IT UNIVERSITY OF COPENHAGEN

Master Thesis

Interactive Data Visualization for a better understanding of Recommender Systems

Author: Dennis Leszkowicz
Supervisor: Martin Aumüller

IT University of Copenhagen
Copenhagen, Denmark

A thesis submitted in fulfillment of the requirements for the degree of Master of Science.

January 2019

Abstract

Recommender systems provide a useful mechanism to overcome information overload by reducing a dataset to items that a user may like. Due to the high complexity recommender systems are often perceived as "black boxes" because they don't explain or show how recommendations are computed, leading to a lack of transparency.

This project conducts an experimental study that compares an interactive- and a non-interactive (baseline) recommender system. Using state-of-the-art algorithms and the Last.fm 360K Users Dataset a web application and surrounding experiment is created. The results show no remarkable insights, mainly due to a small sample size. The main takeaway is coming to realize how difficult and interdisciplinary user-centric recommender system studies are. The findings constitute a collection of weaknesses and future suggestions proposed to alleviate the problems encountered throughout this study.

Keywords: Machine Learning, Recommender System, Implicit Feedback, Matrix Factorization, Spotify

Acknowledgments

I would like to thank my supervisor Martin Aumüller, for guidance and support. Additionally, I would like to thank Hugh McGrade for advice within the realm of web applications and the Flask microframework. Thanks to Ben Fredrickson for the Implicit library. Without this fast implementation of Implicit Matrix Factorization, this experimental study would not have been possible. Thanks to Òscar Celma for the data collection that constitutes the Last.fm 360K Users Dataset used in this study. Finally I would like to thank all the people who participated in the experiment providing valuable feedback.

Contents

List of Figures

List of Tables

1

Introduction

Artificial intelligence and machine learning technologies continue to broaden and influence the way we interact with computers. Moreover it influences our access to information, entertainment, products, services and each other through data-driven recommendations. As increased access to AI has undoubtedly improved many aspects of social welfare, the ability of Recommender Systems (RSs) to provide genuinely personalized recommendations and well explained recommendations remains limited [6]. The main motivation for using RSs is to overcome the information overload problem by providing people with only information and items they are likely to have interest in. This task has come to be known as the recommendation problem [1].

This thesis covers aspects of information filtering, more specifically within the field of RSs, focusing on the widely used Collaborative Filtering approach which assumes that people tend to agree with people they agreed with in the past [15]. *The Netflix Prize*, a machine learning and data mining competition for movie rating prediction (announced in 2006) was an important event for Recommender System (RS) research. The competition which span between 2006-2009 highlighted the importance of the recommendations and accelerated the development of many new recommendation- and data mining techniques. The main outcome of the prize was a simplification of the recom-

mender problem, achieved by predicting user's ratings while optimizing the Root Mean Square Error (RMSE) between the predicted and actual ratings [41]. Another event worth acknowledging is the the Million Song Dataset Challenge [31], both events spun of algorithms and datasets of importance to the research community.

In recent years researchers have realized that RS evaluation extends well beyond the measure of accuracy prediction. Luckily for the end-user, for which the primary purpose of the system is to provide personalized help in discovering relevant content or items [26] and thereby reduce the choice overload. This acknowledgement has resulted in important changes in the field. Those will be presented in chapter 2.

The problem domain and the goals of this thesis are outlined in section 1.2. Overview of current work and state-of-the-art approaches in the domain are presented in chapter 2. This project implements two different RSs in one web application and conducts an experimental study of those. The implementation and design choices are described in chapter 3. Chapter 4 presents the results of the study followed by a discussion and suggestions for future work in chapter 5. Finally the project is concluded in in chapter 6.

The recommender algorithm used in this thesis is implicit matrix factorization as described in the paper Collaborative Filtering for Implicit Feedback Datasets [19] by Hu et al. The library used is Ben Frederickson's `Implicit` Fast Python Library [14] and finally the dataset used is the Last.fm 360K Users Dataset containing data collected in 2008 [8, 9].

1.1 Motivation

The motivation for this study arose from the ACM FAT* (Association for Computing Machinery - fairness, accountability, and transparency of algorithmic systems) conference call for papers on algorithmic transparency. This topic covers machine learning, statistics and data mining: "How to design strategies for communicating the logic behind algorithmic systems [35]. Due to the high complexity of RSs they are often perceived as "black boxes" to the end user because they don't explain how recommendations are computed [42], leading to a lack of transparency. Alleviating this problem will help catering explainable and transparent recommendations to users. Moreover, in the light of the latest ACM RecSys (Recomender System) challenge - the information overload problem is a problem that Spotify and other streaming services have to solve all the time [51].

1.2 Research question

The overall goal of the project is to improve transparency in RSs by utilizing visualization techniques that aim to show how recommendations are computed. A subsequent goal is to allow users to adjust their recommendations by interaction with a RS. This is explored through a scientific study that compares a baseline RS with a RS that integrates visual interaction.

The project outcome is a web application prototype, that implements two different recommender approaches, an experimental study of the application (RS Experiment), evaluation, presentation of results, discussion and suggestions for future work. The hypothesis to be tested in the experiment is: recommending for a subset of user interactions results in a higher satisfaction and algorithmic transparency, in contrast to recommending for a users

full interaction history.

Throughout this paper *italized* terms and concepts are words that the reader is not expected to have prior knowledge of, thus a description or definition can be expected to follow. `teletypefont` is used to emphasize a name, typically indicating program names, classes or libraries. These conventions are used pragmatically.

2

State Of The Art

This chapter covers previous work in RSs and concepts of importance for this study. The intention is to establish a common understanding to serve as a basis for the chapters to follow.

2.1 Recommender systems

This section serves as a brief introduction to RSs where its terms and concepts will be explained alongside a state of the art review. The scope will increasingly go from a general view to a more specific music RS setting. A RS is an example of applied machine learning and data mining algorithms at scale in commercial practice [2]. A RS is a program that suggests items that are most likely of interest to the users [41]. As described in the introduction, this problem is known as the choice overload [1]. Speaking of terminology, *item* represents the object that the system recommends or takes as part of its input. This could be movies, songs, products or books. A *user* is the person who gets the recommendation. The purpose of recommendations is to simplify the decision making process for the users of a given service [41]. RSs comes in many forms and shapes. Widely speaking these can divided into four categories [5] that differ substantially in the way they solve the

recommendation task. The categories are:

Collaborative Filtering (CF)
Demographic Filtering
Content-based Filtering
Hybrid Filtering

Within those four categories there are many subcategories. CF can be seen as word-of-mouth of the internet. Before the Internet word-of-mouth would be limited to tens or hundred individuals. With help from the internet this can now exceed beyond thousands [46]. In practice CF makes recommendations based on relations between one users likes/dislikes (opinion) of a subset of the whole and another users opinion. In Demographic Filtering user-characteristics in form of sociodemographic attributes such as age, gender, profession, and education are used [41]. Content-based Filtering is recommendations based on the items that a user has liked in the past. The similarity of the items are calculated based on item features (e.i. if a user has positively rated an item that belongs to comedies, then the system learns to recommend other items from this movie-category) [30]. Hybrid Filtering combine above mentioned methods for the same purpose; making recommendations [5]. If the data at hand does not provide rich meta features of the items, Content-based Filtering will perform poorly [38]. In the scope of this study features in form of characteristic of music artists are not part of the dataset, thus CF is the intuitive way to go. Pazzani concludes that both Content-based- and Demographic Filtering improve recommendations over CF, however they suffer from the problem of information availability [37], which is not a problem for CF, as the approach is domain independent [19]. However, CF suffers from what is known as the cold-start problem due to its lack of addressing new items and new users to the system, which Content-based Filtering is capable of [19].

2.2 Implicit and explicit feedback data

This section explains the difference between implicit- and explicit feedback data. Explicit feedback is a measure provided by the user, where implicit feedback is derived from the users behaviour. Explicit feedback data express a given users opinion directly, whereas the implicit feedback shows the users opinion indirectly [19]. Explicit feedback is generally more accurate than implicit feedback which exists in abundance [21] and thus is easier get hands on. Moreover, explicit feedback can be both positive and negative, whereas implicit feedback is only positive [21]. In terms of availability implicit feedback is always available if the user has consent to it, whereas explicit feedback takes an active effort from the user [19, 23]. The different feedback measures for the scope of the thesis are described in section 3.1. Kelly and Teevan [23] noted that implicit measures are available in abundance, which is also the case for implicit music consumption data, given the rise of streaming services.

2.3 Collaborative filtering

CF has been improved a great deal, since its original form. The most popular CF approach shared by all earlier CF systems is user-oriented [19]. This method estimate ratings based on recorded ratings of like minded users. Since then an analogous item-oriented has become popular, where recommendations are calculated by estimating the unknown based on the known ratings from the same user [29]. Item-oriented approaches are favorable in many cases due to better scalability and improved accuracy [3, 45, 52]. Item-oriented methods are also more amenable to explaining the underlying reason behind the prediction, because users obviously are familiar with items they have previously preferred, but they do usually not know allegedly like

minded users [19]. To make recommendations a CF relates two different entities, namely items and users. For this comparison there exists two primary approaches, which represent the two main techniques in CF, namely neighborhood- and latent factor models [41]. One such model is known as matrix factorization. The model combines implementation convenience with a relatively high accuracy, making it the preferred technique for addressing large datasets, such as the publicly available Netflix Prize dataset [41].

2.4 Matrix factorization

The intention is not a thorough walk-through of matrix factorization (MF) and underlying theory, rather a common understanding of the concepts. How the concepts have been applied in this specific setting, will be covered in section 3.4. The theory is partly based on the authors prior work from the thesis preparation course [28].

Research has shown that MF can uncover latent features that explain ratings (explicit) / observations (implicit) r_{ui} [19]. The overall task is to estimate the missing values r_{ui}, given a sparse *user item* matrix and thereby complete the matrix. This can be achieved by matrix factorization. There exists different methods to uncover latent factors, here among Probabilistic Latent Semantic Analysis (pLSA) [18], neural networks [44] and Latent Dirichlet Allocation [4]. The task of MF is to decompose the original user item matrix R into two sub matrices X and Y as shown in fig. 2.1. To predict the values for Y and X the inner product is calculated: $\hat{r}_{ui} = x_u^T y_i$ [19]. x_u consists of user factors and y_i items factors. x_u and y_u values are obtained by minimizing the following cost function: [27]:

$$\min_{x\star,y\star} = \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \tag{2.1}$$

x_u and y_i are estimated by minimizing the above cost function by *stochastic gradient descent* (SGD). The second term λ is the regularization, where parameters are learned - discussed later in this section. Comparing this approach with neighborhood models tend to create consistently superior results [19].

As shown in fig. 2.1 a MF model associates each user u with an user-factors vector $x_u \in \mathbb{R}^f$ and each item i with an item-factors vector $y_i \in \mathbb{R}^f$.

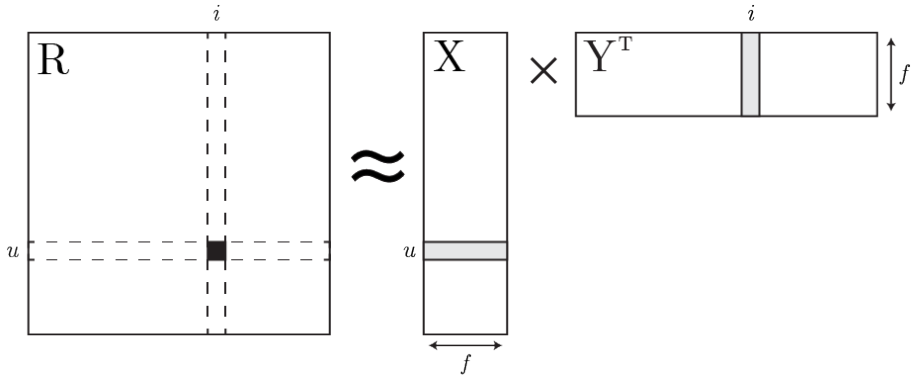


Figure 2.1: Matrix factorization

Since a user is only capable of interacting with a modest subset of the whole item collection, the *user/item* interaction matrix R is by nature sparse. Moreover, user interactions can easily be in the magnitude of billions. To overcome this the latent factors can be treated as parameters to be learned and the factorization as an optimization problem. The primary difference in approaching explicit vs. implicit feedback data is how missing values are treated.

Hu et al. suggests a preference-confidence partition which indicates the pref-

erence of user to item derived from the r_{ui} variables arguing that it is better to predict preference p_{ui} for an item rather than score r_{ui} [19]. This convention is also closer aligned to observational data, as it makes little sense to talk about ratings and scores when dealing with implicit feedback. Most MF models treat missing values as unknown, however this won't work for implicit feedback. The fact that a user has not interacted with an item implies more than just unknown. By missing values one can simply not tell whether the user likes or dislikes an item, considering the fact that the user is not yet familiar with the item.

