Context-Aware Group Recommendation Systems

Simen Fivelstad Smaaberg

Master of Science in Computer Science
Submission date: June 2014
Supervisor: John Krogstie, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Context-Aware Recommendation Systems (CARS) model and predict user preferences by incorporating pre-existing contextual information into the recommendation process by explicitly incorporating additional categories of data. In a group recommendation system, this context is applied to a recommendation process where recommendations are given to a group of people instead of to individual persons.

A prototype of such a system has been developed, looking at concert recommendation for groups of people. This thesis aims to improve the recommendations given by the prototype and improve its usability.

New functionality should be added to the prototype and it should be evaluated in a rigorous manner. The project is expected to follow a design science research approach, producing and evaluating an artifact in a scientifically sound manner. Code produced should be made available under an open source license. It is preferred that the project report is written in English. The results from a good project should be possible to be used as a basis for developing a scientific publication.

Abstract

For a group of friends going to a concert or a festival, finding concerts that everyone is happy with can be challenging as everyone have their own preferences and wishes when it comes to music.

In this thesis, a prototype of a group recommendation system for concerts is presented to solve this issue. The prototype is context sensitive; it takes a user's location and time into account when giving recommendations. The prototype implements three algorithms to recommend concerts by taking advantage of what users have listened to before: a collaborative filtering algorithm ($\boldsymbol{k}$-Nearest Neighbor), a Matrix Factorization algorithm, and a Hybrid approach of these two. The thesis was written following the Design Science Research paradigm.

The thesis covers the design and implementation of the prototype in addition to a brief state of the art of the recommendation systems literature. The usability of the prototype was evaluated using the System Usability Scale and a user centered evaluation was performed to evaluate the quality of recommendations. The results from the usability evaluation shows that users generally were satisfied with the usability of the prototype. The results from the Quality Evaluation shows that the $\boldsymbol{k}$-Nearest Neighbor and Hybrid approach produces satisfactory results whereas the Matrix Factorization implementation is lagging a bit behind. The users testing the prototype were generally satisfied with the quality of recommendations, however further evaluation is needed to draw any final conclusions.

Sammendrag

Som en gruppe personer som er på en festival og vil gå på en konsert, det å finne konserter alle er fornøyde med kan være en utfordring siden alle har sine egne preferanser og ønsker når det kommer til musikk.

I denne oppgaven vil en prototype av et gruppeanbefalingssystem for konserter bli presentert for å gjøre dette lettere. Prototypen er kontekstsensitiv; den tar en brukers lokasjon og tid med i beregningen når den lager anbefalinger. Prototypen implementerer tre algoritmer for å anbefale konserter ved å bruke data om hva brukere har hørt på før: en *collaborative filtering* (k-Nearest Neighbor); en *matrisefaktoriseringsalgoritme*; og en kombinasjon av disse. Oppgaven ble gjennomført ved å følge Design Science Research metodologien.

I denne oppgaven presenteres designet og implementeringen av prototypen i tillegg til en kort state of the art av anbefalingssystemlitteraturen. Prototypens brukervennlighet ble evaluert med en System Usability Scale, og en brukersentrert evaluering er brukt til å evaluere kvaliteten på anbefalingene. Resultatene viser at brukerne generelt var fornøyde med brukervennligheten til prototypen. Kvalitetsevalueringen viser at *k-Nearest Neighbor*- og *Hybrid* algoritmen produserte tilfredsstillende resultater, mens *matrisefaktoriseringen* er litt bak når det kommer til dette. Generelt var brukerne som testet prototypen fornøyd med anbefalingene gitt, men testing med flere brukere må gjøres før endelige konklusjoner kan trekkes.

Acknowledgements

This Master's thesis was written as my final report for the Master of Science (MSc) degree in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor John Krogstie for his guidance and ideas when developing and writing this thesis. I would also like to thank my friends and family for their encouragement and support throughout my five years at NTNU.

Simen Fivelstad Smaaberg

Contents

List of Figures

List of Tables

Abbreviations

MVVM	Model View ViewModel
IDE	Integrated Development Environment
REST	Representational State Transfer
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
AJAX	Asynchronous JavaScript and XML
JAX-RS	Java-API for RESTful Web Services
SUS	System Usability Scale
AS	Application specific Survey
NTNU	Norwegian University of Science and Technology
SVD	Singular Value Decomposition
CRS	Concert Recommendation System
CARS	Context Aware Recommendation System
RMSE	Root Mean Square Error
MRS	Music Recommendation System
RS	Recommendation System
GRS	Group Recommendation System
GD	Gradient Descent
QE	Quality Evaluation
UE	Usability Evaluation
MF	Matrix Factorization
kNN	k-Nearest Neighbor

Chapter 1

Introduction

A Context Aware Group Recommendation System is a Recommendation System that recommends items for groups of people instead of a single person. The group and context part adds additional challenges compared to a normal Recommendation System. A group of people is more dynamic than a single person. You have to consider how the group is formed, how you can combine the group so that unified recommendations for the whole group can be provided, and the dynamics within the group. Context comes in many forms, but can be seen as external constraints that affects the recommendation process. This naturally makes the algorithms more complicated when it has to be considered. The purpose of this thesis is to construct a Context Aware Group Recommendation System for Concerts that takes the location and time of a user into account when making recommendations. This is done to show that traditional methods for Music Recommendation Systems can also be applied when concerts are recommended and extra context have to be considered.

1.1 Motivation

Recommendation Systems are becoming an increasingly important part of large systems such as Amazon.com and eBay.com, and also in the music industry for example Spotify, iTunes and Last.fm. The field is changing rapidly with a lot of interesting research going on. Even though Group Recommendation Systems have been explored, they are not as thoroughly explored as normal Recommendation Systems. The same can be said for Recommendation Systems for Concerts, and Context Aware Group Recommendation systems, where next to no existing research was found.

1.2 Previous work

This thesis is based on the outcome from the authors specialization project [1] in the course "Computer Science, Specialization Project (TDT4501)" at NTNU, where a prototype of a context aware group recommendation system for concerts was developed and empirically evaluated. This prototype utilized a similar collaborative filtering approach as the one described in Chapter 4.6.1 to provide recommendations. It utilized previous listening habits from users that are similar to the user getting recommendations, to predict a rating for all concerts within the given context. This was then done for all of the members of the group, and the results from each individual user was combined into a final set of recommendations. The prototype was evaluated by constructing a set of scenarios and discussing the outcome of them. The discussion concluded with that there was a need for an improvement in the recommendations given by the prototype. In addition to this, a brief review of the state of the art of the recommendation systems literature was performed.

To summarize, the work done in the specialization project consisted of:

- A brief review of the state of the art of the recommendation systems literature, explaining Recommendation Systems, Group Recommendation Systems, and Context Aware Recommendation Systems.
- A prototype of a Context Aware Group Recommendation System for Concerts implementing a $\boldsymbol{k}$-Nearest Neighbor algorithm.
- An empirical evaluation of the prototype that concluded that there was a need to improve the recommendations given by the prototype.

1.3 Contributions

The main contributions from this thesis are:

- A more in depth review of the state of the art based on the authors specialization project [1], explaining: Music Recommendation Systems; the use of Matrix Factorization; challenges with recommendation systems; and an explanation of Recommendation Systems, Group Recommendation Systems and Context Aware Recommendation Systems.
- A prototype of a Context Aware Group Recommendation System for Concerts, including explanations on how the algorithms were designed and implemented. The prototype was based on the prototype implemented in the authors previous work [1]. The $\boldsymbol{k}$-Nearest Neighbor algorithm was changed, a Matrix Factoriazation algorithm and a Hybrid approach of the two were implemented. In addition, the user interface of the prototype was improved to increase its usability.
- An evaluation of the prototype, where its usability and quality of recommendations were evaluated.

1.4 Readers guide

This thesis starts with a state of the art of the recommendation systems literature in Chapter 2. Topics covered here includes Recommendation Systems, Group Recommendation Systems, Music Recommendation Systems, and context awareness. Chapter 3 describes the methodology used for research and evaluation, including Design Science Research and the System Usability Score. It also describes how the prototype is going to be evaluated. Chapter 4 starts by identifying the requirements for the final prototype before formalizing the notion of a Group Recommendation System for Concerts. After this, an algorithm based on the well known k-Nearest Neighbor and Matrix Factorization algorithms that can be used for such problems, is presented. Chapter 5 describes the main architecture of the prototype, and how the algorithms described in Chapter 4 were implemented. In Chapter 6, the prototype with its main functionality is presented. Chapter 7 describes the outcome from the evaluation of the prototype as described in Chapter 3. Chapter 8 summarizes the thesis, including the most important results found in Chapter 7. It also features possible directions of research that can be taken based on the work done in this thesis.

