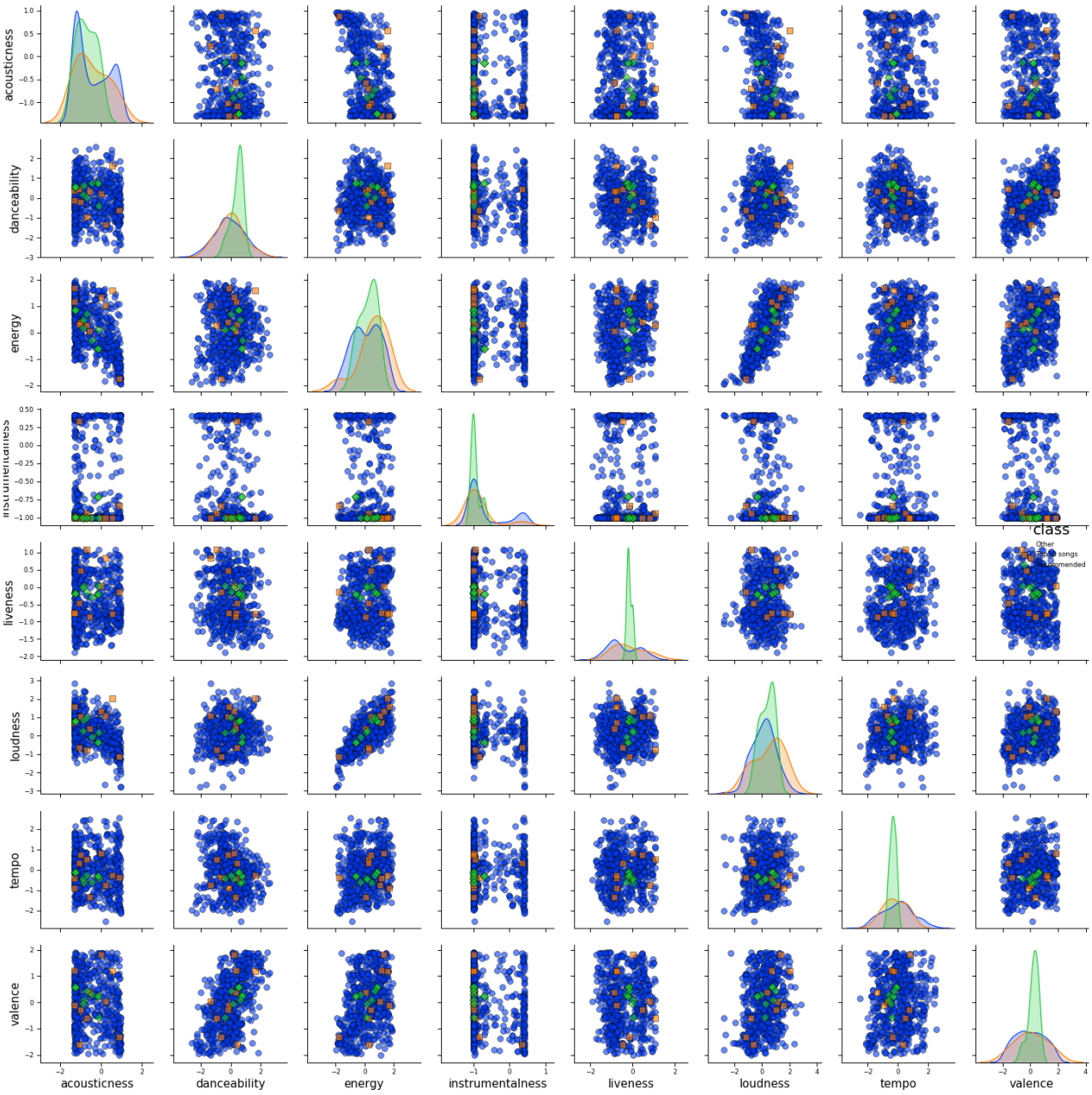

Figure 4.2: Example of transformed variables with Recommendations. The diagonal displays the distribution for that user's features. Green: Recommended songs. Orange: The distribution of the users top ten songs. Blue: the rest of that user's songs.