A set of binary variables p_{ui} are defined to account for low confidence:

$$p_{ui} = \left\{ \begin{array}{l} 1 \text{ if } r_{ui} > 0 \\ 0 \text{ if } r_{ui} = 0 \end{array} \right\} \tag{2.2}$$

$r_{ui} = 0$ represents low confidence and $r_{ui} > 0$ represents high confidence, the larger the r_{ui} value the stronger the preference, analogous to the more times one have interacted or clicked an item, the more one presumably likes it. To account for uncertainties the following formula is proposed by Hu et al. introducing a new matrix C where c_{ui} measures the confidence of the observation p_{ui}: $c_{ui} = 1 + \alpha r_{ui}$

The constant α is a hyperparameter tunable to the specific dataset. $\alpha = 40$ has proven to deliver good results [19]. The preference is assumed to be the inner product of x_u and y_i: $\hat{p}_{ui} = x_u^T y_i$. What distinguishes this method from Explicit MF is the fact that varying confidence is accounted for and the optimization accounts for not only non-zero observations but all *user/item* pairs in the matrix R.

A clear advantage over neighborhood CF models is the decomposition into latent factors, which makes it possible to predict rating or preferences, by

inference based on the non-zero ratings/observations [27]. Comparing to the cost function for explicit feedback in section 2.1 there are two distinctions to take note of: (1) Accounting for varying confidence levels. (2) The optimization should account for all u, i pairs, in contrast to only considering known values [19]. The factors are computed by the cost function:

$$\min_{x\star, y\star} = \sum_{r_{u,i}} C_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda\Big(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\Big) \tag{2.3}$$

Similar to the cost function in section 2.1 the second term regularizes the model ensuring that the model does not overfit the training data. The cost function contains $m \times n$ terms, where m is users and n is items. The size of $m \times n$ can easily reach the magnitude of billions, which prevents the use of the explicit approach (fig. 2.1) for optimization [19].

Hu et al. propose an efficient Alternating Least Squares (ALS) optimization that integrates the confidence levels c_{ui}. The model described in [19] is hereinafter referred to as Implicit ALS (iALS). The optimization works as follows: A mean squared error (MSE) with regularization is used to measure the average of the squares of errors. The error is considered the difference deviation between estimator and what is estimated. The trick for ALS is when either the user-factors or item-factors are fixed, then the problem can be transformed to a regularized *least squares problem*, where the cost function becomes quadratic and its global minimum can then be readily computed [19]. Assuming that all items are gathered within an $n \times f$ matrix Y (similar to Y^T in fig. 2.1) and all users in similar fashion within an $m \times f$ matrix X (similar to X in fig. 2.1).

The objective is to compute X and Y so user $u's$ preference towards an item i can be estimated by the inner product: $\hat{r}_{ui} = x_u^T y_i$. The preferences is computed by alternating between re-computing user-factors and item-factors,

thus each step is guaranteed to lower the value of the cost function. In first step Y is fixed by filling the matrix with random values and then updating X by differentiating the cost function 2.3.

This step is repeated for all m users and results are stored in a $f \times m$ matrix X and each x_u as a column. In the following step the same is done for the item-factors Y_i. Item-factors are updated and stored in $f \times n$ matrix Y and the user-factors can be updated again. Alternating back and forth minimizes the cost function until the factors *converge*, each iteration guarantees to minimize the error rate, given that the other side is fixed. To converge in this context means that the cost function keeps iterating until the error rate is not minimized further, thus the function is said to has converged - 10 iterations are suggested by the authors for good results [19]. This kind of optimization problem is considered non-convex as both Y's and X's are unknown, which can be transformed to a simpler quadratic problem as previously shown.

Finally, it is well accepted that a good recommendations should be accompanied by meaningful explanations [17], thus improving the users' trust in the system and perspective. Hu et al. demonstrate that the latent factors can be used to explain the recommendation providing qualitative insight for why recommendations are made [19]. As shown in `lastfm_explain_function.ipynb`[1] a person was recommended the artist Bon Iver. This is explained by providing the top n contributors from the user items, in this case $n = 5$ and the top contributors are (Fleet Foxes, Bonnie 'Prince' Billy, Department of Eagles, Band of horses, MGMT). Which makes good sense, if you are familiar with the artists, showing that the basis for recommending Bon Iver (indie folk, singer-songwriter, electro) are rooted in similar artists from the user items.

Additionally, the latent factors can be used for similarity search between items, as demonstrated by Johnson [22]. In this study the model is used

[1] See appendix G for code (notebook reference)

out-of-box made possible by Ben Frederickson with his fast Python library `Implicit` [14]. The interactive part is possible thanks to the `recalculate()` function in the library. More details on this will follow in section 3.4.

2.5 Visual recommender systems

This section highlights state of the art within Visual RSs of relevance for this study and with the purpose of providing users with more transparency and thus a more useful experience.

Richthammer et al. argue that the quality of recommendations has considerably improved in recent years. On the contrary the complexity of the algorithms used in RSs has notably increased [42]. This leads towards high non-transparency [11], whereas understandability and high transparency may result in higher trust towards recommendations [50]. Richthammer et al. propose usage of interactive visualizations for presenting recommendations. Their findings illustrated by means of an example scenario that shows how interactive visualizations can support the user's entire decision process, spanning from the information reduction phase to presentation of the recommended items [42]. The authors stress a current research gap within RSs, arguing that visualizations help us leverage the human's ability to see patterns and spot trends or outliers [16]. Thereby notably supporting the understanding of the data. Moreover they conclude that a comprehensive and general discussion of Interactive Visualizations for RSs data is missing [42]. Richthammer et al. argue that *"the pool of suitable visualization techniques for each information block is limited to the those fitting the identified data type, the selection of the most useful and usable technique is hard to measure and a quite subjective estimate"* [42]. The authors suggest visualization techniques for among other item features, feedback data and context (conditions

and constraints). A technique that has become popular in recent years is the *tag cloud*, also known as *word cloud* [47]. As seen in fig. 2.2 the words or tags from a given text, varies in size. The size represents importance, the bigger the word the more frequent it usually occurs in the text [42].

Figure 2.2: Tag cloud visualization of the textual reviews of an example horror movie. The color of the tags refers to the related rating value [42].

Moreover, the authors argue that the decisions of traditional black box RSs are difficult to understand, as they do not provide any information on the nature of the input [42]. This being especially relevant when a user is not satisfied with the recommendations he or she is given. To solve this problem Richthammer et al. suggest a concept of interactive visualizations for RSs data that reduces the set of all alternatives to manageable size, instead of directly reducing input data to a final set of recommendations [42]. This is visualized by the use of parallel coordinates as seen in fig. 2.3 and 2.4.

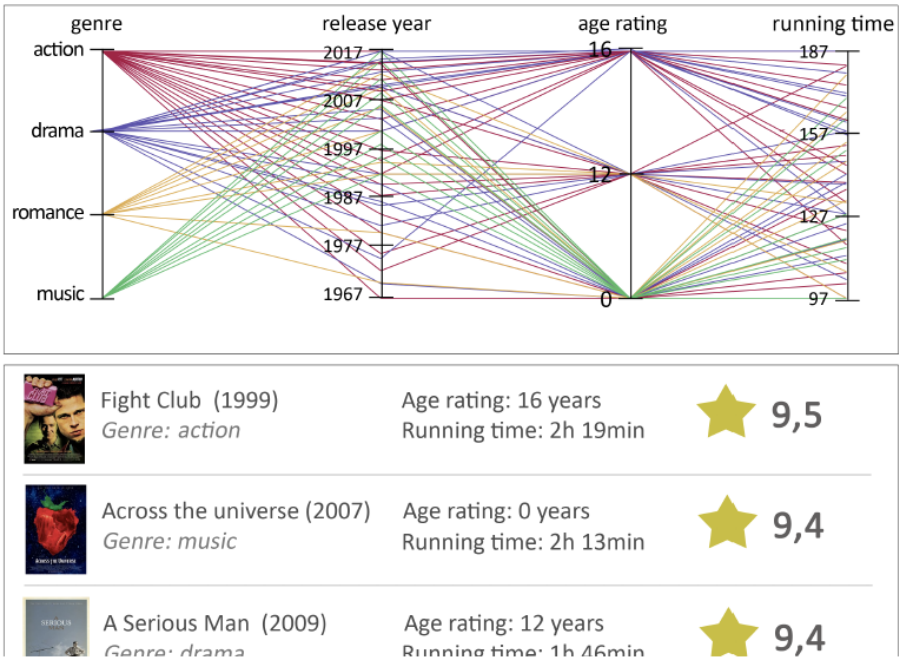


Figure 2.3: Parallel coordinates showing the movies with highest predicted ratings in terms of their item features [42].

In fig. 2.3 the user Bob obtains a quick overview of the items with the highest predicted ratings and their features. As the visualization shows all items belong to one of four genres, in this example Bob decides that tonight is action and/or drama movie night. Additionally he decides that the movie he would like to watch should be from within the past decade from 2017, have a minimum age rating of 16 and a duration of at least 150 minutes. Therefore Bob interacts with the parallel coordinates visualization to reduce the set of polylines to match his criteria as seen in fig. 2.4. This defines a conditional and constraint-based approach which is valuable when integrated in item-related or opinion-related visualizations. Richthammer et al. find that the approach is more meaningful in the output stage of a RS, rather than as input criteria, since Bob has no idea about what items satisfy which constraint [42].

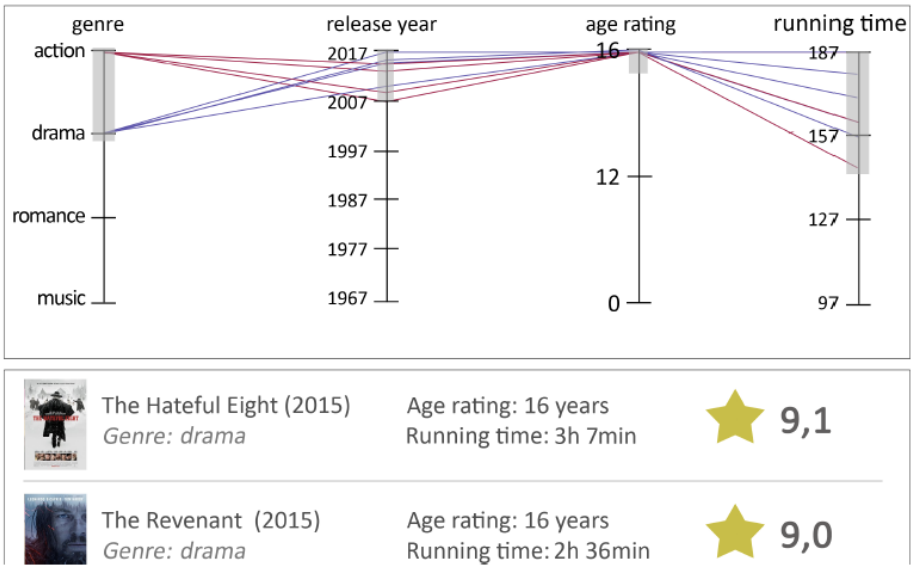


Figure 2.4: Interactive specification of constraints with parallel coordinates [42].

These findings support the more general findings from the prior section, calling for more user-centric focus within RSs research.

2.6 Evaluation of recommender systems

Conforming to the goal of this study, this section presents current work from a user-centric perspective, including metrics, approaches, frameworks and other relevant details.

The need for user-centric studies has resulted in two important changes in the field [26]: The first change was proposed by McNee et al. who argued that *"being accurate is not enough"*. One should instead study RSs from a user-centric perspective to make them not only accurate and helpful, but a

pleasure to use as well [32]. McNee et al. suggested broadening the scope of research and evaluation of RSs beyond accuracy measures focusing on online user experiments with an user-centric evaluation metric covering user retention, consumption as well as attitudes (e.g. usability, choice satisfaction, and perceived usefulness [25, 39]). The second change is about broadening the scope of research beyond the system and its algorithmic performance. To understand the distinction it is useful to emphasize the purpose of a RS:

"In essence, RSs apply algorithms on user input with the goal of providing some kind of personalized output. This means that aside from the algorithm, there are two important interactive components to any RS: the mechanism through which users provide their input, and the means by which they receive the system's output" [24].

Realizing the importance of interactive components in RSs McNee et al. [33] suggested more focus on the "Human-Recommender Interaction" and study of interaction in RSs. To support this, research has shown that preference elicitation mechanisms and the presentation of recommendations indeed have a substantial impact on users', counting acceptance and evaluation of RSs as well as their behaviour while using the system [10, 25, 40]. However, the majority of current research is still primarily focused on creating better algorithms, and conducts offline machine learning evaluations instead of "live" user experiments. The contribution of that research is thus limited to claims about algorithmic accuracy and precision. However, without performing any user-centric evaluations it is difficult to extend these claims to the more user-centric objective of RSs: giving users a pleasant and useful personalized experience [24].

2.6.1 Evaluation: offline vs. online

Many recommendation approaches consists of some interaction between user and system. Offline testing is difficult to conduct, due to the fact that it is hard to create reliable simulations of users interactions with a given system. Thus, in order to evaluate a system properly, real user interactions must be collected. Once an offline setting is possible, integrating additional interactions with real users will provide additional information about system performance and also facilitate user studies [41]. These can eventually before very insightful, however the main disadvantage of user studies are that they are expensive and tedious to conduct, which eliminates the option for this study.