Chapter 2

Background

2.1 Recommendation systems

A recommendation system is, as Ricci et al. [2] says, “software tools and techniques providing suggestions for items to be of use to a user”. These items, to name a few examples, can range from music in services such as Spotify[1], last.fm[2] and iTunes[3]; movies in Netflix[4]; or physical objects at eBay[5] or Amazon[6]. Recommendation systems can be used for a variety of purposes, as Herlocker et al. [3] states by identifying the 11 typical tasks a recommendation system can assist with seen in the list below:

- **Find good items** Finding a ranked list of items for a user and a prediction on how much the user would like them (for example star ratings). This usage area is the focus for this thesis.
- **Find all good items** Finding all the good items, and only the good items for a user. Sometimes it is necessary to find all the good items for a user,

[1]http://www.spotify.com
[2]http://www.last.fm
[3]http://www.apple.com/itunes
[4]http://www.netflix.com
[5]http://www.ebay.com
[6]http://www.amazon.com

not only some of them. Examples of this would be in medical, financial and legal situations, where if present, false positives could potentially have huge consequences.

- **Annotation in context** Finding useful items in the current context of the user. For example providing recommendations such as ”Customers who bought this item also bought this” or ”Similar movies” when looking up a movie.
- **Recommend Sequence** Finding a sequence of items for a user that is ”pleasant as a whole” instead of just good as an individual. An example of such a recommendation system is song playlist generation and the radio function in Spotify.
- **Just Browsing** Assisting users that want to browse a set of items by providing suggestions that is relevant for them. Recommendations like these can be found in services such as Amazon.com and eBay.
- **Find Credible Recommender** Some users tend to not trust recommendation systems. They can play around with the system and see how the system behaves in different settings, for example by changing their profile. Recommendation systems can help asserting their credibility by also including why a given item was recommended in the results, for example ”this song was recommended because you listened to xy.”
- **Improve Profile** By rating items, a user can help improving the outcome of the recommendation algorithm as the algorithm can get a better understanding of what kind of items the user likes.
- **Express Self** Users can also rate items because they like it, they like to express themselves and feel good about it. The main goal isn’t to improve the results of the recommendations, but to express your feelings about an item
- **Help Others** The main goal for rating an item might also be to help others. If you for example have a bad experience with a tour company, posting a

response and rating on a web site such as tripadvisor.com to warn others might be your motivation for using the recommendation system.

- **Influence Others** Recommendation systems can be used to promote a business. Sites such as hotels.com and tripadvisor.com can be used to give ratings to hotels amongst other things. By getting people (for example employees) to extensively give high ratings to your business and give lower ratings to competitors, recommendations for other people can be influenced to promote your business.

Multiple approaches are used to provide these recommendations, some of the more popular ones being Content based filtering, Collaborative filtering, and more recently, Matrix Factorization. Each of the approaches have their strengths and weaknesses, so making hybrid solutions of the approaches can be a viable option to overcome these.

2.1.1 Content based filtering

Content based filtering algorithms are designed around the idea of recommending items similar to the items a user have liked before [4]. The items a user has liked and interacted with before can be seen as the user's profile. Depending on what type of system it is, the user profile can for example be what movies a user have watched, when and what type of music the user has listened to, and what kind of news articles the user has viewed. To find items to recommend, items are compared against this user profile, and recommendations are given based on how well they match the profile.

2.1.2 Collaborative filtering

Instead of recommending items based on their similarity to the items a user has listened to before as with a Content-based filtering approach, a Collaborative Filtering (CF) approach utilizes ”the power of the crowd”. Recommendations are based on what items similar users to the one being recommended for have interacted with before.

Generally, CF approaches are split into two groups, memory based and model based approaches. In memory based solutions, recommendations are created based on the whole dataset of users, ratings and items. In a model based approach, a model of user ratings is created that is used to predict a user's rating to items based on previous ratings [5]. The model can be created using machine learning algorithms such as Bayesian classification and different clustering algorithms.

2.1.2.1 Neighborhood models

In many memory-based CF approaches, a neighborhood of users that previously rated similar items as the user, is created [5]. These neighborhoods can be formed with nearest neighbor algorithms such as a $\boldsymbol{k}$-Nearest Neighbor algorithm using Cosine Similarities (see Section 4.6.1.2) or Pearson Correlation to find similarities between users. The ratings of this neighborhood are then aggregated and compared to provide a list of top-$\boldsymbol{N}$ items for the user.

2.1.3 Matrix Factorization

Matrix Factorization (MF) is based on the idea of latent features; a set of underlying features of the data that can't be explicitly captured, but describes the interactions between the elements of the dataset [6]. The original matrix of user-item-ratings are factorized into matrices of reduced dimensionality describing these latent features. Depending on how much the dimensionality of the matrix is reduced, one could potentially save significant amounts of space and computing power in addition to removing *noise data* from the matrix as Osmanli and Toroslu [7] explains.

These latent feature matrices can then be used to approximate the original matrix. Matrix factorization can also be used as a basis for collaborative filtering or content based filtering, by calculating user or item similarities from the factorized matrices instead of the actual ratings. An example of a MF algorithm is Singular Value Decomposition (SVD).

2.1.3.1 Singular Value Decomposition

SVD is a technique for matrix factorization. It first reduces the dimensionality of a matrix, before using the outcome of this process to approximate the original scores. SVD reduces an $\boldsymbol{mxn}$ matrix into three matrices, $\boldsymbol{U}$, $\boldsymbol{S}$ and $\boldsymbol{V}$ [8].

$$\boldsymbol{R = USV'} \tag{2.1}$$

$\boldsymbol{U}$ and $\boldsymbol{V}$ are two orthogonal matrices of size $\boldsymbol{m \times r}$ and $\boldsymbol{n \times r}$ where $\boldsymbol{r}$ is the rank of the matrix $\boldsymbol{R}$. $\boldsymbol{S}$ is a $\boldsymbol{r \times r}$ matrix and contains all the singular values of $\boldsymbol{R}$ as its diagonal entries. The dimensionality of the matrices can be reduced by choosing $\boldsymbol{S}$ to only contain the $\boldsymbol{k}$ largest singular values of $\boldsymbol{R}$. This makes $\boldsymbol{S_k}$ a $\boldsymbol{k \times k}$ matrix, and since reduction can also be applied to the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices by removing $\boldsymbol{(r-k)}$ columns from $\boldsymbol{U}$ and $\boldsymbol{(r-k)}$ rows from $\boldsymbol{V}$, the dimensionality of $\boldsymbol{USV'}$ can be greatly reduced into $\boldsymbol{U_k S_k V'_k}$, thus saving space and computing power in addition to removing *noise* data from $\boldsymbol{R}$ [7]. The outcome of performing SVD on $\boldsymbol{R}$ is therefore $\boldsymbol{R_k}$, a reduced dimensionality version of $\boldsymbol{R}$.

$$\boldsymbol{R_k = U_k S_k V'_k} \tag{2.2}$$

These matrices can then be used to predict a rating $\boldsymbol{r}$ a user $\boldsymbol{u}$ would give to an item $\boldsymbol{i}$.

2.1.4 Hybrid approaches

Another popular method of providing recommendations is the use of a hybrid approach of different recommendation algorithms. As Adomavicius and Tuzhilin [9] explains, you can either implement the different systems separately and combine the results, incorporate characteristics from one of one of the systems into the others, or create a unifying model that has characteristics from the systems implemented. The use of multiple algorithms can potentially eliminate each algorithms' weaknesses, and thus provide more satisfying results than the individual ones.

2.1.5 Context awareness

The term *context* is used in many situations such as in data mining, information retrieval, and E-commerce personalization [10]. Bazire and Brézillon [11] analyzes 150 definitions of context coming from different domains. Therefore, it is important to have a clear definition of context. In this thesis, context is defined as "Any information that can be used to characterize the situation of an entity, where the entity is a person, place, or object that is considered relevant to the interaction between a user and its application, including the user and the application themselves" [11].

In many cases, additional context other than ratings might affect what a user thinks about an item. For example, in a tourist recommendation system, a user might find hiking less desirable when it is snowing than when sunny, and would want different recommendations depending on the weather. A tourist in London might want recommendations for things to do in London and not in New York. Research exist on how to take such additional relevant contextual information into account when making recommendations for a user [10, 12, 13], however research when it comes to incorporating such contextual information into recommendation algorithms for groups isn't as explored. A model that supports context-aware group recommendations is presented by Stefanidis et al. [14]. Context is here modelled as a set of parameters affecting a user. Each parameter is individually arranged in a hierarchy so that the higher up the hierarchy you are, the more general information you will receive. An example of such a parameter could be location, which could be arranged as City $\prec$ State $\prec$ Country $\prec$ All. A context state is defined as the state of these variables at a given time. A user may define preferences for an item for a given context state. An example on this could be that a user prefers to go to indoor concerts if it is raining, and outdoor concerts if it is sunny. This hierarchical structure would allow for relaxing of context states if not enough recommended items for the context state in question was found. An example could be if there are not enough concerts playing on a given Saturday then recommendations for the whole weekend are given instead. This hierarchical and context state model is then incorporated into a collaborative filtering algorithm using a memory based approach. Individual preferences of each user are aggregated into a common user profile, which then is used to create recommendations. This is done using a least misery (see Section 2.2.2) approach combined with a fair design method.