Table 4.4 displays the overall results. As seen in the table, the model fails to outperform the baseline over all the 35000 users. One important note is that the the users are sorted on the number of songs in descending order. As seen, the difference is increasing between each interval which calls for further analysis regarding relationship between the number of songs (listen count) and the evaluation.

Table 4.4: Overall Results

	MSE Likelihood Model	MSE Popularity Model	Difference
0-5000	0.0145	0.0098	0.0047
5000-10000	0.0276	0.0131	0.0145
10000-15000	0.0389	0.0131	0.0145
20000-25000	0.0493	0.0117	0.0375
20000-25000	0.0603	0.0098	0.0505
25000-30000	0.0699	0.0067	0.0633
30000-35000	0.09	0.0	0.09
Overall	0.0450	0.0088	0.0361

Figure 4.3 displays this relationships further, showing the users listening information in relation to the error. Again since the users are sorted from the most amount of songs and highest listen count it can clearly be seen that the error increases linearly for the model in absence of enough information.

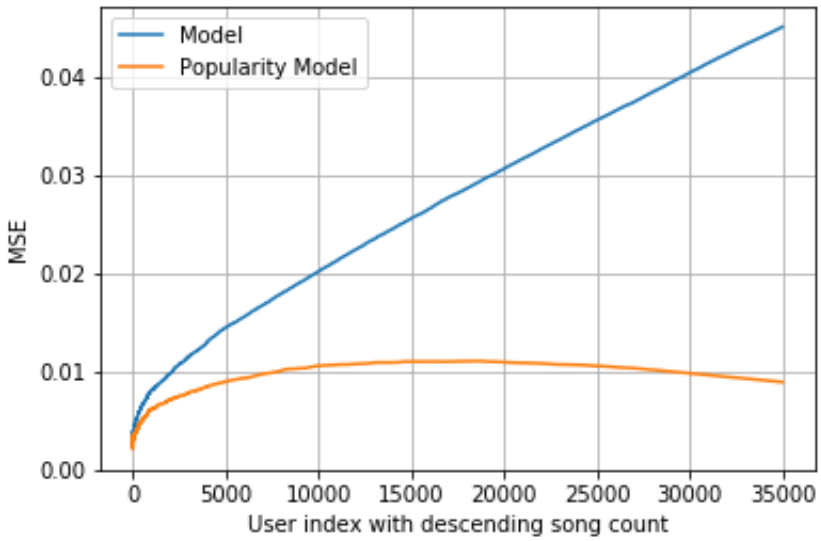

Figure 4.3: Error in relation to user information

Figure 4.4 evaluates this relationship further to determine if the strong linear trend comes mainly from a lack of unique songs or listen count. Each dot represents a user with its respective error, listen count and unique songs. As shown, the number of unique songs strongly determine the error. This is expected from the definition of the evaluation method since a decentile cannot have any listen count if there is no song there to begin with. This is discussed further in the later sections of this paper.