The other side of the coin is online evaluations. Ricci et al. argue that the strongest evidence in terms of system value is given by online evaluation, where simulation is at its minimum. *"It is most trustworthy to compare a few systems online, obtaining a ranking of alternatives, rather than absolute numbers that are more difficult to interpret"* [41]. Therefore many real world RSs employ online testing systems where multiple systems and approaches can be tested. Pu et al. propose a user-centric framework for measuring the quality of recommendations and a system's usability and usefulness. The framework consists of thirty two questions and fifteen constructs, defining the essential qualities of an effective and satisfying recommender system [39]. How the RS Experiment and its test-system is built will be covered in section 3.5.

3

Methodology

This chapter accounts for the proposed solution, which includes the web application prototype and the general design of the RS Experiment. Initially a brief overview is presented of the nature of music consumption data and the data sources used in this study. Finally the system design and implementation approach is described on a high level with some low level insights.

3.1 Data source

This section briefly presents the key characteristics of music consumption, followed by a description of the dataset used in this study, how it has been cleaned and finally enriched.

3.1.1 Music consumption

Music consumption is indeed interesting because the consumption time of a song or album is usually much shorter than a movie or a book. Where the duration of a movie is typically 90 minutes or more, not to mention books. The duration of a song typically ranges between 3 and 5 minutes. Thus we can assume a naturally higher demand for music recommendations as the

item throughput here is substantially larger. In terms of volume, commercial music catalogs such as Spotify are in the range of tens of millions music pieces, while movie catalogs such as Netflix deals with significantly smaller catalogs, typically thousands to tens of thousands unique movie items[1] [28]. The bigger the item catalog and consumption throughput the more sparse the user interactions in the corresponding dataset.

3.1.2 Data types

The following list shows the different data types used and created throughout the experiment:

Implicit Feedback Data:

- **Last.fm 360k Users dataset:** one entry represents a user's history in terms of what artists the user has listened to how many times.
- **RS Experiment:** User clicks collected in the web application.

Explicit Feedback Data:

- **Last.fm tags:** explicit feedback, in terms of *tags* users have explicitly associated with artists.
- **RS Experiment:** *Artist* and *tags* preferences collected from participants.
- **RS Experiment:** Evaluation in terms of a survey that is filled in by the individuals who participate in the RS Experiment.

[1] Spotify reports 40+ million song catalog in 2018 (`https://newsroom.spotify.com/company-info/`)

3.1.3 Last.fm 360K Users Dataset

Last.fm is a music website dating back to 2002. The Last.fm dataset consists $\sim$ 360.000 unique user profiles from Last.fm collected during fall 2008 and released for the first time in 2009 [8]. The dataset contains $\langle$ user, artist, plays $\rangle$ tuples collected with the `Last.fm API`[2], using the `user.getTopArtists()` function. The work was carried out by Òscar Celma [9]. The dataset contains user demographics, but has missing information in form of 32.775 gender features and 74.900 age features, corresponding to roughly 10 and 20% missing values (see appendix A). Why these values are missing is unclear. Since those numbers are not catastrophic, the information could have been included in the model. However, the main dataset used in this study consists of implicit historical data in terms of how many times a user has been listening to an artist (*user plays*, used interchangeably with *user items* and *user interactions*), the more general term.

3.1.4 Data cleaning and enrichment

Initially the data was loaded and cleaned manually, where missing values were attempted to be replaced, otherwise removed. As this approach resulted in a poorly performing model, Ben Frederickson's cleaned dataset has been reused. Essentially the data source is reused from this script using the `get_lastfm()` function[3]. The dataset entry described in section 3.1.3 looks like this in the source file `<userId (string), artistId (string), artistname (string), plays (integer)>`. The cleaning approach[4] removes invalid entries if one of the following conditions are satisfied:

[2] `https://www.last.fm/api`

[3] `https://github.com/benfred/implicit/blob/master/implicit/datasets/lastfm.py`

[4] Source code for cleaning approach, `clean_dataset()`: `https://github.com/benfred/bens-blog-code/blob/master/distance-metrics/musicdata.py`

1. if the entry has not exactly 4 elements.
2. if play count (`plays`) is a non integer.
3. if the `artistsId` is invalid.

	Raw	**Cleaned**
Unique Users:	359,347	358,868
Unique Artists:	294,015	292,385
Artists with tags:		240,505
Unique tags:		149,201

Table 3.1: Last.fm 360K Users Dataset in numbers

By adding artist *tags* to the dataset, it becomes possible to identify artists of similar style without beforehand being familiar with the artist. In order to get hands on this data, a script has been written to retrieve top-10 tags for each unique artist from the dataset. For this purpose the python Last.fm interface `pylast`[5] is used, utilizing the `Artist.getTopTags()` function[6]. Obviously not all artists have been tagged by users and thus only artists with tags available are enriched, as summarized in table 3.1[7].

3.2 System description

This section describes the system (web application), what problems it solves and how. The goal of the system is to test the hypothesis formulated in section 1.2 *"recommending for a subset of user interactions results in a higher satisfaction and algorithmic transparency, in contrast to recommending for a users full interaction history"*. `Implicit` iALS recommends for all user items out of the box. By adjusting the user items with the `Implicit` library's

[5] `https://pypi.org/project/pylast/`
[6] `https://www.last.fm/api/show/album.getTopTags`
[7] For source code see `get_artist_tags_api.ipynb` (appendix G, table G.1)

`recalculate()` functionality, it is possible to recommend for a subset, thus reducing information and simultaneously catering to the users specific need in almost real-time. Implementation details follow in section 3.4. Recall Bob from section 2.5 who decides to reduce the recommendations to drama and action movies. In this case the filtering happens directly on the artist (see appendix D) where r_{ui} (user items) are adjusted by the user and recommendations instantly recalculated and updated in a visually interactive fashion. This interactive approach is compared against a baseline that recommends for all user items, corresponding to the complete artist history of a user. The A/B test is equally split between visitors, utilizing a threaded counter, ensuring as close to equal sample size for both approaches. Eventually users fill in a survey, that measures their satisfaction rate.

3.2.1 Additional considerations

One advantage of adjusting user items on the artist level is more control and freedom to adjust items, in contrast to adjusting prefixed groups of artists by tags. The clear disadvantage of adjusting for each artist is that it takes way more clicks and ultimately time. In order to carry out an experimental study it takes a system that can interact with new users. This is known as the cold start problem and is the first obstacle to overcome when designing the system for this study. To solve this problem a user provides explicit information that is representative for his or her music taste. The solution is inspired by Netflix's cold start solution[8], where a new user has to select at least three movies that he or she likes. Moreover, a tag cloud visualization (appendix C) is used to show the tags that constitutes the dataset. The tag cloud is computed based on a frequency mapping of top 10 artist tags for all the profiles in the dataset. This is done using Andreas Mueller's `word_cloud`

[8]See appendix B for a screenshot of the service.

library for python [34]. Moreover, the tags feature is consistently displayed in the proposed system alongside the artist, showing top-3 associated tags[9].

As seen in appendix D, the system comprises of an onboarding part that solves the cold start problem by matching a user with a music profile in the dataset. This is achieved through a filtering process where a user either provides at least two tags or two favorite artists. Once the user is somewhat satisfied with the suggested profile, ten recommendations are provided, along side their associated top-3 tags.

3.3 Interactive data visualization

The interactive data visualization is integrated as two adjust buttons that allow the user to either increase or decrease the user items for each artist in the music profile. The adjustment interval is fixed to 200. Additionally a button is implemented that sets all user items to zero in one click, saving the user a lot of clicks if the intention is to start from scratch as seen in appendix D, fig. D.4. Moreover a general idea was to integrate meaningful explanations for a given recommendations using the `Implicit` library's `explain()` function, proposed by Hu et al. [19]. Unfortunately it has not been possible to integrate the functionality in the system, due to implementation difficulties. The proposed interactive adjustment conforms with a personalized experience that simultaneously support information reduction [24, 42].

[9] See appendix G for source code.

3.4 Implicit matrix factorization (iALS)

In opposition to the cost function in 2.3 that recomputes iteratively for all users in the training phase, a recalculation (`recalculate()`) is one call that fixes the item factors constant and then take the derivative of the loss function x_u [19]. A recomputation of the user-factors is followed by a recomputation of all item-factors in a parallel fashion [19]. After recomputing the user- and item-factors, the K available items with the largest value of $\hat{p}_{ui} = x_u^T y_i$ are the new recommendations, where $\hat{p}_{ui}$ is the predicted preference of user u for item i [19].

3.4.1 Hyperparameters

iALS has several hyperparameters. Increasing α adds more weight to non-zero entries in matrix R versus zero-entries. λ regularizes the model to prevent overfitting in the training phase. Additionally a number of latent factors has to be set. In general for the model, the more factors to be computed the more accurate the model but at the same time slower [19]. The last hyperparameter to be defined is the number of iterations. Values used in this experiment are $(\lambda = 20, \alpha = 40, factors = 128, iterations = 15)$, where the `implicit` libraries default settings are $(\lambda = 10, \alpha = 40, factors = 100, iterations = 15)$.

3.5 Implementation

The goal of the experiment is to compare two different user experiences, namely baseline and interactive. It must be noted that the underlying algorithm iALS is fixed for both systems, what differs is the user items that

is recommended for. The baseline recommends for all user items, whereas the interactive for an adjusted whole or subset of the user items. As online evaluation allow direct measure of the overall system [41], it can be used to understand the users' behaviour. This is achieved by tracking user clicks and storing them in the session cookie.

The system is implemented as a web application written mainly in Python using `Flask`, Javascript (`jQuery`), Bootstrap (HTML, CSS, and JS libraries) and finally deployed on Google Cloud in *Docker*. Docker is a tool that makes it easier to deploy and run applications by utilizing containers to package code, dependencies and libraries. Unlike virtual machines, Docker can be used on the same kernel as the operating system (OS) and thus only requires to be shipped with non-OS dependencies. This gives significant speedups compared to virtual machines [36]. Flask is a microframework, that comes with a very light footprint making it possible create a minimal web application with 5 lines of code [12]. The application is structured with inspiration from the Model–view–controller (MVC) architectural design pattern [20], widely used for applications dealing with user interface design. The (`model.py`) class is responsible for file reading, training the recommender model and creating the necessary data structures. A `Controller` module, contain `controller.py` class that takes care of all the backend logic, supplemented by a handful of helper classes. Finally the view is integrated in `app.py`, which is responsible for configuring and running the application. This approach is not ideal, as the `app.py` serve multiple purposes and could be additionally partitioned into `view.py`,`config.py` and a run script. Moreover, a design choice has been made to have no dedicated database. The model (iALS) is trained and stored in memory, when the application is run. User data is stored in a cookie-session object using Flask `Sessions` [48] across requests. There is one drawback of using cookies for temporary storage and that is the browser cookie limit of 4093 bytes pr. domain [43]. All session data including recommendations, clicks and input is written to .json (JavaScript

Object Notation) files and named with `uuid`+timestamp. `uidd` is a python library that ensures an unique identifiers and the timestamp ensures that no files in storage (on disk) are overwritten. An example case for this is a user that tries the system multiple times, will produce multiple files. For analysis only the most recent session file is used.

3.5.1 High level architecture

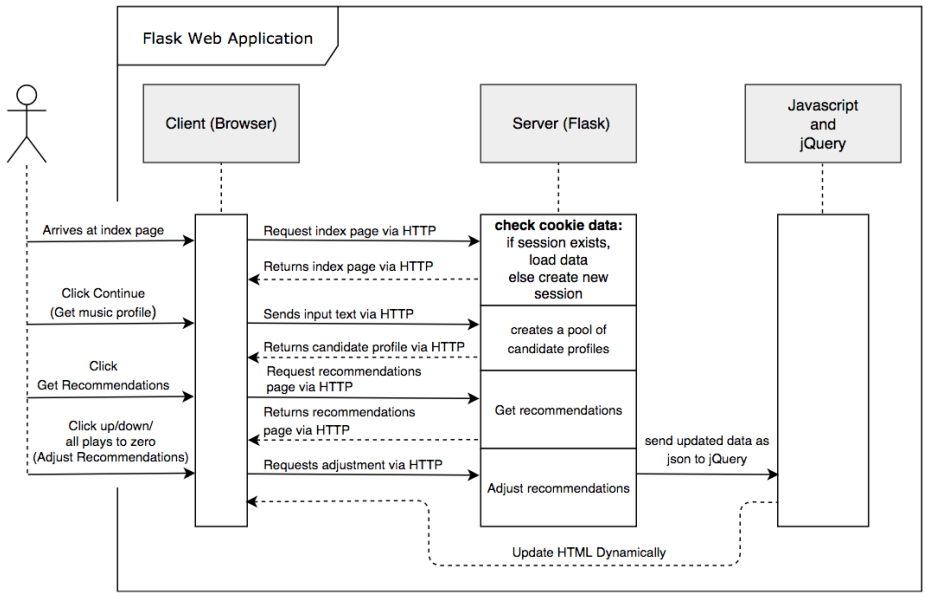


Figure 3.1: High level architecture sequence diagram

Figure 3.1 shows a selection of the most important sequence calls of the application. The purpose of the diagram is to show how the client and server communicates with the usage scenario in mind. For simplicity not all functionality is included.