Wang et al. [15] uses a fuzzy bayesian network to infer if a user is in a given state based on sensor data gathered from a mobile device. The characteristics of a group are modelled as four separate parts: Leader, Expert, Social and Similarity. Leader and Expert describes a single user whereas Social and Similarity describes relationships between users. Expert and similarity are based on the content, Leader and Social are based on the social structure of the group. These characteristics in combination with the context state inferred is used to provide recommendations.

2.1.6 Challenges with Recommendation Systems

Most approaches to recommendation systems comes with well known challenges that should be tackled to ensure that a good result is attained. Some of these challenges are the cold start problem, novelty and serendipity, popularity bias, and implicit/explict data.

2.1.6.1 Cold start

One known challenge, especially when it comes to CF, is the cold-start problem. When a new user joins the recommendation system, no information about his usage habits or ratings are available. Therefore, finding users similar to him based on this is impossible, and thus recommendations for him will be difficult to provide. The same problem comes to light when a new item is put into the system. As no user have rated or used the item yet, it probably won't be picked for recommendation by the algorithm as there is no way to connect it to a user. Solutions to the cold start problem could be to ask a user to rate a subset of the items available, or find items that he likes to establish a starting profile for him. In addition, content based approaches could be utilized to overcome at least the new item problem, for example by utilizing acoustic analysis (see Section 2.3.1) for music recommendations or text analysis (see Section 2.3.2) for movie recommendations, and then finding similar items to the new item and recommending them.

2.1.6.2 Sparse data

In a recommendation system, the user-item-rating matrix is usually sparse, meaning that a user usually rates or interacts with only a very small subset of the available items. This can lead to challenges when it comes to providing satisfyingly accurate results.

2.1.6.3 Novelty and Serendipity

The usual approach to recommendation system research has been to produce the one with the most accurate recommendations based on metrics such as RMSE, not the recommendations that are actually the most useful for a user. McNee et al. [16] goes as far as saying that "not only has this narrow focus been misguided, but has even been detrimental to the field". An example on this would be a movie recommendation system where the 7 last Harry Potter movies are recommended because you watched the first Harry Potter movie. From a statistical perspective, this might provide the most accurate results as it is likely they are the most similar movies, however it might not be a very interesting choice for a user. This is similar to a music recommendation system where only songs that you have listened to before are recommended. While this might provide the most accurate results, it might not be very interesting for a user. Users might not be interested in the items that are most similar to what they have listened to before or the highest rated items, they want recommendations for items that they wouldn't have thought of themselves. Another example on this would be a music recommendation system that frequently produces only widely known artists. McNee et al. [16] describes a system that recommends The Beatle's "White Album" most frequently. As this is a widely known album, it can be said that this is accurate. However, a user might not get very much out of the recommendation, as a user most likely knew about it from beforehand and has either chosen to listened to it or not.

These examples show that there is a need for serendipity and novelty in recommendation systems, finding items that might be seen as surprising and not discovered in other ways, and items that a user hasn't encountered before. Celma [17] puts an emphasis on the importance of focusing on artists on the *long tail* of the listening count curve in a MRS, the lesser known artists.

2.1.6.4 Popularity bias

A problem related to Novelty and Serendipity as described above is *popularity bias* or, as it is more commonly referred to, the *rich gets richer*. This problem often occurs in CF systems [17]. It is based on the fact that popular items have more connections to other items in the dataset, and thus they are more likely to be recommended. This might lead to more interesting items in the *long tail* being left out because of these popular items. This popularity bias will reinforce itself when the popular items are recommended, since the interactions with them will increase and they will gain more connections in the dataset, and become recommended even more often.

2.1.7 Implicit/explicit data

Generally, two types of data are being utilized in recommendation systems: implicit and explicit data. Implicit data is data the system automatically gathers about a user. This can be for example, how many times a user has listened to a song, what books a user has bought, how much time a user has spent viewing an item, or the time of day the user listened to what kind of music, just to give a few examples. Explicit data is data the user explicitly give to the system. This can be for example ratings of an item, a list of preferences on what kinds of items the user likes or demographic information. In the recommendation system literature, there is a heavy emphasis on explicit data and especially ratings to items. This emphasis is mostly due to the Netflix challenge[7] where the submission that would score the best RMSE score on a predefined dataset of movies would win $1M. The Netflix challenge resulted in a boom of research on recommendation systems, especially Collaborative Filtering algorithms. Therefore, as the dataset was based on explicit feedback, research on algorithms that utilize implicit feedback is lacking behind in comparison. However, multiple solutions [18][17][19] are based on normalizing the implicit feedback to behave as explicit feedback, and so, the algorithms developed for explicit feedback systems can also be used for implicit feedback.

[7]http://www.netflixprize.com/

2.1.7.1 Rating Normalization

As Celma [17] states, “it is common that a user’s listening habits distribution is skewed to the right, so it shows a heavy-tailed curve”. A user typically listens to a few artists a lot, whereas the rest have next to no listening counts in comparison. An example of this can be seen in Figure 4.1, where the top 30 artists for a random user and their respective play counts can be seen. In a traditional recommendation system, these statistics tend to be more normalized, and in the case of an explicit feedback recommendation system, in a predefined range. In an implicit feedback recommendation system, there is no such predefined range; ratings (listen counts in this case) can take on any positive value. Since a user can’t state anything about an artist explicitly, there is no such thing as negative feedback in such a system. A rating of 0 just means that a user have listened to the artist 0 times, it doesn’t necessarily mean that the user doesn’t like the artist. To tackle these challenges, there is a need to normalize the implicit feedback if traditional explicit recommendation algorithms are to be applied.

2.2 Group Recommendation Systems

A Group Recommendation System (GRS) is a RS that provides recommendations to a group of people instead of a single individual. There are two main approaches of accomplishing this: calculating recommendations individually for each of the members of the group, and then aggregating the individual results; or merging the preferences of each of the members of the group, and then providing one set of recommendations based on the merged profile. In either of the approaches, there are many ways this merging can be accomplished [20]. This includes least misery, average, and average without misery. The choice of aggregation strategies should be decided based on the problem you are trying to solve, as there is no universal best strategy that works in all cases.

2.2.1 Average Aggregation

Average aggregation uses the average of the individual ratings for an item as a final rating for that item. With this approach, nothing extra such as misery or the dynamics between individual persons are taken into account, only the pure ratings of the users. In some cases this might be to prefer, in other cases not.

2.2.2 Least Misery Aggregation

A least misery approach uses the minimum of the individual ratings of an item as the group's rating. This method basically says that a group is as happy as its least happy member [20]. This might be a good strategy in some cases. For example, if a member of a small group has an allergy to seafood, recommending a seafood restaurant to that group might not be a good choice, even though the rest of the group really want to go. However, this also means that one person with a negative preference can decide the outcome of the recommendation even though all the other members have a really positive preference to the item.

2.2.3 Average Without Misery Aggregation

With an average without misery approach, items with a rating below a threshold for one of the users are excluded and the average of the rest of the items is used as the score for the item. This approach is similar to the least misery approach in that if a person has a really strong negative preference for an item, that item will be excluded given that its rating is below the threshold. However this approach might yield results that better reflect the wishes of all the members of the group given that a proper threshold is chosen.

2.3 Music Recommendation Systems

Recommendation Systems for Music (MRS) have increasingly become an important part of music services. Services such as iTunes, Spotify, last.fm and Pandora all incorporate music recommendations centrally in their user interface. With an ever growing collection of music, these services compete in finding new and innovative ways on how users can discover these wast amounts of data.

Celma [17] identifies three use cases typical for a MRS: *neighbor finding*, *playlist generation* and *artist recommendation*. *Neighbor finding* consists of finding users with a similar taste in music as you. *Playlist generation* usually means finding songs to recommend for a user, but instead of just returning the top N songs, songs that go well together are preferred. *Artist recommendation* usually consists of finding artists based on a user's profile, be it the artist with the highest predicted rating or novel artists.

Different services apply a variety of techniques when it comes to the recommendation process [21]. Some of them are acoustic analysis, text analysis, editorial review, and the use of activity data.

2.3.1 Acoustic Analysis

Music can be recommended based on their acoustic characteristics such as rhythm and beat. This can be used to find music that sound alike by comparing these characteristics [22], or used to provide recommendations in different settings (for example high beat music when a user is working out, and low beat when a user wants to sleep). Acoustic Analysis have also been used to identify music characteristics and map them to mood [23][24]. Feng et al. [25] analyze the tempo and articulation of music and classifies it into four categories of mood: happiness, sadness, anger and fear, and lets users browse the categories in a 3D visualization.