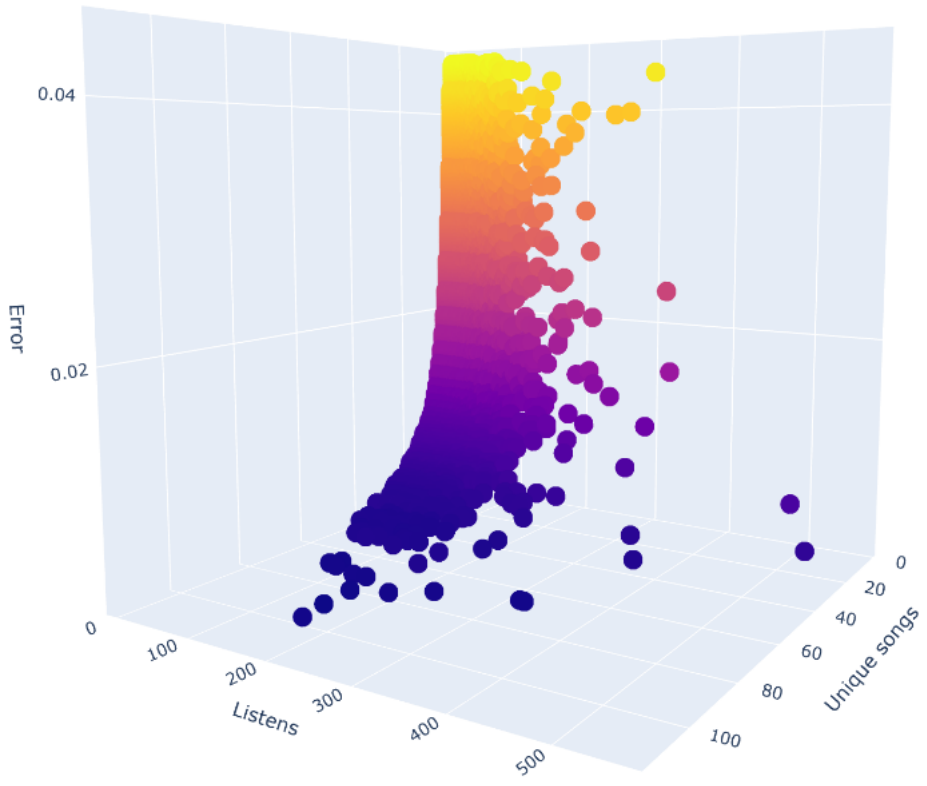

Figure 4.4: Likelihood model: Relationship between Error, Listen Count and Unique songs

4.2.1 Results on users with more songs

Table 4.5 shows the first ten users average results after ten cross validations. The overall error of the model has dropped significantly as a result of better on average approximations due to more songs. Here the model outperforms the baseline with a decreased MSE of 7.2% when examining the mean for both of them, the differences for all the 100-pairwise values is, however not significantly different from zero with a t-value of 0.51.

Table 4.5: Results for users with most songs after 10 cross-validations

User	MSE_model	MSE_public	Listen Count	Songs (Test Set)	Difference
0	0.00575	0.00562	556	113	-0.00012
1	0.00177	0.00153	196	106	-0.00025
2	0.00181	0.00284	178	96	0.00103
3	0.00248	0.00137	182	89	-0.00112
4	0.00179	0.0035	227	88	0.00171
5	0.00206	0.00105	165	84	-0.001
6	0.00188	0.00274	264	83	0.00086
7	0.00246	0.00247	178	83	1e-05
8	0.00185	0.0014	129	81	-0.00044
9	0.00158	0.00235	116	80	0.00076
Mean	0.0023	0.00248	219	90	0.00014

Table 4.6 is an example of one user's decentile distributions. As shown, the model is fairly accurate since the mean user frequency, which can be interpreted as the true label, is close to 0.1. Here the model manages to outperform the popularity baseline model with an average error of 0.0011 compared to 0.0033.

Table 4.6: Example of a decentile distribution for one user

User	User LC	Public LC	Model Freq	User Freq	Public Freq	Model SE	Public SE
P_{10}	30	7773	0.1	0.13393	0.27898	0.00115	0.02104
P_{20}	15	986	0.1	0.06696	0.03539	0.00109	0.001
P_{30}	29	2246	0.1	0.12946	0.08061	0.00087	0.00239
P_{40}	31	3338	0.1	0.13839	0.1198	0.00147	0.00035
P_{50}	20	3694	0.1	0.08929	0.13258	0.00011	0.00187
P_{60}	25	2566	0.1	0.11161	0.0921	0.00013	0.00038
P_{70}	15	1685	0.1	0.06696	0.06048	0.00109	4e-05
P_{80}	10	1157	0.1	0.04464	0.04153	0.00306	1e-05
P_{90}	25	1074	0.1	0.11161	0.03855	0.00013	0.00534
Mean	22	2931	0.1	0.098	0.1052	0.0011	0.0033

LC: Listen Count ($\mathcal{L}_i$),
Freq: Frequency

In this example it is also worth noting that the first decentile is very off for the Public Count, creating a large error which affected the final result.

5. Discussion

To start out, the dataset provides a realistic environment in terms of listen counts. In most cases there will be information on songs but users will only have listened to the song once or some will also have listened to a few songs many times. This creates a difficult environment for especially the collaborative model where the fundamental idea is to make recommendations based on user information only. Using baselines (3.1) to account for user specific scenarios can partly help the model but not enough.This was displayed in the predictions as the model did perform well on songs with a lot of ratings from other users where the user rating matched the popular opinion. Also, the true conversion from listen counts to ratings might not be completely accurate, a user can still enjoy a song he has listened to only once. But the assumption that a users tend to listen more to song they like was made to enable further analysis given the data.