3.6 Evaluation

This section describes the evaluation method of RS Experiment followed by a presentation of the actual results. The purpose of online evaluation or A/B testing is to measure the effect of a recommendation algorithm on users and their behaviour [49]. The system is continuously tested if it behaves as intended by qualitative feedback throughout the development process. The substantial and final part is the online evaluation by user testing the system and conducting a survey. The RS Experiment is evaluated by a web-based survey. The survey has been written with the main purpose of being as short and concise as possible, not jeopardizing clarity over brevity. Most importantly to avoid users to dropout and not finalize the survey. In contrast to the 15 constructs and 32 question evaluation framework proposed by Pu et al. [39] the length of the survey has been shortened to mainly support the research hypothesis. This is done by asking six different questions, covering four of the proposed constructs - counting recommendation accuracy, novelty, serendipity and transparency. The questions are covered in section 3.6.1.

3.6.1 Evaluation metrics and validating results

In order to gather feedback from users, Google Form is used to create and host the survey. The purpose of the survey is to measure two aspects:

(1) *Did the user get a profile that somewhat represents their music taste?*
(2) *How satisfied is the user with the recommendations?*

This is achieved by getting participants to answer six questions on a linear scale (Likert scale) 1-5 from "strongly disagree" to "strongly agree" (see the survey in appendix E). The survey results for the two different approaches are

compared alongside with the session data. As the Likert scale is an ordinal measure it makes sense to apply ordinal statistics (e.g. median or mode) for analysis. In an ordinal scale values are ranked according to a magnitude, without revealing their exact size in comparison to each other [7].

While asking questions can provide valuable insight, it can also result in misleading information. Thus it is important to ask neutral questions, that don't suggest a right or wrong answers. There is always a chance that people answer untruthfully, for example if the question asked is too private or if they think their true answer will put them in an unflattering position [41]. As survey and questionnaire writing is a science itself, this is the minimum this study will conform to.

Construct	**Question**
Accuracy	The music profile I was given represented my music taste
Accuracy	The tags function (option A) helped me find a representative music profile
Accuracy	The artist function (option B) helped me find a representative music profile
Novelty	The recommendations inspired me
Serendipity	The recommendations were surprising
Transparency	The recommendations had a strong relation to the music profile

Table 3.2: Survey construct

Table 3.2 shows how the different constructs has been translated into system specific questions using layman terms. Moreover, the option of answering *"I don't know"* is added to each question leaving a backdoor for participants who are not able to make up their mind or don't understand the question.

3.6.2 System-wide limitations and hyperparameters

In order to not exceed the cookie size limit some decisions has been made. This limit is exceeded when storing more than roughly 500 profile id's (6 character strings) including additional session data (see appendix F).

In order to avoid users wait too long to get a profile, some decisions has been made to decrease query-time. Timeouts are implemented in the filtering functions `filter_on_tags()` and `filter_on_artists()` in `controller.py`[10]. The loop terminates if either number of max candidates (500) has been collected or timeout is reached. `filter_on_tags()` is a general filtering function that shuffles through the dataset considering each profile as a tag frequency mapping. If at least 50% (`tags_threshold=0.5`) of the input tags are represented in the top-5 (`n=5`) frequency map, then it is a candidate. For example, if Alice likes Rock and Hip Hop, then if at least one tag is represented in a profiles top-5 e.g, (Metal 94, Doom 88, Rock 72, ...), then it is a candidate. `filter_on_artists()` initially uses `filter_on_tags()` to sort the whole dataset based on the input artists' associated top-n tags. Then the sorted profiles are examined one by one, considering only the top 10 artists (`top_n=10`) in the profile. If at least 60% of the input artists (`artist_threshold = 0.6`) are represented in top-n, then it is a candidate. All parameters are tunable and has been defined by a pragmatic approach, thus there is a lot of room for further improvement.

[10] See appendix G for source code.

4

Results

This chapter presents the findings of the RS Experiment. The experiment is evaluated by analyzing the collected data (survey and session) and comparing the two. Initially the method for analysis is described followed by the experiment in numbers and lastly the most significant findings are presented.

The analysis is conducted in an exploratory style using a Jupyter Notebook[1] environment (see the code reference in appendix G). The session-data is split into two dataframes, namely `baseline` and `interactive` which are eventually merged with the survey data. As touched upon in section 3.6.1 the collected survey data is of ordinal scale and the analysis must conform to this. For this reason simple frequency tables or count plots and bar plots are used.

	Baseline	**Interactive**	**Total**
Unique visitors	28	23	51
Surveys filled	21	19	42
Valid entries	21	12	33

Table 4.1: RS Experiment: participation in numbers

Table 4.1 shows the number of total participants. While 51 people participated, roughly 20% of those failed to fill in the survey. Moreover 8 out of

[1] `https://jupyter.org/`

23 participants failed to interact with the recommendations. Thus it only makes sense to consider the remaining 15 participants of which 12 filled in the survey. This brings the number of valid entries down to a total of 33 participants, resulting in an unequal sample size. This tiny collection of data must be treated with caution, as it can easily lead to wrong conclusions.

4.1 Findings

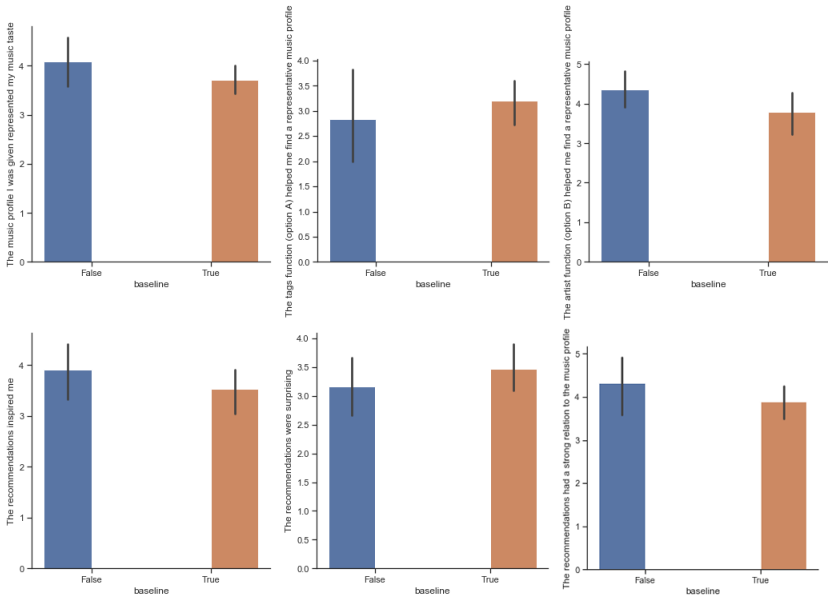


Figure 4.1: Bar plots showing baseline vs. interactive survey results

baseline = True (right bar) corresponds to the baseline recommender
baseline = False (left bar) is the interactive recommender.

The numeric scale is linear 1-5 (strongly disagree - strongly agree). The mean is taken by default and the solid color shows the mean value, whereas the black line is the error bar which represents the uncertainty measured. The error bars can be used to measure the statistical significance e.g., if there is no overlap in the error bar between bars, then it might indicate that there is a statistical difference.

Looking at the bar plots in fig. 4.1 all of the error bars are overlapping. This tells that the data is clustered around the mean. The larger the error bar, the larger the spread of values. The collected data does not show any significant difference between the two tested systems. It is important to note that for both systems the participants on average were satisfied with their matching

music profile (fig. 4.1, top left corner) and that the artist filtering function (option B) provided better results than the tags filtering function (option A)[2].

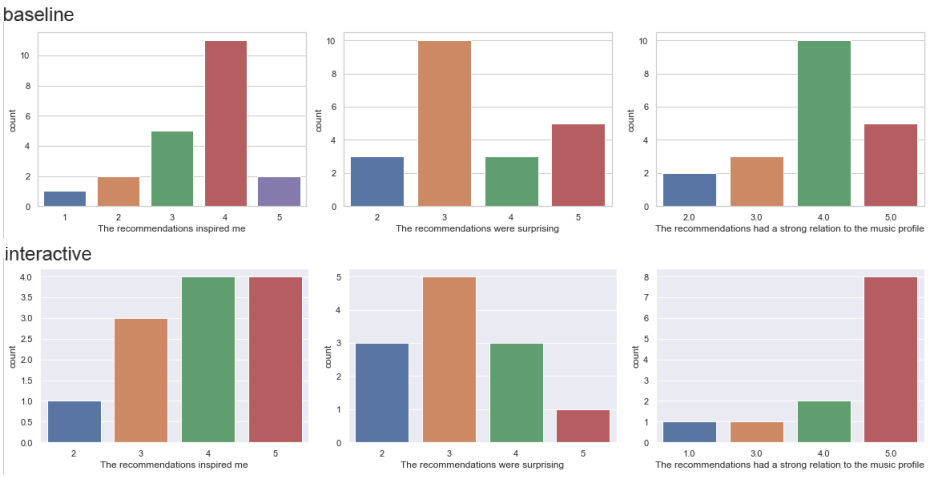


Figure 4.2: Count plots showing baseline vs. interactive valid response count

The count plots shows the responses in numbers. Upper row is baseline, lower is interactive. The numeric scale is linear 1-5 (strongly disagree - strongly agree). It has not been possible to include the response count for "I don't know", due to formatting issue in the source.

The result of the experiment do not support the research hypothesis (see section 1.2). However, due to the small sample size, the experiment is inconclusive. Further research is needed with a larger sample size, in order to draw a final conclusion. Lastly, the data does not reflect any substantial improvement in regards to improving transparency.

[2]See jupyter notebook: `rs_experiment_analysis.ipynb`

5

Discussion

The most important finding of this study is coming to realize how extremely difficult it is to design recommender systems and surrounding experimental study. The discussion presented is two-fold, where the first part discusses potential weaknesses identified during the project and the second part presents suggestions for future work.

The objective of this paper is to improve transparency in RSs by utilizing visualization techniques that aim to show how recommendations are computed. Due to lack of insights, it has not been possible to reveal any findings of significance. The subsequent goal of allowing users to adjust recommendations was implemented, with room for further improvement (see section 5.2). The proposed interactive system conforms to a personalized experience, simultaneously supporting information reduction by allowing the user to get recommendations for a subset versus the users complete user items (history). As the conducted experiment did not provide any meaningful insights to whether the proposed solution improves transparency or not, it is important to discuss potential weaknesses and alternatives in terms of future work.

5.1 Potential weaknesses

This section presents weaknesses identified during the project.

5.1.1 System design

A weak spot of the created web application prototype is the lack sufficient resources and knowledge to create a holistic, intuitive and thoroughly tested design. In general extensive qualitative studies in the design phase of the system would help enhancing the user experience further, closing potential gaps where users fail to complete the experiment (dropout). Despite continuous improvements of the system throughout the project, the prototype was deployed with usability flaws. An example of this is the system's inability to ensure that users interacting with the interactive recommender system, before filling in the survey, as noticed in chapter 4. However, for the scope of this thesis, it seems unrealistic to talk about interdisciplinary teamwork as a solution, but rather a suggestion for future work.

5.1.2 User studies - online evaluation

Conducting user studies is hard. In hindsight it can be questioned whether the choice of translating and reducing the survey construct and questions [39] suggested by Pu et al. was a good idea. Other than approaching the collected data with utmost care, participation bias should be considered. As the RS Experiment was distributed to participants by the author throughout his social and professional network, participation bias can be expected[1]. In

[1]Participation bias or a non-response bias is when data become non-representative, because participants disproportionately possess certain traits which affect the outcome [13]

terms of online surveys, it can be viewed as cases where participants don't have incentive to present their true preferences (further elaborated in section 5.1.4).

5.1.3 Cold start problem

The cold start problem in the RS Experiment is relevant to discuss because it has huge importance for the experiment as a whole. The first thing to question is whether it makes sense match a participant with a profile from a dataset. It worked for this experiment, but it has not been investigated or questioned what potential side effects it might bring. It would be ideal to have conducted the experiment on real-life settings, such as based on participants Spotify profiles.

5.1.4 Minimize participation bias

In extension to the weaknesses presented in section 5.1.2 and 5.1.3 it is important establish a discussion around the cold start problem. The problem lies in the way this experiment is designed. Take for instance a user Charlie, who likes Justin Bieber. However, he is reluctant to express this opinion, because he finds it a bit embarrassing. As Charlie enters the experiment and gets a matching profile, he will then do whatever it takes not to disclose his real preference (Justin Bieber). On the other hand, if say Spotify was in charge of this experiment and Charlie was already an existing Spotify user, then his data would reveal his secret. As pictured, the problem in this study evolves at the cold start where a new user has to be matched to a profile. There are many ways to alleviate this problem and thus minimizing participation bias.

5.1.5 Database

A lot of decisions were made along the way to avoid exceeding the browser's cookie size limit, restricting the amount of information tracked, stored and ultimately collected pr. participant. This is most evident in the cold start problem solving, where profiles from the dataset were filtered based on users' input. This problem can be circumvent by using a dedicated database.