2.3.2 Text Analysis

Music can be analyzed and recommended based on text related to the music. Echo Nest[8] analyzes millions of music related webpages. They look for mentions of artists in the pages analyzed, and parse the language used around them [21]. The parsed text is then used to find terms describing the artists such as noun phrases that is related to the artist. The most used terms found, can then be used to for example find similar artists. This data is also combined with structured data such as data from musicbrainz[9] and Wikipedia[10] to further improve the results given.

2.3.3 Editorial review

The popular internet radio Pandora[11] employs professionals to listen and categorize music by their *genome*; 400 attributes in categories such as melody, rhythm and lyrics. These *music genomes* are then used to provide recommendations by finding music with similar properties as the given artist.

2.3.4 Activity data

In an approach where activity data is used, the recommendation system utilizes data it knows about the user. For example, this can be ratings the user have given to artists, how many times a user have listened to different artists, and likes on Facebook. This activity data form the basis for recommendations when collaborative filtering algorithms such as the k-nearest neighbor and Matrix Factorization algorithms (see Section 2.1.2) are used. The use of activity data forms the basis of the approach taken in this thesis. Examples of the use of activity data can be seen in [19]; [26] and [17]; where listening records from services such as Last.fm, Yahoo! Music and Douban are used as a basis for recommendation.

[8]http://echonest.com/

[9]http://musicbrainz.org/

[10]https://www.wikipedia.org/

[11]http://www.pandora.com

2.3.5 Types of users

A challenge for a MRS is the variety of users it has to consider when making recommendations. Emap International Limited[12] performed a study to map the music fan economy as explained by Jennings [27]. They categorized listeners into four groups: savants, enthusiasts, casuals and indifferents. Savants represented 7% of music listeners, and they were characterized by their huge interest in music, "everything in life seems to be tied up with music". Enthusiast represented 21% of the listeners, "music is a key part of life but is balanced by other interests". Casuals represented 32% of the listeners, "Music plays a welcome role, but other things are far more important". Indifferents represented 40% of the listeners, "Would not lose much sleep if music ceased to exist".

This study shows that people have different ways to think about music. Enthusiasts and savants might prefer to try out new and little known artists, whereas casuals might prefer well known artists and the latest 'big hits'. With such a diverse user base, creating a music recommendation system that works well for all of them is challenging.

[12]http://www.emap.com/

Chapter 3

Methodology

3.1 Design Science Research

This project was undertaken using an approach based on the Design Science Research methodology. Design Science "seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished."[28]. Hevner et al. [29] provides a framework with a set of seven guidelines on how to conduct design science research (see Table 3.1). Design science evolves around creating an artifact and rigorously evaluating it. Four types of artifacts are also described in [29]: *constructs*, *models*, *methods* and *instantiations*. Constructs are defined as concepts or vocabulary used in IT related domains (e.g. entities, objects or data flows). Models can be seen as a combination of constructs that are used to aid problem understanding and solution development. Methods are guidance on the models to be produced (e.g. formal mathematical algorithms, methodologies). Instantiations are working systems that demonstrate that constructs, models and methods can be implemented in a computer-based system.

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

TABLE 3.1: Design-science research guidelines [29]

3.1.1 Application to this project

3.1.1.1 Guideline 1: Design as an Artifact

In this thesis an artifact will be created and evaluated. An instantiation of a web-based group recommendation system for concerts will be created and the algorithms used will be outlined.

3.1.1.2 Guideline 2: Problem Relevance

A group of people wanting to attend a concert together might have trouble finding a concert they all would be happy with. A easy to use group recommendation system could help ease this problem by providing recommendations for concerts based on what music each of them likes. Music Recommendation Systems are more and more becoming a vital part of popular applications such as Spotify and iTunes. They assist users in music discovery, and thus provide them with a more personalized user experience.

3.1.1.3 Guideline 3: Design Evaluation

The prototype will be evaluated rigorously by first performing a usability evaluation using the System Usability Scale (explained in Section 3.3.1.1). Then follows another period of development, before the quality of recommendations will be evaluated in a user centred way through a scenario based evaluation method (explained in Section 3.3.2).

3.1.1.4 Guideline 4: Research Contributions

The algorithms utilized for the recommendation process are explained in detail, both how they were designed and implemented. In addition to this, all source code used is made available so others can test and verify the results.

3.1.1.5 Guideline 5: Research Rigor

This thesis explains both the implementation and design of the recommendation algorithms implemented to make sure the outcome of the thesis can be reproduced. Reproduction of the results is made easier by making the code-base open source. The prototype is also evaluated using the System Usability Scale, and a Quality Evaluation.

3.1.1.6 Guideline 6: Design as a Search Process

This project is based on what was produced in the author's fall semester project [1]. A prototype of a concert recommendation system was developed based on the k-Nearest Neighbor algorithm and empirically evaluated. The user interface of the prototype was then improved based on the feedback from the evaluation. This development period was followed by a user evaluation using the System Usability Scale. Based on the outcome of this evaluation, changes were made to the user interface, and the recommendation algorithm was improved. Finally an evaluation of the finished prototype was undertaken. These steps show that multiple iterations were performed to search for better solutions to the problem in questions.

3.1.1.7 Guideline 7: Communication of Research

This thesis contains both technical descriptions of the implementation of the prototype (see Chapter 4 and Chapter 5), and a more general description and presentation of its main functionality in Chapter 6. This thesis will be publicly available through NTNU. All source code will be publicly available at Github with a GNU General Public License (GPLv2[1]) license for others to test, use and extend (see Appendix B).

3.2 Evaluation tools

Vital to the Matrix Factorization algorithm described in Section 4.6.2, is the choice of constants to be used in the process. Choosing the optimal constants will significantly increase the accuracy of the algorithm. Two methods were used for this: Root Mean Square Error and n-Fold Cross Validation.

[1] http://www.gnu.org/licenses/gpl-2.0.html

3.2.1 Root Mean Square Error

Root Mean Square Error (RMSE) is the root of the average squared error between two sets as seen in Equation 3.1, where x is the original value and $\widehat{x_i}$ is the predicted value. RMSE is commonly used to find the average error between original ratings and predicted ratings in recommendation systems, and can be seen as a measurement on how accurate a recommendation system predicts its ratings.

$$RMSE = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (x_i - \widehat{x_i})^2} \tag{3.1}$$

Often, RMSE is used in combination with n-Fold Cross Validation to find the most optimal parameters for such algorithms.

3.2.2 n-Fold Cross Validation

n-Fold Cross Validation is usually used to tackle the well known problem of overfitting in statistical evaluation of recommendation systems and other estimation systems. Let's say that the performance of a recommendation system is to be analysed by calculating the RMSE between the known ratings in its dataset and ratings it predicts. If the system is trained with all the data (or a too large portion) of the ratings known, it is not unreasonable that the RMSE could be close to perfect; the system would just return the already known ratings, not actually make any predictions. In n-fold cross validation, this problem is overcome by splitting the set of items into n parts (folds). In turn, every of the n folds is used a validation set, whereas the other $n-1$ folds are used as a training set. The model is trained with the training set, and the validation set is used to validate the model. The final error estimate is then the average of the error estimate from the individual folds. By choosing an appropriate n (10 is frequently used), the model is evaluated with actual predictions and not just predict ratings it already knows.

This technique is used in this thesis to tune the Matrix Factorization algorithm (see Section 5.4.3.1), by running it for different input parameters and choosing the ones that yielded the lowest RMSE.

3.3 Evaluation plan

The prototype will be evaluated in two steps. After the first period of development, the usability of the prototype will be evaluated using the System Usability Scale (SUS). After the second development period, the quality of the recommendations will be evaluated using a Quality Evaluation survey.

3.3.1 Usability Evaluation

After the first period of development, 15 users will be asked to play around with the prototype and answer three questionnaires, the System Usability Scale (SUS), an Application Specific survey (AS) and a questionnaire to gather Background Information (BI). The System Usability Scale is used to evaluate the usability of the prototype as explained in Section 3.3.1.1. The Background Information (BI) schema is used to find out background information about the users, such as their demographics and previous experience with recommendation systems. The BI schema used can be seen in Appendix 7.2. The Application Specific Survey (AS) asks more application specific about the prototype's usability than the SUS survey. The AS schema used can be seen in Appendix A.2.

The participants will be asked to undertake the following steps:

1. Answer the Background Information schema (see Appendix 7.2).
2. Create a user
3. Find recommendations for both a group of users and only the user that was created (no information about dates or location was specified here)

4. Adjust the results to include nearby location and dates (relax context)

5. Answer the System Usability Scale schema (see Appendix A.1).

6. Answer the Application Specific Survey (see Appendix A.2).

3.3.1.1 System Usability Scale

Brooke [30] states that the System Usability Scale is a "reliable, low-cost usability scale that can be used for global assessments of systems usability". It gives a global view of subjective assessments of usability. It contains ten statements where the participants indicate how much they agree or disagree with the statement on a 5 point scale. These statements can be seen in Appendix A.1. The individual scores to these questions forms the basis for a final score, the SUS score, that can help determine the usability of an application.