This asymmetry of low ratings also creates difficulties to correctly assert high ratings, as displayed in 4.3. The example showcased in 4.1 for a specific user provides a good intuition for this. When the relative frequency of top ratings are higher than usual the model determines this to be a good song, despite the actual users opinion. This implies that there could be a bias towards more popular songs and thus a need for a more objective recommendation in some cases. Due to the nature of the model relying heavily on user interaction, in an environment with a majority of the users who have listened to songs only once and therefore given 'low ratings', the model could in this case be accurate but still have a hard time making any substantial recommendations.

After examining the content based results it is clear why the model performs poorly on users with a few songs since the definition of the evaluation method does not take this into account.

When calculating MSE_i^{Model} in the test set for users with less than 50 total songs, hence $\mathcal{S}_i \leq 50 \implies Test_i \leq 10$ since the test set only will hold 20% of the data. Since the user has not listened to enough songs there will not, per definition, be enough to understand the user distribution and thus one can see that if, for user i in decentile m:

$$\mathcal{L}_{i,m} = 0 \implies SE_{i,m}^{Model} = (0.1 - 0)^2 = 0.01.$$

Meaning that the model will be punished for not having enough information on each user. This can be clearly observed in both table 4.4 but more clearly in figure 4.3 where the almost linear trend between error and amount of songs is displayed. This is also confirmed in 4.4 where one can see that the set of unique songs for each user has a stronger impact on the error. Listen count and unique songs are per definition correlated but referring to the above implication, if Unique Songs$_{i,m} = 0 \implies \mathcal{L}_{i,m} = 0$ per definition of the evaluation method. This is not necessarily discouraging for the purpose of this paper, as the main focus was on dealing with new items rather than new users. Furthermore, to comment on the trade off between choosing the ratio between training and test set: a smaller training set allows for better estimation of the true label[1] but will worsen the approximation of the user distribution. Focusing optimization on this matter is on the verge of overfitting and therefore standard practice of 80 : 20 split was chosen.

The cross-validated result for the users with the most songs (4.5) is approximating the true label well enough to give the model a chance to occasionally outperform the baseline popularity-model. The purpose of the baseline model was not only to get comparable results, but in some sense to tell how much a users listening preferences, in terms of features, matches the mainstream opinion. If the true label and the public frequency are close it means that they come from similar distributions.

Figure 4.2 illustrates the basic idea of this approach, that the recommendations will lie close to the center of the user distribution. Thus representing features that the user on average prefers. It is also intuitive to understand why this approach requires a lot of information

[1] $\mathcal{L}_{i,m}$, the listen count for user i in decentile$_m$, for each $m = \{1, ..., 9\}$ (3.9) will be refereed to as the true label since this is the actual user distribution.

about each user but does not, in comparison to the collaborative model, require any information about other users. Since popularity is not a variable in the feature space, songs will be assessed only based on 'closeness' to the estimated distribution. The collaborative struggles to recommend songs that a few users have interacted with but can spot underlying correlations between users and thus make good recommendations based on user-to-user profiling. One core assumption of this paper was the normality of each user distributions. This assumption led to the removal of categorical and binary values that possibly could have provided useful information if extracted properly. One could argue that both acousticeness and instrumentalness neither are close to normal as well (B) and should thus be removed. But examining the clustered recommendations in figure 4.2 and cross-validating the results with and without the mentioned variables, they were kept as they actually provided some additional information and created better results. Nevertheless, since the difference between the model and the baseline average was not significantly different from zero one can argue one of the following:

1. A majority of the user's distributions are not significantly unique from that off the general public.

2. Given that users do have a different taste in music, the feature approximation is not fully captured by the methods used in this paper.

First, the reasoning that all users would have very similar taste and therefore existing a strong bias towards more popular music cannot be disregarded. However, it is more likely, given the results in this paper and previous research on collaborative models, that users would rather have grouped preferences than all go with the popular opinion. Second, both the method and evaluation has a lot of room for improvement. The Yeo Johnson transformation was arguably not the optimal method to use for this data. Both Box Cox and Yeo Johnsson are primarily to deal with outliers and tails in the distributions [7]. In terms of normality, the problem with the feature data was not necessarily the tails but rather that some were not particularly close to normal to begin with. There are several other methods one could use but this is currently outside the scope of this paper and left for further research.