5.1.6 System-wide hyperparameters

The system-wide hyperparamters from section 3.6.2 could potentially represent a weakness if the user is not able to retrieve a matching profile. Thus it would be useful to tune applicable parameters. Moreover, the adjustment interval in the interactive recommender is fixed to 200. This value was found to work well during development where the application was user tested on a small scale (5 people).

5.1.7 Visual inconsistency

Lastly, there is a recognizable difference between the way the music profile (user items) is shown in the recommendation step between the two approaches (see fig. D.4 and fig. D.8). The baseline shows a profile sorted by plays in descending order, whereas the interactive shows the profile alphabetically sorted on artists. This inconsistency can mislead or confuse participants resulting in potential dropouts.

5.2 Future work

Other than improving and addressing the presented weaknesses, this section suggests future work directions. The experiment should be conducted and compared across different datasets (MovieLens 20M, Last.fm 360K Users Dataset, Million Song Dataset etc.) Additionally a newer and bigger music dataset would be ideal, preferably in the magnitude of millions of users.

Looking forward it is not a question of either or, when it comes to advanced math (iALS) and interactive recommender systems. On the contrary the best of both worlds must be combined to create the future of transparent and better performing recommender systems. Especially as tag clouds and other association techniques influence the metadata of every entry in the dataset.

5.2.1 Engaging users in active information reduction

In order to actively engage the user in information reduction a more stringent approach should be tested. Whereas as this study propose a loose approach that allows the user to freely adjust user items, not necessarily resulting in information reduction. The approach taken does not guarantee information reduction and thus might result in choice overload, in terms of too many user items to adjust and buttons to click. Moreover, measuring choice overload in the user survey might have provided an indication. A stringent approach would force the user to reduce information, by for example matching a user's current criteria, as proposed by Richthammer et al. [42] (see fig 2.3 & 2.4). Additionally, it makes sense to supplement information reduction with insightful explanations for why a given recommendation has been made, supporting a better understanding of the algorithms decision

making and transparency of the system.

5.2.2 Explaining recommendations

As described in section 3.3 it has not been possibly to utilize the `explain()` function, resulting in limited visual interaction in the RS Experiment. This function provides a sorted representation of the user items that constitutes the basis for a given recommendation, as described in section 2.4. It would be interesting to measure how well this insight improve transparency of recommender systems.

5.2.3 Performance speedups

An unknown factor is whether the ∼1.75 second query time pr. adjustment click has refrained users from interacting with the system. If this shows to be the case, luckily speedups are possible for both training and the `recalculate()` step by utilizing GPU (CUDA[2]).

[2]urlhttps://developer.nvidia.com/cuda-zone

6

Conclusion

In this paper the author designed and carried out an experimental study that measures satisfaction rate between two different recommender approaches. Despite an inconclusive experiment, a web application prototype that implements visual data interaction was created. Allowing the user to adjust recommendations dynamically. The proposed interactive system conforms to a personalized experience, simultaneously supporting information reduction by allowing the user to get recommendations for a subset versus the users complete user items (history). The study finds that the proposed solution is defined to loosely, leaving the information reduction optional in contrast to mandatory. Thus, it is not guaranteed that users ends up with a simplified personal user experience.

Promising aspects of future work include engaging users in active information reduction, in opposition to the loose approach taken in this project. Moreover, alleviating the cold start problem by establishing a real-life setting would minimize participation bias increasing the validity of research.

Bibliography

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

[2] Xavier Amatriain. Mining large streams of user data for personalized recommendations. *SIGKDD Explor. Newsl.*, 14(2):37–48, April 2013.

[3] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 43–52. IEEE, 2007.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[5] J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. Recommender systems survey. *Know.-Based Syst.*, 46:109–132, July 2013.

[6] Craig Boutilier. Toward user-centric recommender systems, keynote talk. `http://ifaamas.org/Proceedings/aamas2018/pdfs/p2.pdf`, 2018. Online, accessed 2018-11-11.

[7] Alberto Cairo. *The Truthful Art: Data, Charts, and Maps for Communication*. New Riders Publishing, Thousand Oaks, CA, USA, 1st edition, 2016.

[8] O Celma. Last.fm dataset - 360k users (www.last.fm). `https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/`

`lastfm-360K.html`, 2010.

[9] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[10] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1):125–150, Apr 2012.

[11] Michael D. Ekstrand, Daniel Kluver, F. Maxwell Harper, and Joseph A. Konstan. Letting users choose recommender algorithms: An experimental study. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 11–18, New York, NY, USA, 2015. ACM.

[12] Flask. Flask documentation. `http://flask.pocoo.org/docs/1.0/`, Online, accessed 2018-12-10.

[13] F. J. Fowler. *Applied Social Research Methods: Survey research methods (4th ed.)*. SAGE Publications, Inc., Thousand Oaks, CA, 2009.

[14] Ben Frederickson. Fast python collaborative filtering for implicit feedback datasets. `https://github.com/benfred/implicit`, 2016–.

[15] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

[16] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *Commun. ACM*, 53(6):59–67, June 2010.

[17] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00,

pages 241–250, New York, NY, USA, 2000. ACM.

[18] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004.

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

[20] iandavis.com. What are the benefits of mvc? `http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/`, Online, accessed 2018-12-10.

[21] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA, 2010. ACM.

[22] Christopher C Johnson. Logistic matrix factorization for implicit feedback data.

[23] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: A bibliography. *SIGIR Forum*, 37(2):18–28, September 2003.

[24] Bart P. Knijnenburg and Martijn C. Willemsen. *Evaluating Recommender Systems with User Experiments*, pages 309–352. Springer US, Boston, MA, 2015.

[25] Bart P. Knijnenburg, Martijn C. Willemsen, Zeno Gantner, Hakan Soncu, and Chris Newell. Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, 22(4):441–504, Oct 2012.

[26] Joseph A. Konstan and John Riedl. Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22(1):101–123, Apr 2012.

[27] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[28] Dennis Leszkowicz. Music recommender systems: Automatic playlist continuation. `https://www.overleaf.com/read/rsfkyhgdnhfw`, 2018.

[29] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan 2003.

[30] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA, 2011.

[31] Brian McFee, Thierry Bertin-Mahieux, Daniel P.W. Ellis, and Gert R.G. Lanckriet. The million song dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 909–916, New York, NY, USA, 2012. ACM.

[32] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[33] Sean M. McNee, John Riedl, and Joseph A. Konstan. Making recommendations better: An analytic model for human-recommender interaction. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1103–1108, New York, NY, USA, 2006.

ACM.

[34] Andreas Mueller. Wordcloud for python. https://github.com/amueller/word_cloud, 2016–.

[35] n/a. Acm fat* 2019 call for papers. https://fatconference.org/2019/cfp.html, 2018.

[36] Opensource.com. What is docker? https://opensource.com/resources/what-docker, Online, accessed 2018-12-10.

[37] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408, Dec 1999.

[38] Michael J. Pazzani and Daniel Billsus. *Content-Based Recommendation Systems*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[39] Pearl Pu, Li Chen, and Rong Hu. A user-centric evaluation framework for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 157–164, New York, NY, USA, 2011. ACM.

[40] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4):317–355, Oct 2012.

[41] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[42] Christian Richthammer, Johannes Sänger, and Günther Pernul. Inter-

active visualization of recommender systems data. In *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems*, SHCIS '17, pages 19–24, New York, NY, USA, 2017. ACM.

[43] By Iain Roberts. Browser cookie limits. `http://browsercookielimits.squawky.net/`, Online, accessed 2018-12-10.

[44] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[45] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[46] J. Ben Schafer, Dan Frankowski, Jonathan L. Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The Adaptive Web*, 2007.

[47] Christin Seifert, Barbara Kump, Wolfgang Kienreich, Gisela Granitzer, and Michael Granitzer. On the beauty and usability of tag clouds. *2008 12th International Conference Information Visualisation*, pages 17–25, 2008.

[48] Flask Sessions. Flask documentation. `http://flask.pocoo.org/docs/1.0/api/#flask.session`, Online, accessed 2018-12-10.

[49] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011.

[50] Rashmi Sinha and Kirsten Swearingen. The role of transparency in recommender systems. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, pages 830–831, New York, NY, USA, 2002. ACM.

[51] Spotify. Spotify recsys challenge 2018. `https://recsys-challenge.spotify.com`, 2018.

[52] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *Acm Sigkdd Explorations Newsletter*, 9(2):80–83, 2007.

Appendices

A

Last.fm 360K Users Dataset

```
In [13]: df = pd.read_csv("dataset_lastfm360k/lastfm-dataset-360K/usersha1-profile.tsv",
                          sep="\t", names=['userId','gender','age','country','signupDate'])

In [14]: df.head(1)

Out[14]:
```

	userId	gender	age	country	signupDate
0	00000c289a1829a808ac09c00daf10bc3c4e223b	f	22.0	Germany	Feb 1, 2007

```
In [17]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359347 entries, 0 to 359346
Data columns (total 5 columns):
userId        359347 non-null object
gender        326572 non-null object
age           284447 non-null float64
country       359347 non-null object
signupDate    359347 non-null object
dtypes: float64(1), object(4)
memory usage: 13.7+ MB

In [15]: df.isna().sum()

Out[15]: userId            0
gender        32775
age           74900
country           0
signupDate        0
dtype: int64
```

Figure A.1: Last.fm 360K Users Dataset, profile missing features.

B

Netflix, cold start problem (screenshot)

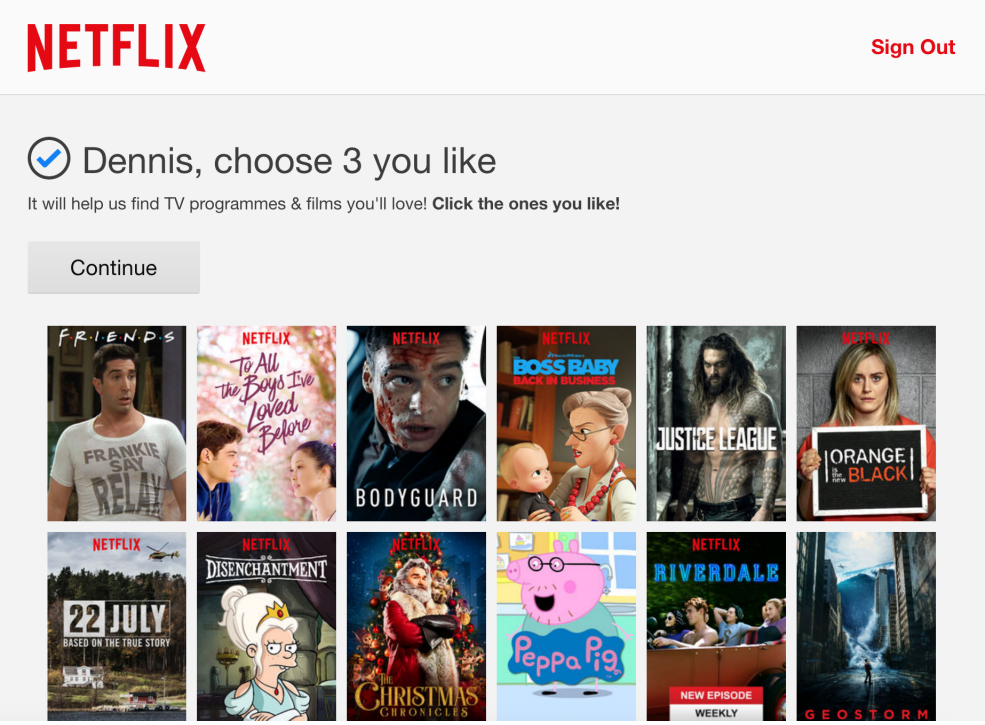


Figure B.1: Netflix overcoming the cold start problem on sign up.

C

Last.fm 360K Users Dataset tag cloud

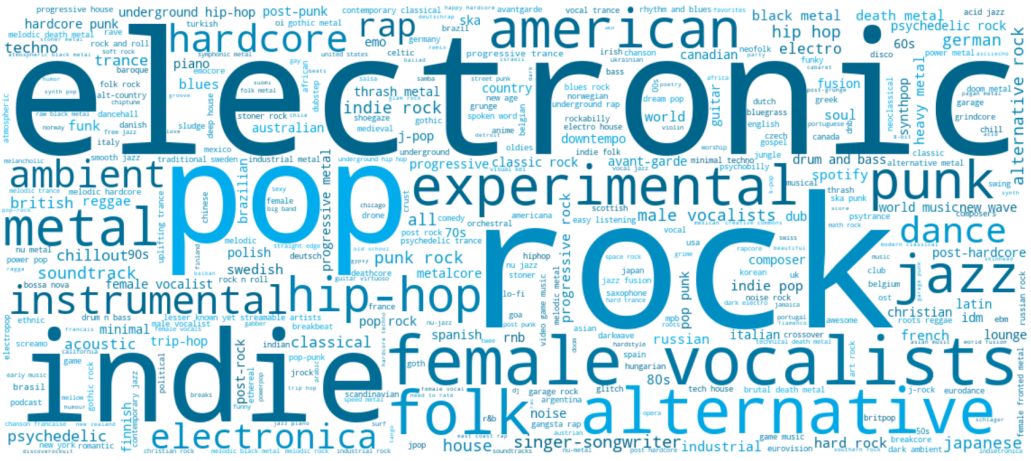


Figure C.1: Tag cloud visualization of the Last.fm 360K Users Dataset.