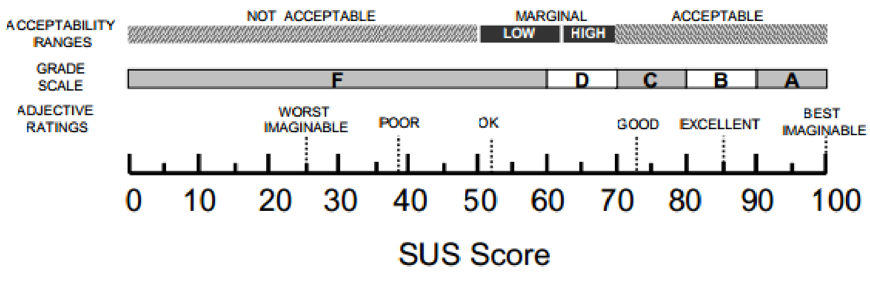


FIGURE 3.1: SUS adjective rating scale (Bangor et al. [31])

The SUS score is as a single number that is "representing a composite measure of the overall usability of the system being studied". Question 1, 3, 5 and 7 contributes to the final score with the score position minus 1. Question 2, 4, 6 and 8 contributes with 5 minus the score position. To get the final SUS score, this sum is multiplied with 2.5. The final SUS score will be a number between 0 and 100.

Bangor et al. [31] proposes an adjective rating scale (see Figure 3.1) to help determine what SUS scores actually mean. They show that this Likert scale is highly correlated with SUS scores ($\alpha < 0.01$ with $r = 0.822$). Bangor et al. [32] adds to this by stating that "products with scores above 70 are at least passable, better products have a score in the high 70s to upper 80s, and truly superior products score better than 90. Products with a score below 70 should be considered candidates for increased scrutiny and continued improvement".

3.3.2 Quality Evaluation

To evaluate the quality of the recommendations given by the prototype, two groups of students will be asked to participate. First, for both the groups, each of the members of the group will be asked to:

1. Create a user and rate 5 of your favorite artists
2. Find recommendations for the user created between 18/02/2014 and 03/03/2014 in London
3. Find recommendations for the user created between 05/03/2014 and 09/07/2014 in New York
4. Create a new user and rate 10 of your favorite artists.
5. Find recommendations for the new user between 18/02/2014 and 03/03/2014 in London
6. Find recommendations for the new user between 05/03/2014 and 09/07/2014 in New York
7. In a group of 2, find recommendations between 18/02/2014 and 03/03/2014 in London
8. In a group of 2, find recommendations between 05/03/2014 and 09/07/2014 in New York
9. In a group of 3, find recommendations between 18/02/2014 and 03/03/2014 in London
10. In a group of 3, find recommendations between 05/03/2014 and 09/07/2014 in New York

After every task, the users will be presented with three sets of recommendations (cases), one for each of the three algorithms implemented. The users will then be asked to rate each of the cases on a scale from 1 to 5 on how satisfying the recommendations given were, where 1 is *Very Satisfied* and 5 is *Very Dissatisfied*. The user will also be asked to choose which of the cases looked most appealing. The schemas, including the questions asked, used for this evaluation can be seen in Appendix C.2.

Chapter 4

Design of Recommendation Algorithm

4.1 Requirements

In the previous work on this project (described in Section 1.2), the following scenario that describes how a group recommendation system for concerts could be used, was presented:

> *Imagine that a group of friends are attending a one-day music festival with a lot of different bands performing. Their tastes in music are quite different, so choosing what concerts to attend is a challenge. Also, they are not familiar with a lot of the bands playing. They would love an application that would give them recommendations for which concerts to attend based on what sort of music they have listened to before, and their personal musical preferences. One of the group members heard that other bands were playing nearby but not as part of the festival, so they would also like to get recommendations for nearby concerts that are not part of the festival. Additionally, the group is in town for a week, so they would like to get recommendations for both the day of the festival and for the remainder of the trip.*

ID	Requirement
R1	Concert recommendations should be based on a user's listening habits.
R2	The prototype should take the location of a concert into account (concerts close to a user are preferred).
R3	The prototype should be able to provide recommendations for more widespread locations if not enough concerts are found for the given location.
R4	The prototype should take time into account; it should not recommend concerts that already have taken place or concerts too far ahead in time. It should recommend concerts within the users provided time constraints if such is provided.
R5	If not enough concerts are found within the given time frame, the time constraint should be relaxed.

Table 4.1: Prototype requirements

From this scenario, five requirements for the final prototype were extracted as seen in Table 4.1. The same requirements will also be used in this thesis. As suggested by the previous work done on this prototype, there is a need to improve the outcome of the recommendation algorithms implemented. Therefore, ways to improve the recommendation algorithms will be investigated, the prototype will implement the proposed algorithms, the usability of the prototype will be improved, and the prototype will be thoroughly evaluated. The project will be following the Design Science Research paradigm.

4.2 Design approach

For the purpose of the design of the recommendation algorithm, a CRS is defined as seen in Definition 4.1.

Definition 4.1. In a Recommendation System for Concerts (CRS), a set of concerts, $\boldsymbol{C}$; artists, $\boldsymbol{A}$; and users, $\boldsymbol{U}$; are used to provide concert recommendations for a user $\boldsymbol{u}$. Each of the concerts have a set of artists performing. Each of the users in $\boldsymbol{U}$ have given a rating to a subset of the artists. A group, $\boldsymbol{G \subseteq U}$, consists of one or more users. A set of context parameters $\boldsymbol{CP(location, time)}$ is defined, which the concerts recommended should adhere to. The goal of the recommendation system is to provide the top $\boldsymbol{N}$ most relevant concerts $\boldsymbol{C'} \subseteq \boldsymbol{C}$ for $\boldsymbol{G}$ as a whole that adhere to the context applied, $\boldsymbol{CP}$, and the ratings, $\boldsymbol{R}$, each of the members of $\boldsymbol{G}$ previously have given.

In this project, three main algorithms were implemented and used for the recommendation process. A $\boldsymbol{k}$-Nearest Neighbor algorithm was implemented, finding the $\boldsymbol{k}$ most similar users to the user finding recommendations, then using the artists these users have listened to and using them for recommendation. A Matrix Factorization algorithm utilizing a Gradient Descent (GD) method was implemented, reducing the $\boldsymbol{NxM}$ artist listening matrix into two matrices $\boldsymbol{Nxf}$ and $\boldsymbol{Mxf}$ which then can be used to predict ratings of the original matrix, including for records not present in the original matrix. The third algorithm implemented is a hybrid approach between these two, combining the results given by both of them.

The recommendation part of this project consists of five main phases:

1. Context definition
2. Listen count normalization
3. Filter dataset
4. Calculate individual recommendations
5. Make Hybrid recommendations.

In phase one, *context definition*, the context to be applied to the recommendation process is defined. The context in this project is defined as the location and time of a concert, in addition to the users requesting the recommendations. In phase two, the number of times a user has listened to each of the artists are normalized for the algorithms to work optimally. In phase three, the dataset is filtered, so that only items that have an impact on the result will be considered. In the *calculate individual recommendations* phase, the top $\boldsymbol{N}$ concerts for each of the members of the group defined in phase one is found, using the $\boldsymbol{k}$-Nearest Neighbor and MF algorithms. Phase five consists of making Hybrid recommendations by aggregating the results given from the previous phase into a single list of recommended concerts for all of the members of the group.

4.3 Phase 1: Context definition

The context parameters $\boldsymbol{CP}$ are defined as $\boldsymbol{(location,time)}$. In a CRS, the location and time plays an important role in the recommendation process. A user would probably want to get recommendations for concerts in locations close to where he is located, and not get recommendations for concerts too far ahead in time, or for concerts that already have taken place. Therefore, the definition of these context parameters should be a central part of any CRS.

As the algorithms applied in this project are based on algorithms used for recommendation systems with explicit feedback, there is a need to normalize the listening counts to a predefined scale so that the algorithms can work optimally.

4.4 Phase 2: Listen Count Normalization

The approach taken to listening count normalization in this thesis, is the one proposed by Celma [17]. For each user, $\boldsymbol{u}$, its listening counts for each artist, $\boldsymbol{a}$, is normalized using the Cumulative Distribution Function (CDF) of the artist plays for $\boldsymbol{u}$. The play counts for the user *simensma* can be seen in Figure 4.1, whereas the associated CDF can be seen in Figure 4.2. The CDF for a variable $\boldsymbol{x}$, $\boldsymbol{F(x)}$, is the proportion of population with a value less than $\boldsymbol{x}$.

As explained in Section 2.1.7.1, the listening habits for a user tends to be skewed. A user tends to listen to a few artists a lot. It can be said that the listening curve for the user tends to be heavy-tailed. To check if the listening count curve for a user has heavy-tailed properties, the coefficient of variation, $\boldsymbol{CV}$, is calculated. $\boldsymbol{CV}$ is the standard deviation of the user's listening counts divided by its mean.