Elaborating on further possibilities for improvement, as briefly mentioned in the introductory section, there are a lot of other factors to consider rather than just the raw listen counts and

features (1.1) outside the scope of dataset. Different consumption behaviour of users would could have big impact on the current method, as a song listened to passively still would result in false assumptions both by the likelihood and collaborative model. Moreover, the contextual aspect of different playlists could imply a lot of inner distributions for each user, making the assumption of a single distribution more difficult. This is followed by the aspect of time since music taste can change a lot over short periods. One could included a more details time series forecasting over listening patterns to further detect changing preference and thus also a shifting distribution. Both timing and playlist contexts could potentially enable stronger approximations and other, more substantial clustering-methods could be relevant to better address the feature data [16].

6. Conclusion and final words

This paper has been elaborating on two aspect of the recommendation system. First, the user to user dimension, where clusters of users tell a lot of information and recommendations are based on that basis. This approach requires a lot of user interaction to accurately build a proper profile and cluster it with like-minded individuals. The collaborative model struggles to recommend items that are not rated by many, but also in the environment of highly skewed data. However, given its scalability and capability to map a large amount of users to each other, one can understand how platforms such as Spotify can capitalize on detailed user data and thus use these methods to their full potential.

Given the extensive research on collaborative models, this paper aimed to address some of its issues by exploring the second dimension of user to item based recommendations. The strength and weakness of the likelihood model lies in its ability to disregard other user's preferences to make more independent selection on the actual context of the songs. These feature recommendations might not be songs one normally would have discovered, but they still matches the characteristics of songs the user has interacted with, allowing for more exploration in terms of both new 'under-the-radar' songs and artists. In future work, combining both models to a hybrid would present the opportunity to deal with the cold start problem through feature data but at the same time exploit the hidden correlations collaborative model identifies between users. Despite the non significant difference between the popularity and likelihood model, the small difference in performance after cross-validation suggests that with some improvements, one could still utilize the feature data to make more personalized recommendations.

A. Appendix - Optimization Algorithms

The point of many machine learning algorithms is to optimize a function $f(x)$ of some sort. This implies either minimizing or maximizing the function with respect to x in this example. The optimization function is called the *objective function*. Usually referred to in terms of minimization as the *cost function* [9]. This section aims to explain the concept of stochastic gradient decent which is used for optimizing the Probabilistic Matrix Factorization model. A brief explanation of gradient decent is given first to then 'explain' how it is performed with a stochastic approach.

A.1 Cost Function

A cost function as explain above is the function we seek to minimize, commonly written as $J(x, w)$ with the goal to minimize $\mathbf{w}$. In words, the cost function tells us how far away the prediction is from the actual value, and intuitively one want this function to be as small as possible since this implies that our model predictions corresponds well to the actual reality. A common cost function taught in early statistic courses is the mean squared error:

$$J(x, w) = \frac{1}{N} \sum_{i=1}^{N} (y_i - g_w(x_i))^2$$

N is the number of training examples, y_i is the real value for training example i, x_i is the input parameters for the training example i and $g_w(x_i)$ is the predicted value for training example i using the parameter(s) w. The objective is then to solve: $\text{argmin}_x J(x, w)$ which is done using gradient decent.

A.2 Gradient Decent

Let $f(x)$ be a function defined on all $\mathbb{R}$ with derivative $f'(x)$. The derivative tells us how we can scale the input to obtain corresponding change in output: $f(x+\gamma) \approx f(x) + \gamma f'(x)$ for a small enough γ [9]. With this in mind the following also holds:

$$\lim_{\gamma \to 0} f(x - \gamma \operatorname{sign}(f'(x))) < f(x).$$

We can thus reduce $f(x)$ by subtracting the derivative, this is called **gradient descent** [13]. Gamma is used as the notation for the learning rate or the step-size, hence how much we will 'move' the function in one direction. Even though practically impossible our goal is to find the global minimum, where no change would result in a lower value of $f(x)$.