NB! the color of the tags means nothing in this context, it is added to avoid boring one color tag cloud.

D

RS Experiment web application in use case in screenshots

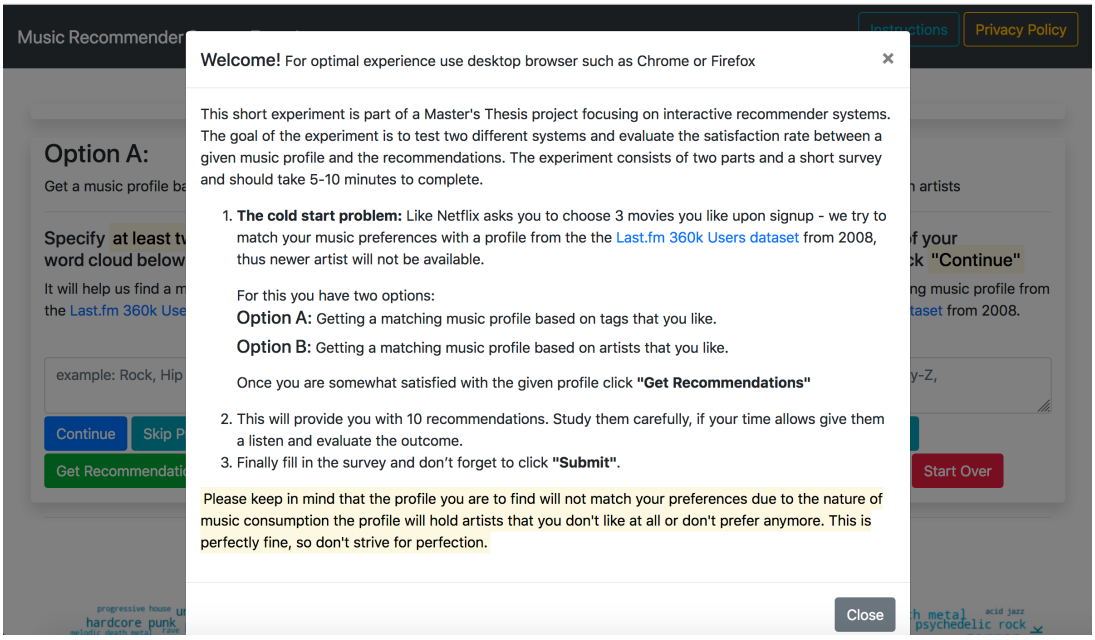
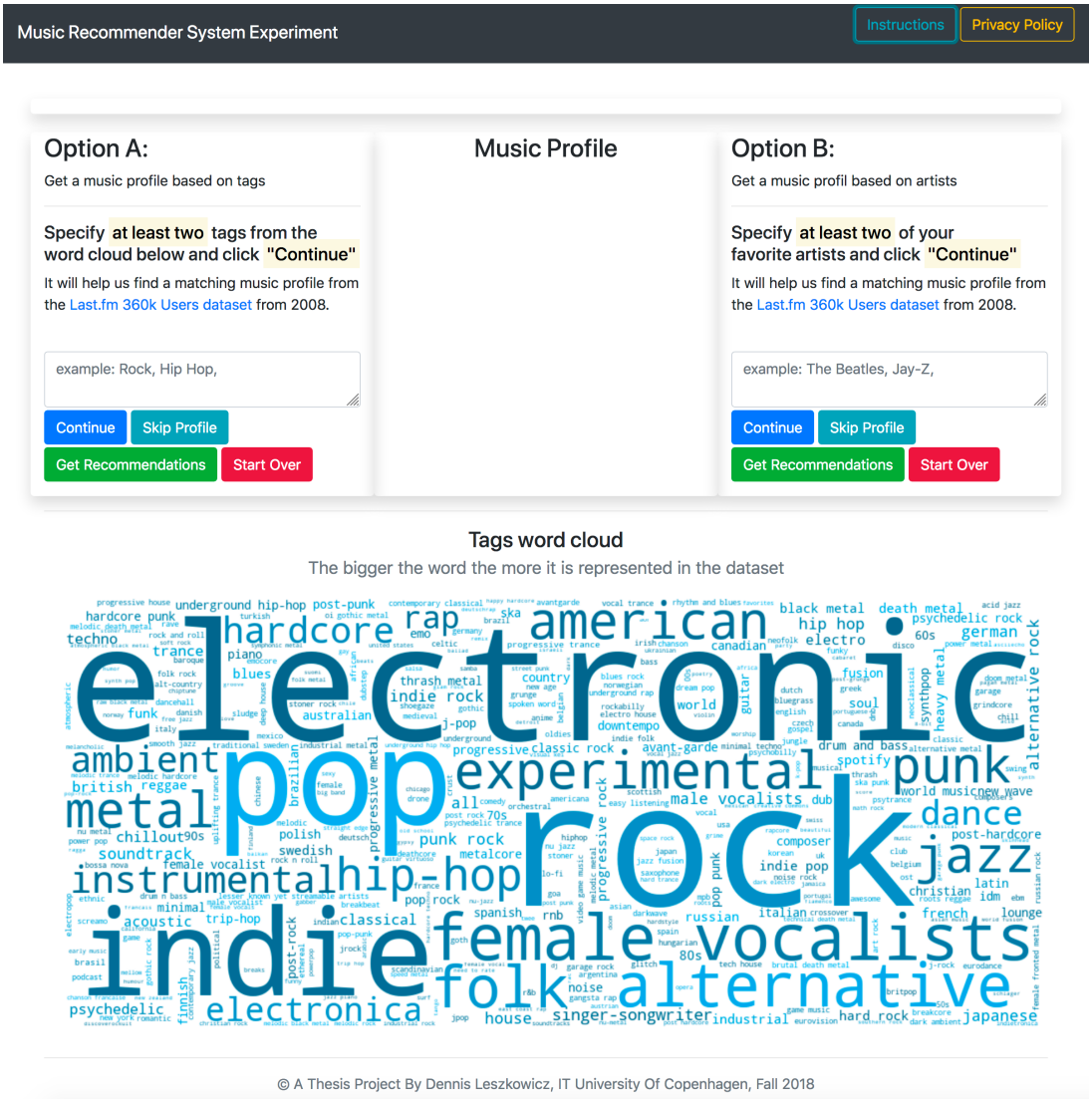
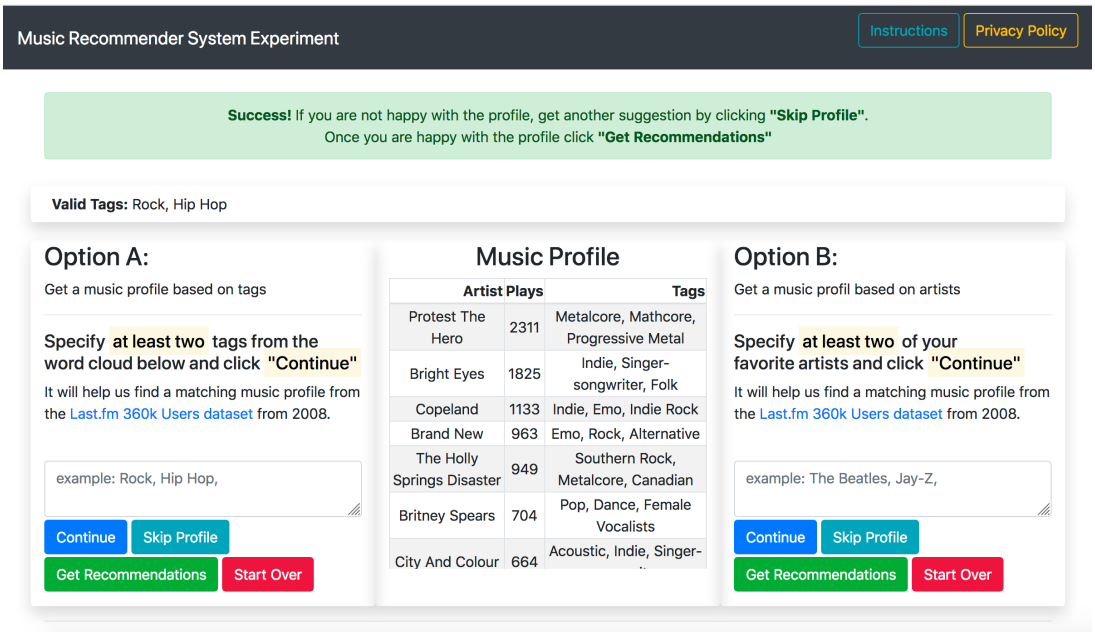


Figure D.1: Web App - Landing page (Instructions)

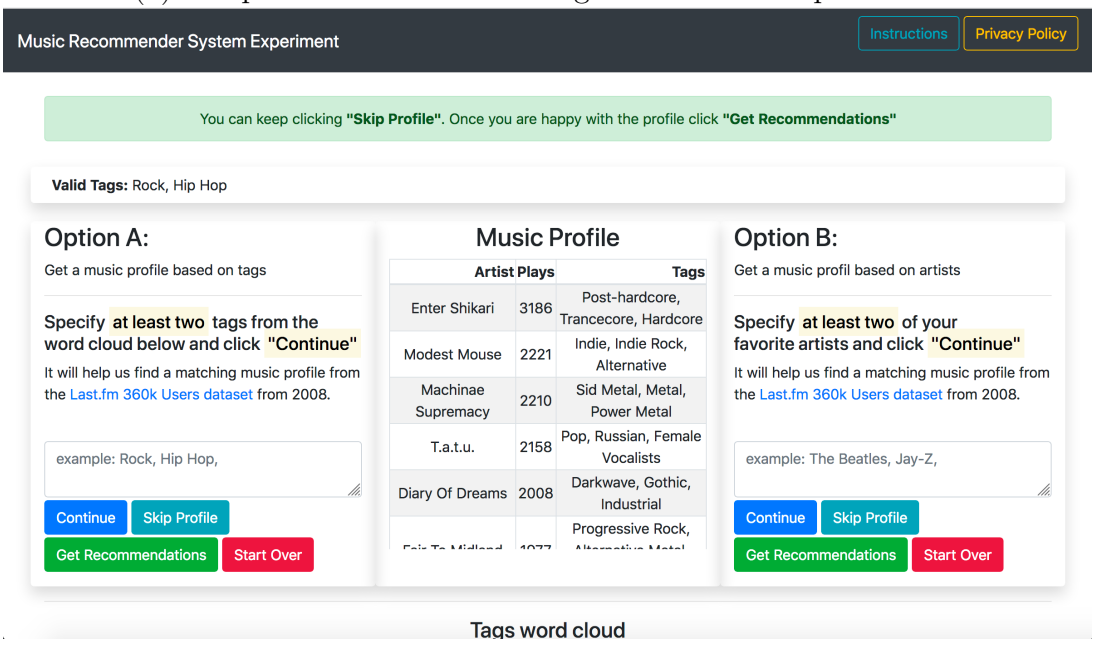


(1) Tags are provided in the text field followed by clicking "Continue". This returns a candidate profile.

Figure D.2: Web App - Step 1: Getting a matching profile (1).

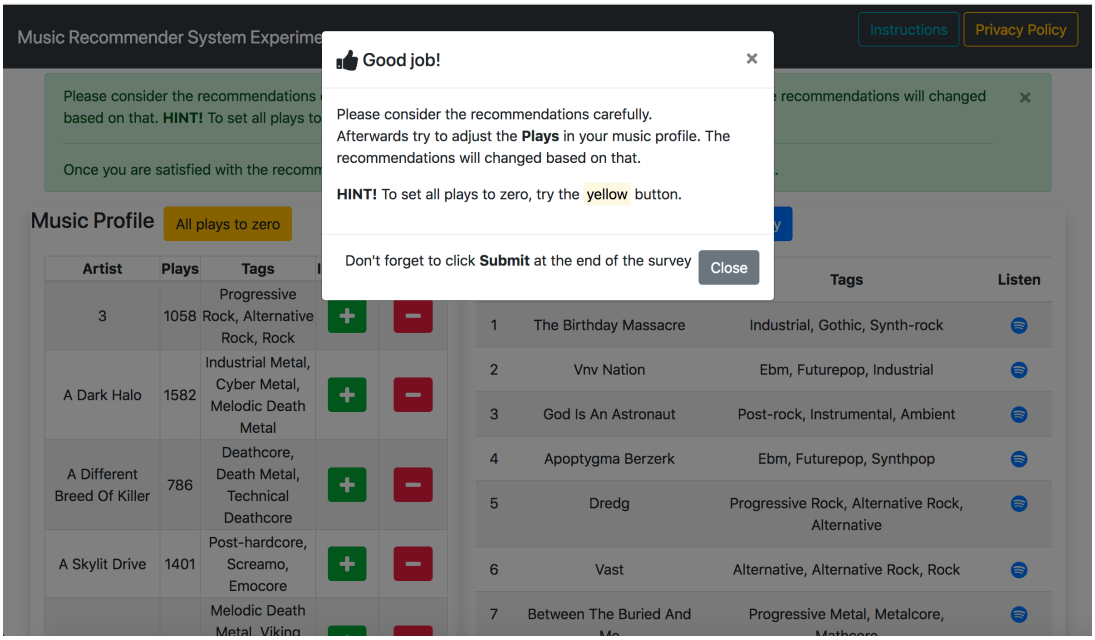


(2) "Skip Profile" browses through the candidate pool.