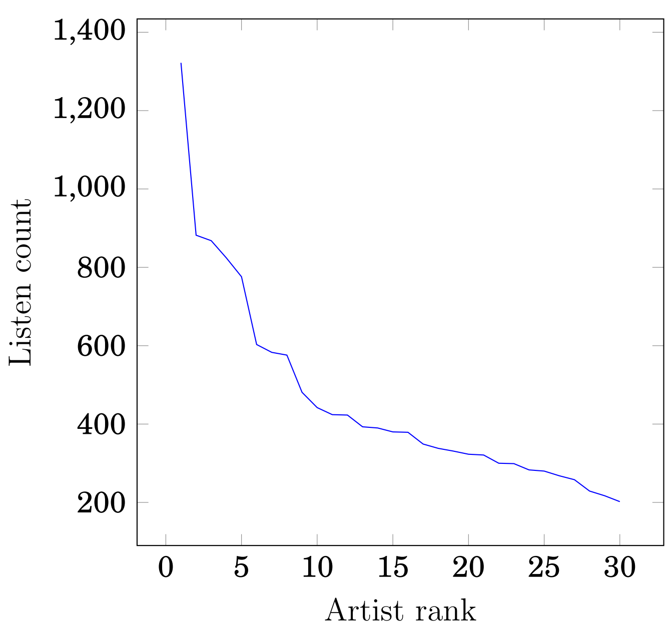


Figure 4.1: Listen count distribution for a user

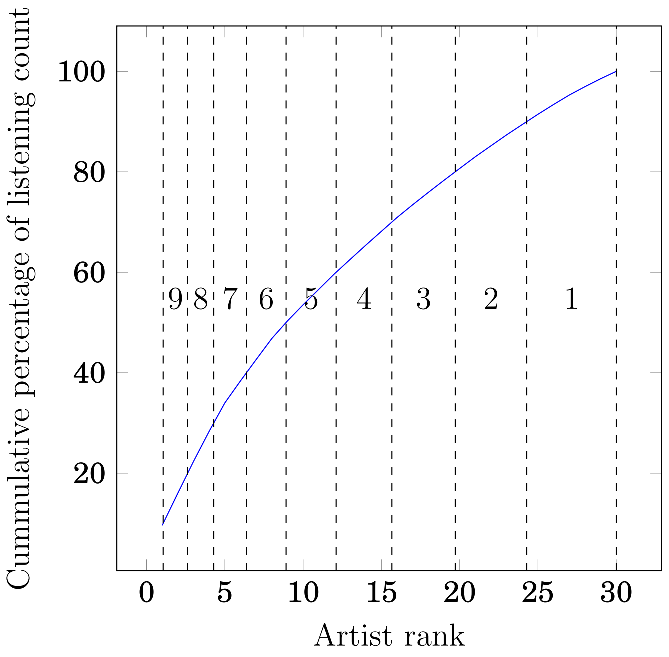


Figure 4.2: Cumulative Distribution Function with rating partitions for the listen counts for a user

$$CV = \frac{\sigma}{\mu} \tag{4.1}$$

If $CV > 0.5$, then the CDF, $F(x)$, of the user listening counts for u is calculated. The artists in the top 90% of this distribution will be assigned a rating of 10, the top 80% a rating of 9, the top 70% a rating of 8, and so on until the artists at the bottom 10% of the distribution will get a rating of 1.

If $CV \leq 0.5$, then all artists will get a normalized rating of 5 as this is a signal of that the listen count curve is not heavy-tailed.

To model the CDF for a user, $F(x)$, the formula proposed by Kilkki [33] as seen in Equation 4.2 is used. N_{50} is the number of objects that cover half of the whole volume, α is the factor that defines the form of the function, β is the total volume of the function, and x is the listening count to be normalized.

$$F(x) = \frac{\beta}{\left(\frac{N_{50}}{x}\right)^{\alpha} + 1} \tag{4.2}$$

4.5 Phase 3: Filter Dataset

In the filter dataset phase, the dataset is filtered so that only records adhering to CP remains for processing by the rest of the algorithm.

Definition 4.2. The function $filter(Con \subseteq C, CP)$ returns the concerts $C' \subseteq C$ that adhere to each of the context parameters CP given.

Definition 4.3. The function $listenedTo(u' \in U, a \in A)$ returns whether or not the user u' has listened to the artist a.

The algorithm first reduces the data set to only contain the concerts in C that adhere to $CP.location$ and $CP.time$.

$$C' \subseteq C = filter(C, CP) \tag{4.3}$$

The context defined in the previous phase is applied to the dataset, filtering the dataset to only contain records that will have an impact on the final outcome of the recommendation process. In a MRS, normally all the available artists have to be considered to find the ones that are best fitting. The approach taken, suggests that only the artists playing at the concerts given in C' have to be considered, as the ones not part of it would not have an effect on the final result. The rating procedure for an artist is not directly dependent on the rating procedure for another artist, and therefore only the artists and concerts fitting with CF will have an impact. This gives the following reduced set of artists A' to consider for the individual recommendation phase.

$$A' = \{a : \forall c \in C' \forall a \in c\} \tag{4.4}$$

4.6 Phase 4: Calculate Individual Recommendations

When the listen counts for the users have been normalized and the dataset has been filtered, two algorithms are applied to the dataset to predict ratings for each of the relevant artists: a k-Nearest Neighbor collaborative filtering algorithm, and a Matrix-Factorization algorithm using a Gradient Descent method. The normalized listened counts from phase 2 will in this section be referred to as ratings, as they now have been normalized to the range 1-10.

4.6.1 k-Nearest Neighbor

The k-Nearest Neighbor algorithm is split into two phases:

1. Filter dataset
2. Recommend concerts

4.6.1.1 Filter Dataset

In a kNN approach, the k Nearest Neighbors of u is found and used as a basis for recommendation. For simplicity reasons, we state that a user that hasn't listened to any of the artists in A' cannot be considered by the algorithm. This can be done because a user that hasn't listened to any of the artists in A', could only contribute with a listening count of 0 to all of them, and therefore the user might as well be left out (see Section 4.6.1.2).

Since the set of artists that is considered in the algorithm has been reduced to the set of artists A' playing at one of the concerts in C', implicitly, the set of users considered for the algorithm can be reduced to the set of users that have listened to one or more of the artists in A'.

$$U' = \{u' : \forall u' \in U \exists a \in A' listenedTo(u', a)\} \tag{4.5}$$

4.6.1.2 Recommend Concerts

In a kNN algorithm, the k most similar users to u are found, and their ratings are used as a basis for recommendation. To find these similar users, a measurement of similarity between two users has to be designed.

The similarity measure chosen in this project, is the cosine similarity between two user's listening count for each artist. The cosine similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as:

$$sim(\mathbf{x},\mathbf{y}) = \cos\theta = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\| \times \|\mathbf{y}\|} \tag{4.6}$$

Definition 4.4. The function $listenCount(u' \in U, a \in A)$ returns the amount of times u' have listened to a.

The user vector $\mathbf{w}_i$ for a user $u_i \in U$ is defined as the vector of the users listening counts to each of the artists in A.

$$\mathbf{w}_i = \{listenCount(i,a) : a \in A\} \tag{4.7}$$

For all unique pairs of users, the cosine similarity between their respective listening vectors is calculated.

The similarity between two users, $sim(u_1,u_2)$ then becomes

$$sim(u_1,u_2) = sim(\mathbf{w}_1,\mathbf{w}_2) = \frac{\mathbf{w}_1\cdot\mathbf{w}_2}{\|\mathbf{w}_1\| \times \|\mathbf{w}_2\|} = \frac{\sum_{i=1}^{n} \mathbf{w}_{1i} \times \mathbf{w}_{2i}}{\sqrt{\sum_{i=1}^{n} (\mathbf{w}_{1i})^2} \times \sqrt{\sum_{i=1}^{n} (\mathbf{w}_{2i})^2}} \tag{4.8}$$

In a normal k-Nearest Neighbor algorithm, the k users with the highest similarity would now be identified and used as a basis for recommendation. For the purpose of a CRS, this isn't enough. Here, a rating for each of the concerts, c_i, in C' has to be predicted. Therefore, a 3 step process is undertaken for each of the concerts:

1. Find the K users, U'', with the highest similarity to u from the subset of U' that have listened to one or more of the artists performing at that concert.