In a multidimensional setting with several inputs one must take the the partial derivative of each of the inputs at the current location. So $\frac{\delta}{\delta x_i} f(x)$ yields the change in f as only the variable x_i changes at point x. The vector consisting of all partial derivatives of all variables evaluated at x is called the gradient and is written as $\nabla_x f(x)$.

$$\nabla_x f(x) = \begin{pmatrix} \frac{\delta f}{\delta x_1}(x) \\ \vdots \\ \vdots \\ \vdots \\ \frac{\delta f}{\delta x_n}(x) \end{pmatrix} \tag{A.1}$$

The direction to move $f(x)$ in order to descent the fastest is the gradient itself. This can be showed if we let $\overrightarrow{v}$ be a unit vector (implying that $|\overrightarrow{v}|_2 = 1$) and project it on $\nabla_x f(x)$.

$$\overrightarrow{v} \cdot \nabla_x f(x) = |\nabla_x f(x)|_2 |\overrightarrow{v}|_2 \cos\theta = |\nabla_x f(x)|_2 \cos\theta \implies \min(\overrightarrow{v} \cdot \nabla_x f(x)) = -\nabla_x f(x).$$

Thus, to decent from point x to x* using a gradient based approach one would compute:

$$x^* = x - \gamma \nabla_x f(x)$$

To describe how x is updated to x^* we will use the notation $:=$ which implies that x will be replaced by the computed value to the right.

$$x := x - \gamma \nabla_x f(x)$$

Gamma, as previously mentioned, is the learning rate. This is a parameter usually set by the researcher as a small constant in a trail- and error setting. If the learning rate is too big the updated values would cover too large intervals, most likely skipping the minimum. With a too big small learning rate the computations would take too much time. Figure 3.1 displays a the gradient decent process with an appropriate learning rate.

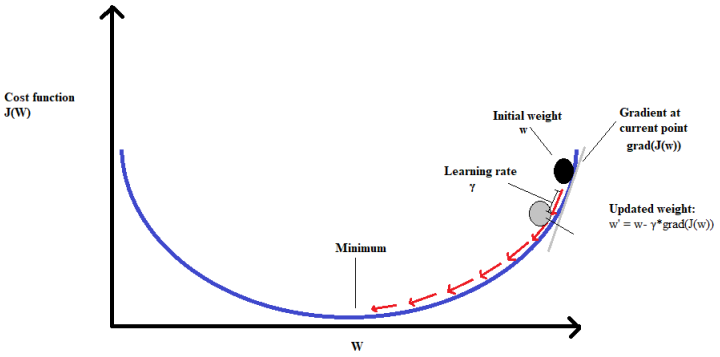

Figure A.1: Gradient Decent Visualization

Figure 3.2 shows the trade-off and difficulties of selecting a proper learning rate.

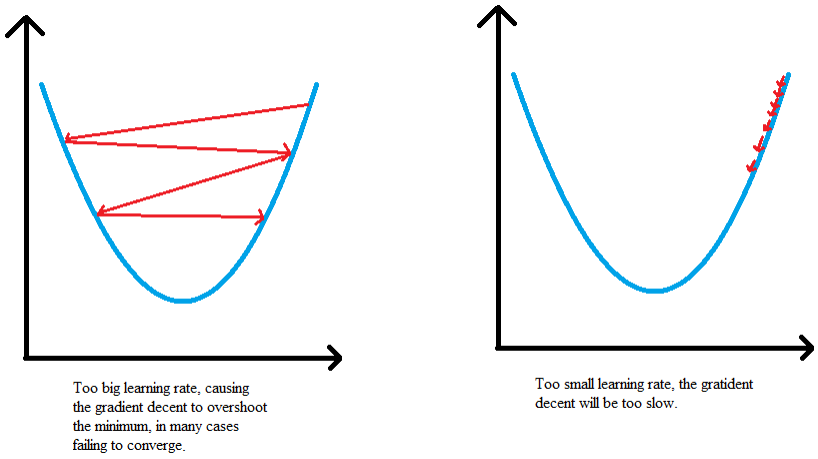

Figure A.2: The dilemma of choosing learning rate

A.2.1 Stochastic Gradient Decent

One problem with the general gradient decent approach is how it scales poorly on large problems as the computational cost becomes too high. The idea of Stochastic Gradient Decent (SGD) is to approximate the gradient using a uniformly drawn sample from the training set, also referred to as a minibatch [9]. Instead of computing the gradient for millions of examples we estimate the gradient with our minibatch of size n drawn from the population of size N:

$$\mathbf{g} = \frac{1}{n'} \nabla_{\mathbf{w}} \sum_{i=1}^{n} J(x^{(i)}, y^{(i)}, \mathbf{w})$$