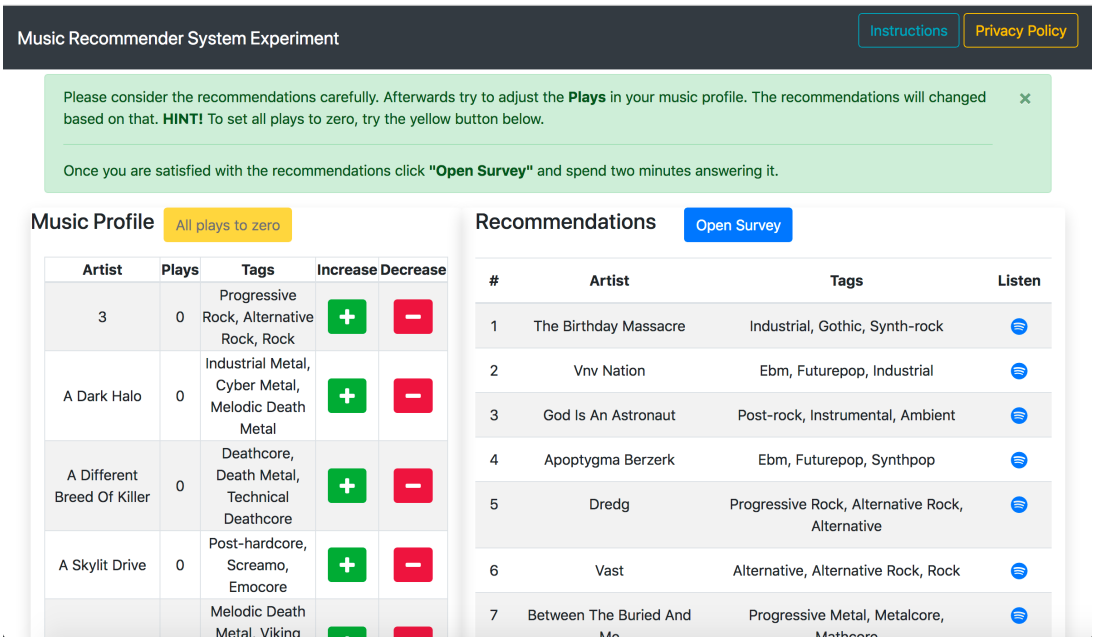
(3) Next profile is displayed.

Figure D.3: Web App - Step 1: Getting a matching profile (2).

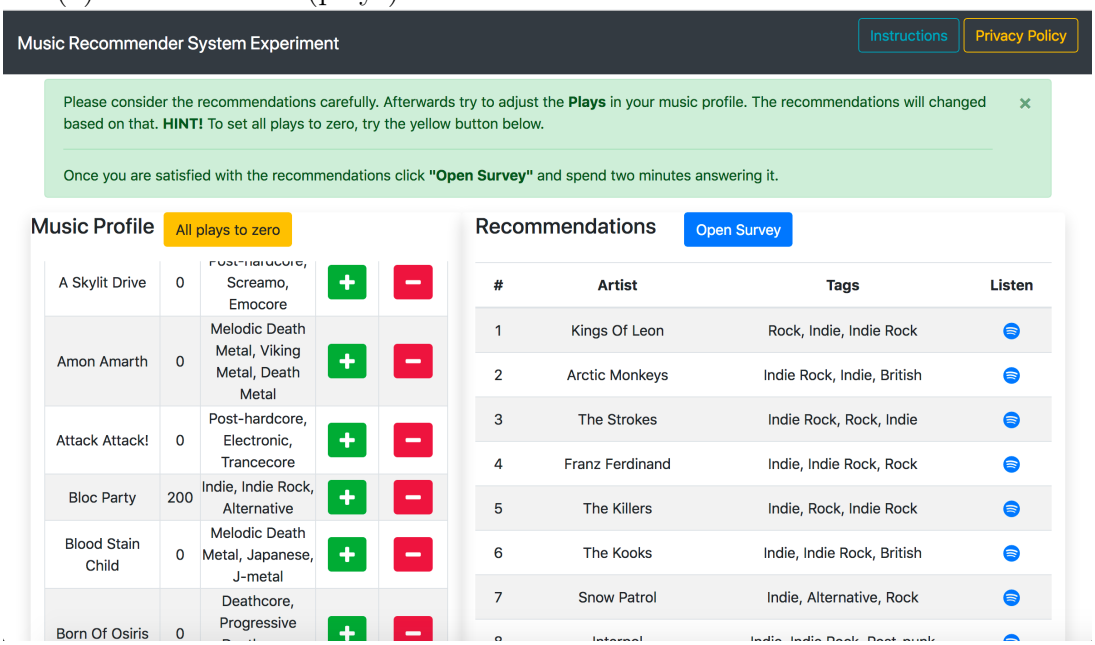


"Get Recommendations" is clicked and instructions appear in a modal.

Figure D.4: Web App - Step 2: Get recommendations

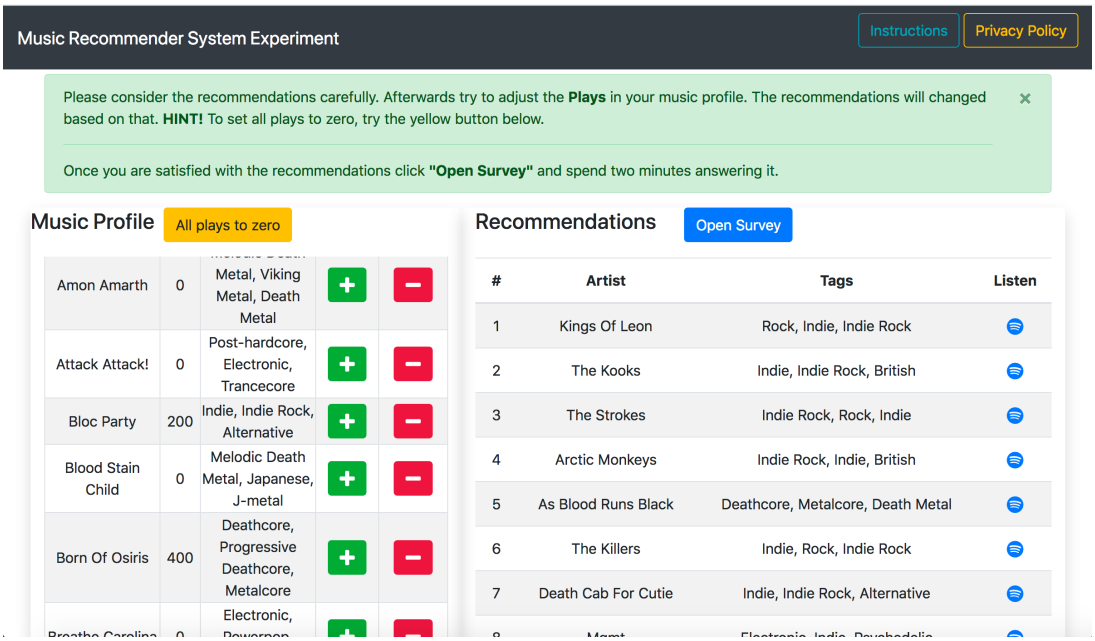


(1) All user items (plays) are set to 0.

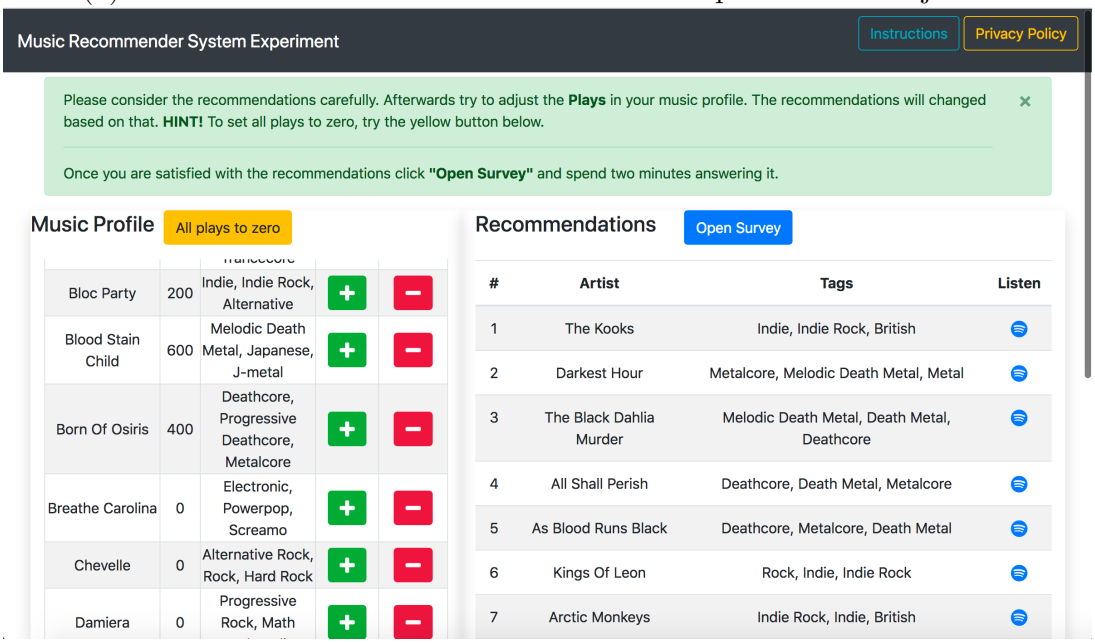


(2) A subset of artists from the music profile are adjusted.

Figure D.5: Web App - Step 3: Adjust recommendations (1)



(1) A subset of artists from the music profile are adjusted.



(2) A subset of artists from the music profile are adjusted.

Figure D.6: Web App - Step 3: Adjust recommendations (2)