2. Calculate the predicted rating for each of the artists a playing at the concert. $totalSimilarity$ is defined as the sum of similarities to u from each of the users in U''. Each of the users, u_i in U'', will contribute to the predicted rating with a percentage of $\frac{sim(u_i,u)}{totalSimilarity}$. The actual contribution is influenced by the rating given to a by u_i, so this is multiplied with $rating(u_i,a)$. The predicted rating for an artist i will then be:

$$artistRating_i = \sum_{j=1}^{n} \frac{sim(u_j,u) \times listenCount(u_j,a_i)}{totalSimilarity} \tag{4.9}$$

3. The overall predicted rating for the concert c_i as a whole for user u is given by the average of the predicted ratings to each of the m artists performing at c_i.

$$knnRating^u_{c_i} = \frac{\sum_{k=1}^{m} artistRating_k}{m} \tag{4.10}$$

4.6.1.3 Choice of Aggregation Strategy

An *average* aggregation strategy is used to aggregate individual ratings into a combined rating for a concert in step 3 of the algorithm in the previous section. In a MRS utilizing implicit feedback, there is no such thing as negative preferences. A listen count of 0 doesn't necessarily mean that a user doesn't like the artist, just that the user hasn't listened to the artist before. The user might love the artist but he hasn't discovered it, or he might hate it; it's impossible to know for certain. Similarly, a low listening count might not mean that a user doesn't like the artist, he might just recently have discovered the artist, or just recently joined the system. Again, it is impossible to know. Based on this, the use of misery in an aggregation method would here not make sense, as it is designed to avoid misery; which in this case doesn't exist. Therefore, an *average* strategy was utilized instead.

4.6.2 Matrix Factorization

The approach to Matrix Factorization (MF) taken in this project is similar to what Koren et al. [6] describes in their paper, and the algorithm outlined by Funk [34]. The algorithm was made popular by Simon Funk as a submission for the Netflix challenge[1], that was a competition to produce the most accurate predictions for movie ratings in the Netflix movie dataset.

This approach is closely related to a SVD approach such as the traditional approach described in Chapter 2.1.3.1. For a traditional SVD approach, missing values in the user-item matrix either have to be guessed first or the rows and columns left out, as the SVD requires *complete* data [35]. This makes such an approach infeasible, as the user-item matrix for the purpose of this thesis is very sparse. The approach taken in this paper takes this into account, and is able to make reasonable predictions in cases where ratings are missing.

As Koren et al. [6] explains: "Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modelled as inner products in that space." A latent factor is a variable that is not observed externally but inferred from the external observed variables of a data set. Latent factor models tries to extract these latent factors and use them to produce recommendations instead of the actual values of the user-item matrices. These models can then be used to predict a rating for a user-item pair, even if the rating was not present in the matrix before. In this thesis, the items are the artists of the dataset.

Similarly to what Koren et al. [6] does, the $n \times m$ user-artist matrix M is reduced into a set of user vectors, V, where $V_i \in \mathbb{R}^f$ and artist vectors, B, where $B_i \in \mathbb{R}^f$. f is the number of latent factors to extract (dimensionality of the latent factor space).

[1]http://www.netflixprize.com/rules

In this thesis, the user-artist matrix consists of the normalized listen counts for all of the users in U and the artists in A.

To approximate a user u's rating for an artist a, r_{ua}, the dot product between u's and a's latent factor vectors, $V_u B_a$, is performed. As Koren et al. [6] says: this dot product "captures the interaction between user u and item i - the users' s overall interest in the item's characteristics".

$$r_{ua} = B_a^T V_u \tag{4.11}$$

B and V are found by minimizing the regularized squared error of the existing ratings in the dataset (R) as seen in Equation 4.12. These ratings make up the training set T; the pairs of artists a and users u, (u,a), where $r_{ua} \in R$.

$$\min_{B^*,V^*} = \sum_{(u,a)\in T} (R_{ua} - B_a^T V_u)^2 + \lambda(\|B_a\|^2 + \|V_u\|^2) \tag{4.12}$$

The minimization of Equation 4.12 is done by using the gradient descent method proposed by Funk [34]. For each of the user-artist pairs (u,a) in the training set T of ratings, the prediction error e_{ua} is calculated. e_{ua} compares the existing rating R_{ua} with the predicted rating r_{ua}. If R_{ua} doesn't exist, R_{ua} is set to be the average of all the ratings a has received.

$$e_{ua} = R_{ua} - r_{ua} \tag{4.13}$$

The associated features B_a and V_u are then incremented as follows:

$$B_a = B_a + \gamma \cdot (e_{ua} \cdot V_u - \lambda \cdot B_a) \tag{4.14}$$

$$V_u = V_u + \gamma \cdot (e_{ua} \cdot B_a - \lambda \cdot V_u) \tag{4.15}$$

γ and λ are constants that should be determined based on the dataset by for example cross validation.

γ is by Funk [34] referred to as the *learning rate*, an arbitrary number that can be set to tune how fast and thus how precise the outcome should be. λ is referred to as K, an arbitrary number that is used for regularization to reduce overfitting of the results.

The calculation of e_{ua},B_a and V_u, is repeated for each feature to be extracted, until the total Root Mean Square Error (RMSE) of e converges. The RMSE converges when the improvement in RMSE compared to the previous iteration is less than a set threshold; *minimum improvement.*

Having calculated V and B, the predicted ratings of all of the artists in A' are calculated for u.

Finally, a rating for each of the concerts c_i in C' is calculated.

The overall predicted rating for the concert c_i as a whole for user u is given by the average of the predicted ratings to each of the m artists performing at c_i.

$$mfRating^u_{c_i} = \frac{\sum\limits_{k=1}^{m} r_{uk}}{m} \tag{4.16}$$

4.6.2.1 Other applications

The latent factors can also be used to find the similarity between two users $u1$ and $u2$, by calculating the cosine similarity between the two users latent factor vectors $sim(V_{u1}, V_{u2})$:

$$sim(u1,u2) = sim(V_{u1}, V_{u2}) \tag{4.17}$$

They can also be used to find the similarity between two artists $a1$ and $a2$, by calculating the cosine similarity between the two artists latent factor vectors $sim(B_{a1}, B_{a2})$:

$$sim(a1,a2) = sim(B_{a1},B_{a2}) \tag{4.18}$$

4.7 Phase 5: Make Hybrid recommendations

The predictions given by the algorithms in the previous two sections are in this phase aggregated to produce the final top N concerts to return to the user. For each of the concerts, c_i, in C the final rating for the concert for u, $r_{uc_i} \in R_u$ is given by:

$$r_{u_ic} = \frac{mfRating^u_{c_i} + knnRating^u_{c_i}}{2} \tag{4.19}$$

The N concerts with the highest rating r_{uc_i} in R are selected and returned to the user.

Chapter 5

Architecture and Implementation

5.1 Development environment

A wide range of tools were used in the development process of the prototype. Development in Java was done in the Eclipse[1] IDE. Dependency management and the build process of the Java project was handled with Apache Maven[2]. The front end part of the project was developed using Visual Studio 2013[3]. The build process for this part was performed using *grunt*, a JavaScript task runner that performs tasks such as compilation, unit testing, linting, and minification [36]. By using *grunt* and *Maven*, repetitive tasks that otherwise would have been done manually, was automatically performed.

For evaluation purposes, the project was deployed to a 12GHz/2560MB cloud server running Debian 7[4], with the *Node.js*[5] http-server[6].

[1]http://www.eclipse.org/
[2]http://maven.apache.org/
[3]http://www.visualstudio.com/
[4]https://www.debian.org/
[5]http://nodejs.org/
[6]https://www.npmjs.org/package/http-server

5.2 Architecture

The application is divided into a Front End and a Back End part. The back end is a Java application exposing a JSON REST interface. This was done to ensure maximum flexibility when it comes to choice of front end technologies. The front end is a pure HTML and JavaScript application utilizing AJAX to communicate with the back end.

5.2.1 Front End

The front end of the prototype was developed using HTML5, JavaScript/JQuery and CSS3. It is based on a Model View ViewModel design pattern and is built using the Durandal.js JavaScript framework.

5.2.1.1 Model View ViewModel

Model View ViewModel (MVVM) is a design pattern that splits the source code into Models, Views and ViewModels [37] as seen in Image 5.1. A Model contains the data and business logic of the application, whereas the View is responsible for the GUI part of an application; all the visual elements and capturing of keystrokes. The Model is completely separate from the View, and thus, either of them can easily be switched out if needed.

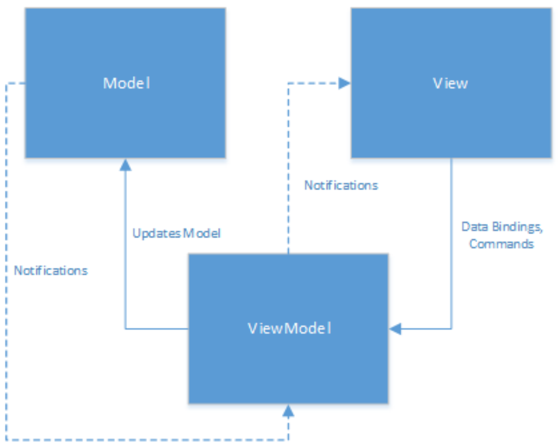


FIGURE 5.1: The Model View ViewModel pattern (based on Gossman [37])

The ViewModel is the glue that connects the model and the view. In simple applications, the controls of the view can simply display what the model contains, the model is directly bound to the view. However in most applications, more logic is involved. The ViewModel can be looked on as the middle man that converts data from the model into data that can be displayed by the view. It can be said that it ”exposes data needed by a view (from a model) and can be viewed as the source the views go to for both data and actions” [38]. To interact with the Model, the View utilizes commands the ViewModel provides [37]. This mapping from a View control to a ViewModel command, is called a data-binding. The main interactions between these components can be seen in Image 5.1. The View communicates with the ViewModel through data-bindings and commands, and the ViewModel updates the Model based on these. When the model changes, the ViewModel is notified and reacts to these changes. When the ViewModel changes, the changes are propagated to the View.