Where n is the sample from the whole population of examples, $J(x, y, \mathbf{w})$ is the cost function which we want to minimize for w. As Figure 3.1 the gradient descent technique is applied with the estimated gradient $\mathbf{g}$:

$$w := w - \gamma \mathbf{g}$$

A.3 Regularization

Overfitting is a very common problem in all machine learning areas; creating a model that is too adapted to the training data, consequently providing a model unable to generalize for

any out of the sample observations. One of the means to reduce overfitting is regularization. By introducing a bias in the cost function with the aim to stabilize the model by penalize big values of the coefficients in our matrices U and V, which will be covered in detail later. By adding the below regularization term to our cost function which we wish to optimize later:

$$\lambda(||U||^2 + ||V||^2), \lambda > 0. \tag{A.2}$$

λ is the regularization parameter, a hyper parameter we tune ourselves as we try to find the best model [4]. $||U||^2$ is the squared Frobenius norm, or euclidean distance which is defined as the square root of the sum of the squares of all the matrix entries. Let A be a matrix of size $n \times m$, then the norm is defined as:

$$||A|| = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} \tag{A.3}$$

B. Appendix - Yeo-Johnson transformations

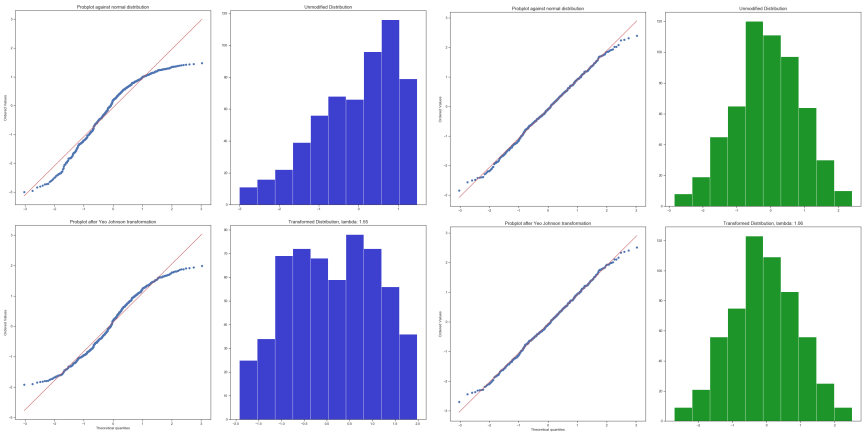

Figure B.1: Transformation results: Energy (left) and Danceability (right)

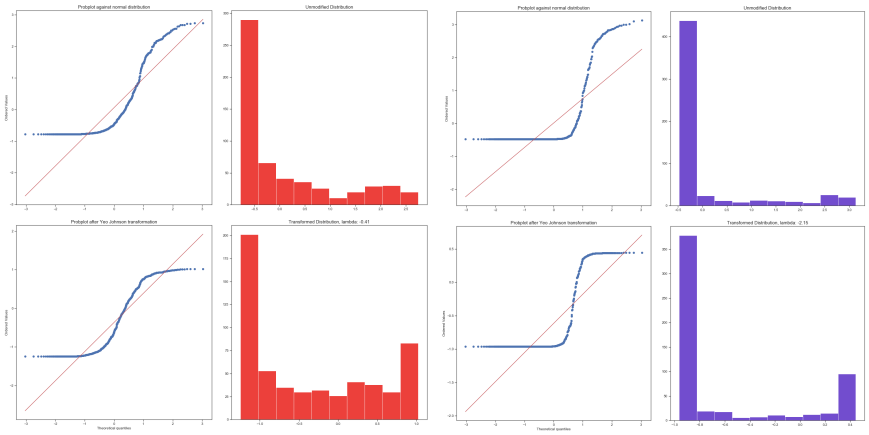

Figure B.2: Transformation results: Acousticness (left) and Instrumentalness (right)