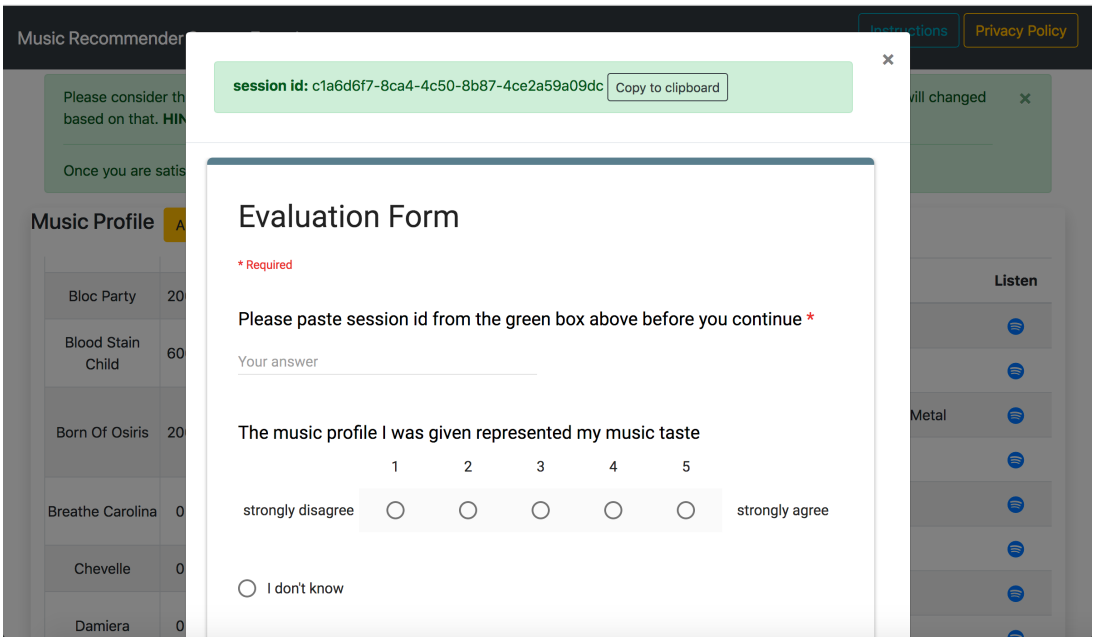
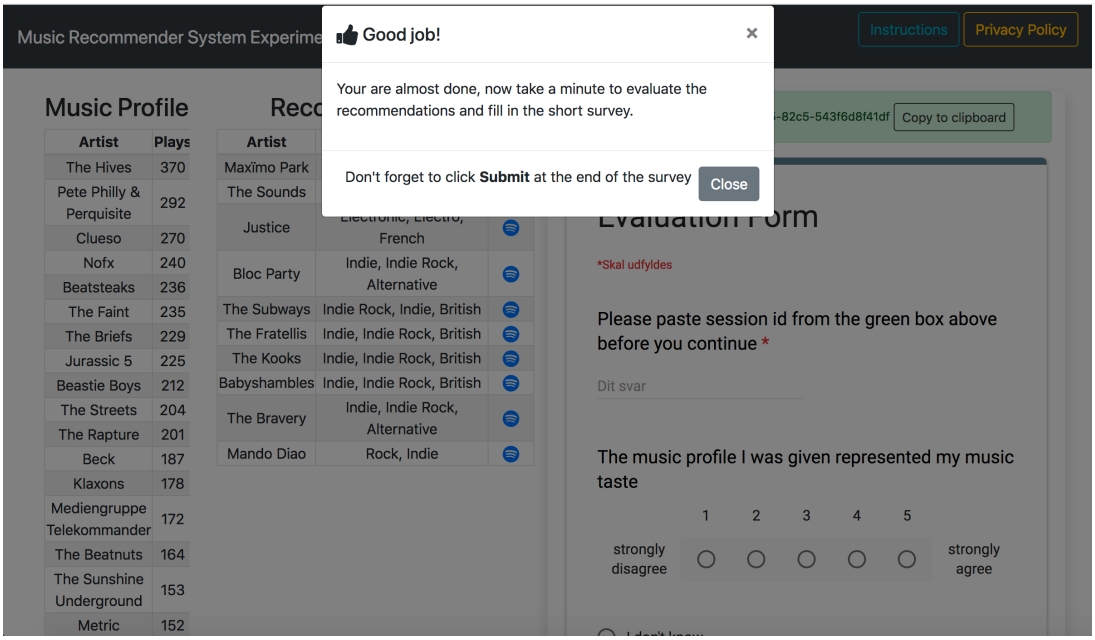
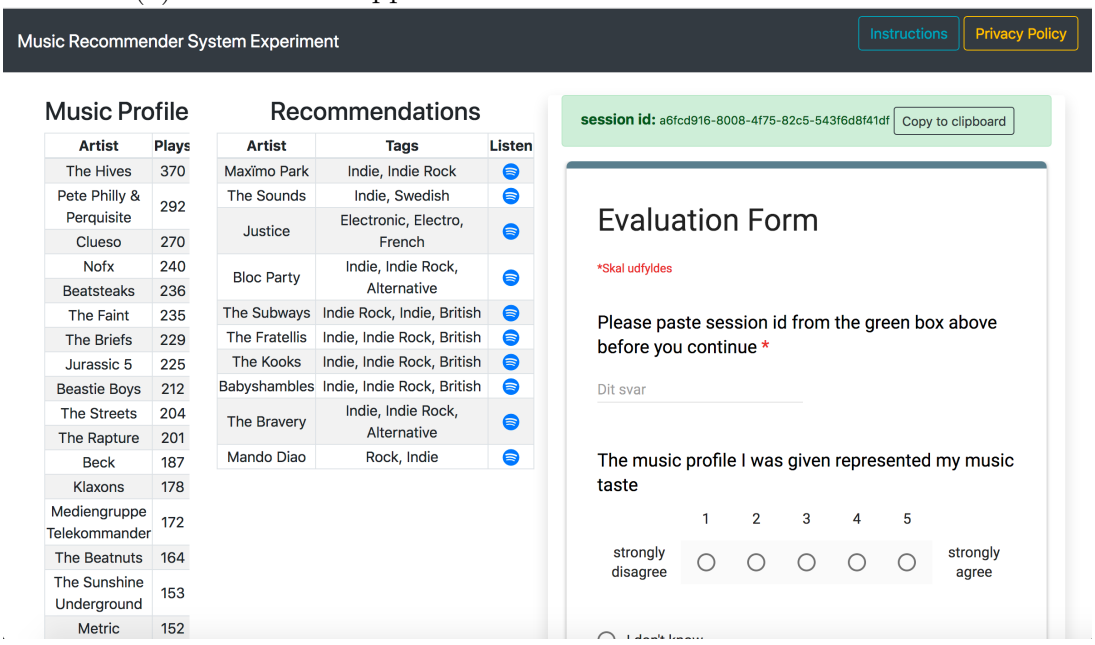


Figure D.7: Web App - Step 4: Open and complete survey



(1) Instructions appear in a modal.



(2) The baseline recommendations.

Figure D.8: Web App - Step 3: Evaluate recommendations (baseline)

E

Evaluation form (survey)

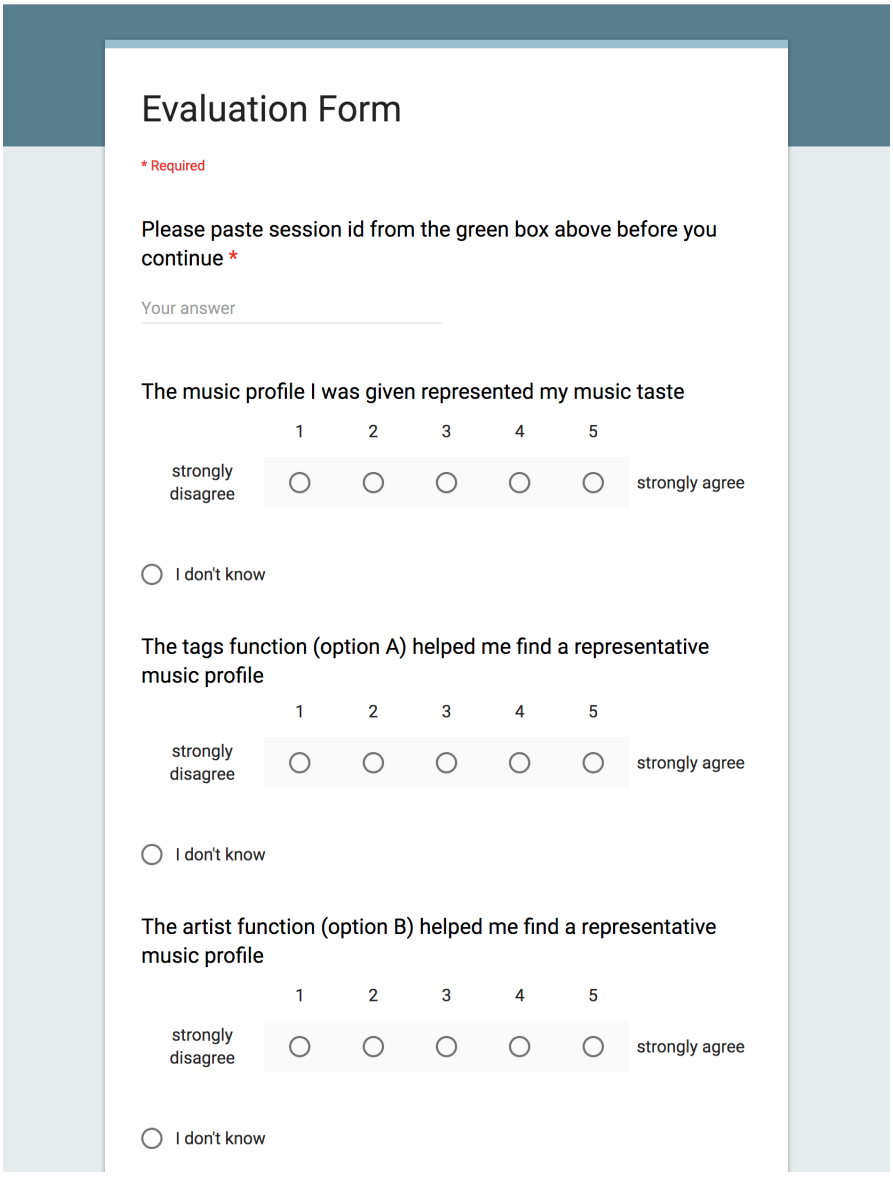


Figure E.1: Google Form - evaluation form (survey) (1)

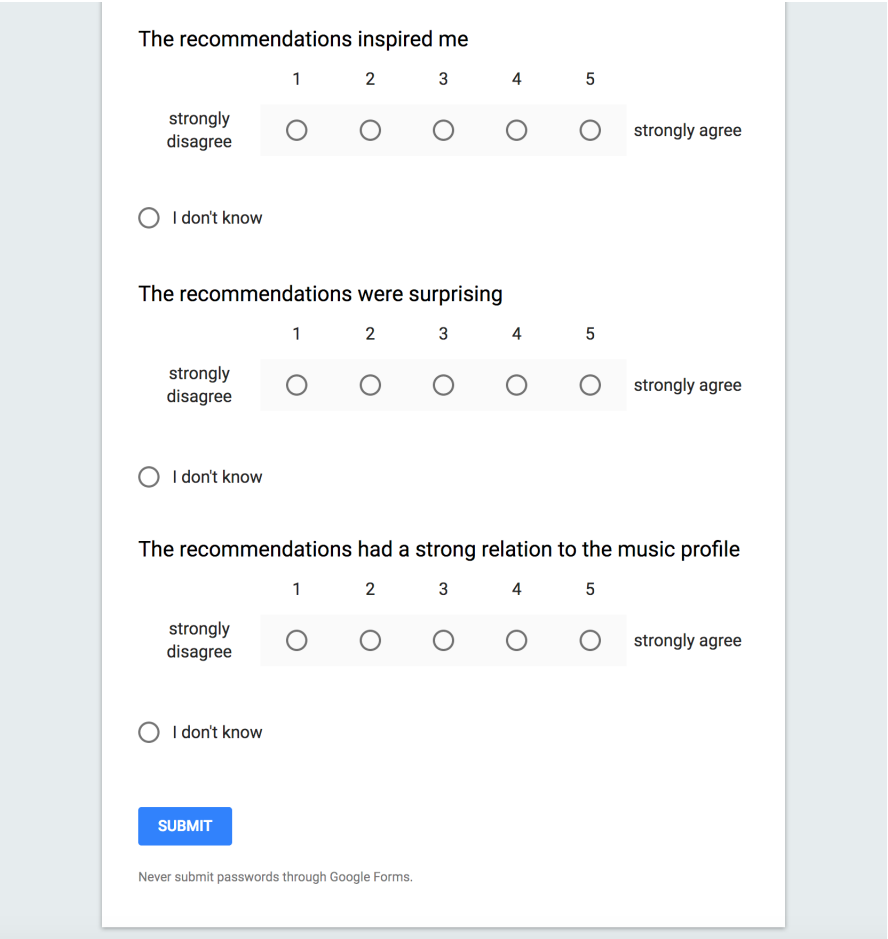


Figure E.2: Google Form - evaluation form (survey) (2)

F

User session dictionary

```
{
"username": uuid,
 "baseline": boolean,
 "new_session": True,
 "valid_tags": '',
 "invalid_tags": '',
 "tag_threshold": False,
 "artist_threshold": False,
 "valid_artists": '',
 "invalid_artists": '',
 "candidate_pool": '',
 "num_candidates": '',
 "final_profile": '',
 "plays_row": '',
 "plays_data": '',
 "recommendations": '',
 "tags_clicks": '0',
 "artists_clicks": '0',
 "boost_up_clicks": '0',
 "boost_down_clicks": '0',
 "clear_data_clicks": '0',
 "zero_all_clicks": '0'
 }
```

Figure F.1: A new session cookie (stored as a python dictionary)

G

Code and data overview

File (path: `notebooks/`)	**Purpose**
`get_artist_tags_api.ipynb`	Collecting artist tags
`make_tag_aggregations.ipynb`	Tag aggregations
`make_user_profile_dict_pickle.py`	Tag aggregations
`lastfm_tagclouds.ipynb`	Tag cloud creation
`recommenders.ipynb`	Skeleton of the two implemented recommenders
`lastfm_explain_function.ipynb`	Showing how the iALS `explain()` function works

Table G.1: Notebooks and scripts overview

File (path: `flaskapp/`)	**Purpose**
`README.md`	Instructions
`app.py`	Configure and run app
`controllers/controller.py`	Controls all backend logic
`controllers/fileWriter.py`	Session data to file
`controllers/inputHelper.py`	Trims and validates user input
`controllers/sessionController.py`	Creates and maintains session data
`model/alertMessages.py`	Alert messages (frontend)
`model/model.py`	Loads data and trains the model
`static/`	Dependencies and libraries
`static/js/code.js`	Interactive recommender `jQuery` code
`template/`	All html files
`views/viewHelper.py`	View helper

Table G.3: Web Application code overview

File	**Purpose**
`rs_experiment_analysis.ipynb`	Analysis of RS Experiment (chapter 4)
`data/recsys survey.csv`	Survey results
`data/user_sessions/`	Collection of `.json` session data files

Table G.4: RS Experiment - results analysis
folder path: `experiment_results/`

File (path: `flaskapp/data/`)	**Purpose**
`artist_tags_dict.pickle`	Dictionary of user profile tag frequencies
`lastfm_360k.hdf5`	Cleaned Last.fm 360K Users Dataset
`profile_dict.pickle`	`profile_index:artist_indices`
`profile_tag_frequency_dict.pickle`	
`tag_set.pickle`	Set of unique tags
(path: `my_data/`)	
`lastfm_artist_tags.csv`	Raw artist tags
`lastfm_tags_cleaned.csv`	Cleaned artist tags
`artists.csv`	Artist names for `get_artist_tags_api.ipynb`

Table G.2: Data overview