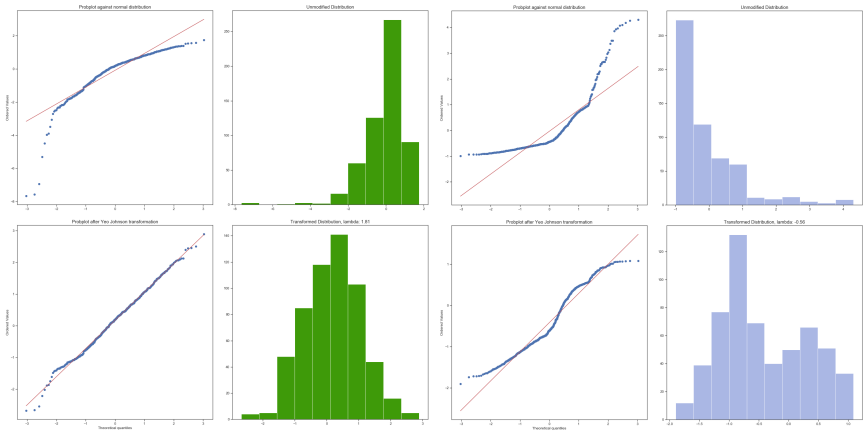

Figure B.3: Transformation results: Loudness (left) and Liveness (right)

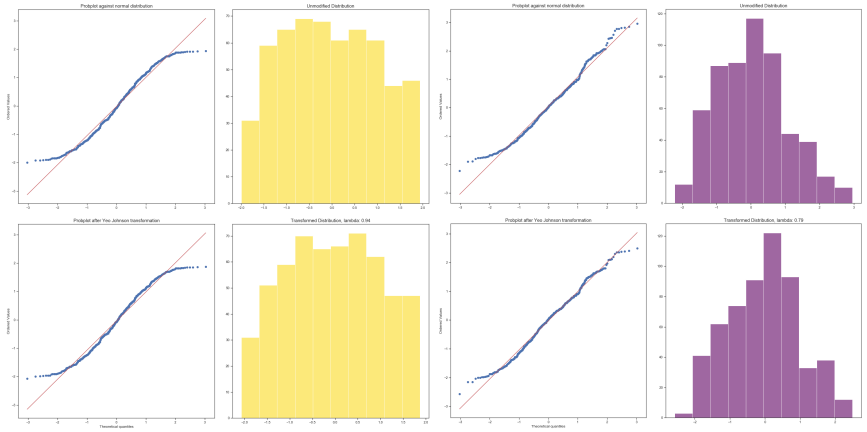

Figure B.4: Transformation results: Tempo (left) and Valence (right)

Bibliography

[1] Spotify. `https://newsroom.spotify.com/company-info/`. Accessed: 2020-02-10.

[2] Scipy. `https://github.com/scipy/scipy/blob/v1.4.1/scipy/stats/morestats.py#L1273-L1363`, 2020.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[4] C. C. Aggarwal et al. *Recommender systems*. Springer, 2016.

[5] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[6] J. Bobadilla, F. Ortega, A. Hernando, and J. Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-based systems*, 26:225–238, 2012.

[7] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[8] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

[10] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[11] F. O. Isinkaye, Y. Folajimi, and B. A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[12] M. Kaminskas and F. Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, 6(2-3):89–119, 2012.

[13] C. Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251:254, 2012.

[14] G. J. McLachlan. Mahalanobis distance. *Resonance*, 4(6):20–26, 1999.

[15] F. Ricci, L. Rokach, and B. Shapira. *Recommender systems handbook*. Springer, 2011.

[16] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116, 2018.

[17] Simon Funk. Netflix update: Try this at home. https://sifter.org/ simon/journal/20061211.htmll, 2006.

[18] Spotify. Spotify: Spotify Web API Surface library. `https://developer.spotify.com/documentation/web-api/`, 2020.

[19] B. W. P. L. Thierry Bertin-Mahieux, Daniel P.W. Ellis. The Million Song Dataset,in proceedings of the 12th international society for music information retrieval conference, 2011.

[20] M. Thill. The relationship between the mahalanobis distance and the chi-squared distribution. `https://markusthill.github.io/mahalanbis-chi-squared/`, 2017.

[21] S. Weisberg. Yeo-Johnson power transformations. *Department of Applied Statistics, University of Minnesota. Retrieved June*, 1:2003, 2001.

[22] I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.