5.2.1.2 Durandal.js

```
define(['artist','knockout'], function(artist,ko) {
    ...
});
```

Listing 1: Simple Durandal module

Durandal.js is a framework for building Single Page Applications in JavaScript. It is built on JavaScript frameworks such as jQuery, Knockout.js and RequireJS[39].

Durandal is module based, which means that you can split your JavaScript code into multiple modules, and let Durandal take care of the loading and dependency handling between them. This is done utilizing the functionality of the RequireJS framework, which is a JavaScript library that handles loading of files and modules [40]. A module is defined as a pair consisting of a ViewModel (JavaScript file) and a View (HTML file). The View is just a plain HTML file that can utilize the properties and functions defined in the ViewModel. The basic syntax of a View-Model can be seen in Listing 1. Durandal and RequireJS will here automatically load the *artist* and *knockout* modules specified by ['artist','knockout'], and inject them into the function passed as arguments when the module is loaded.

```
/** artist.js */
/*
 * Function that defines an artist
 */
function Artist(artistName){
    this.artistName=artistName;
}

/** user.js */

/*
 * ViewModel with properties and functions that will be exposed to the View
 */

define(['knockout','artist'],function(ko, artist){
    var self=this;

    self.newArtist=ko.observable();
    self.name=ko.observable("Simen");
    self.username=ko.observable();
    self.listenedTo=ko.observableArray([new Artist("Eminem"),
                                        new Artist("Metallica"),
                                        new Artist("U2")]);

    self.addArtist=function(){
        self.listenedTo.push(new Artist());
        self.newArtist(null);
    }

    self.removeArtist=function(artist){
        self.listenedTo.remove(artist);
    }

    self.activate=function(username){
        self.username(username);
    }

});
```

Listing 2: User ViewModel

For this prototype, a Model View ViewModel pattern was utilized using Durandals Knockout.js support. The view part of the MVVM pattern is in Knockout.js one or more html files that through data-bindings communicate with the ViewModel, be it directly to underlying data or to methods exposed. In Knockout.js these ViewModels are implemented in JavaScript. In Knockout.js, the underlying data-bindings are two-ways, so if the state of the ViewModel changes, the View will automatically be updated. Likewise, if the state of the View changes, the underlying ViewModel will change as well.

```
<!-- user.html -->

<!-- Display the name and username for the user specified -->
<div>Name: <span data-bind="text:name"></span></div>
<div>Username: <span data-bind="text:username"></span></div>

<!-- Display the artist name for each of the artists in the listenedTo array -->
<ul data-bind="foreach:listenedTo">
    <li data-bind="text:artistName, click:removeArtist"></li>
</ul>
<!-- Bind the value of the input field to the newArtist text field, run the
     function addArtist when the enter Key is pressed -->
Add artist: <input type="text" data-bind="value:newArtist, enterKey:addArtist"/>
```

Listing 3: User View

In the example shown in Listing 2 and 3, a simple user view is created that displays the name, username, and what artists a user has listened to. Artists can dynamically be added and removed from that list. The ViewModel as seen in Listing 2, is wrapped in a Durandal module. It specifies what data and functions should be accessible from the view specified in Listing 3. The ko.observable() and ko.observableArray() statement used in the ViewModel are ”special JavaScript objects that can notify subscribers about changes, and can automatically detect dependencies”[41], which means that when these properties changes, the HTML elements bound to these properties through the ‘data-bind’ attribute will automatically be updated. This example is a stripped down version of how the prototype utilizes Knockout.js. The prototype additionally utilizes more advanced data-bindings and self defined Knockout data-bindings, to interact with third party libraries [42].

```
/** shell.js */
router.map([
    {route: 'user/:username', title:'User', moduleId: 'viewmodels/user'},
    {route: 'artist/name/:artistName', title:'Artist Page',
     moduleId: 'viewmodels/artist'}
]).buildNavigationModel();
```

Listing 4: Simple routing in Durandal

Another central feature of Durandal is its routing capabilities. The routing functionality allows for easy building of Single Page Applications by making it easy to define what modules are loaded when a specific URL is navigated to. Listing 4 shows an example on how the routing can be used in practice. If a user navigates to http://www.yourwebsite.com/#*artist*/name/Metallica, the *artist* module will be loaded and information about Metallica would be displayed. As seen, user-specified parameters can be defined using the *:VAR* syntax. These variables can be accessed by creating an *activate* function as seen in Listing 2, where the URL parameter, *username*, defined in Listing 4, is accessed and can be used for further processing. These routes can also be defined using regular expressions.

5.2.2 Back end

The back end is a Java Representational State Transfer (REST) service. It is based on the Jersey[7] REST framework and utilizes a layered architecture. The application is split into three layers: the *view*, *business logic*, and *data access* layers, as seen in Figure 5.2.

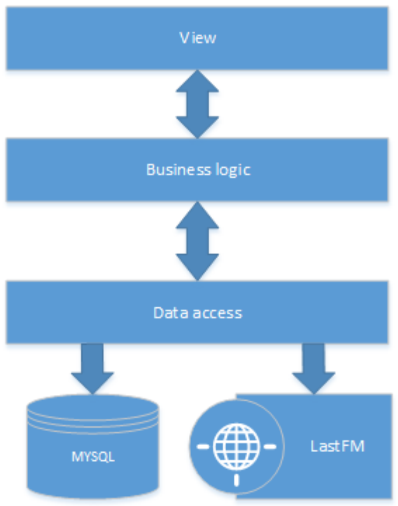
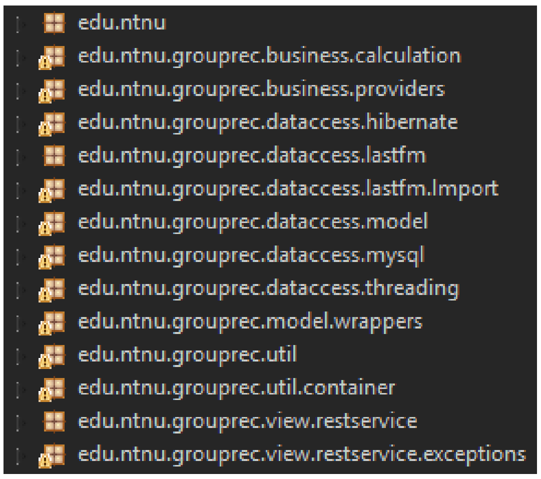


FIGURE 5.2: Layered architecture of the back end



```
edu.ntnu
edu.ntnu.grouprec.business.calculation
edu.ntnu.grouprec.business.providers
edu.ntnu.grouprec.dataccess.hibernate
edu.ntnu.grouprec.dataccess.lastfm
edu.ntnu.grouprec.dataccess.lastfm.Import
edu.ntnu.grouprec.dataccess.model
edu.ntnu.grouprec.dataccess.mysql
edu.ntnu.grouprec.dataccess.threading
edu.ntnu.grouprec.model.wrappers
edu.ntnu.grouprec.util
edu.ntnu.grouprec.util.container
edu.ntnu.grouprec.view.restservice
edu.ntnu.grouprec.view.restservice.exceptions
```

FIGURE 5.3: Back end package structure

The view layer primarily takes care of how the application looks for a user, be it a human or a computer. The business layer acts as a processing layer between the view and the model layer, it is here all the business logic such as similarity and feature calculation resides. The model layer is responsible for persistent

[7]https://jersey.java.net

storage and handling of the data the application uses. From the Java package structure seen in figure 5.3, it can be seen that the application is divided into three main packages: *edu.ntnu.grouprec.business*, *edu.ntnu.grouprec.dataccess*, and *edu.ntnu.grouprec.view*.

All the business logic, including calculation of recommendations, resides in the *edu.ntnu.grouprec.business* package; the REST service in the *edu.ntnu.grouprec.view* package; and communication with the database and external services in the *edu.ntnu.grouprec.dat* package. In addition, the *edu.ntnu.grouprec.model* and *edu.ntnu.grouprec.util* packages contains utility classes and classes that facilitates communication between the layers.

5.2.2.1 Representational State Transfer

Representational State Transfer (REST) is an architectural style of networked systems [43]. It is based on the characteristics described by Fielding [44] as seen in Table 5.1.

Item	Description
Client-Server	The user interface is separated from the data storage. The two components can therefore evolve independently, and Internet-scale applications can be achieved
Stateless	Communication between the client and the server must be stateless. A request from a client must ‘’contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server”.
Cache	Responses must have the capability of being cacheable or non-cacheable to improve network efficiency.
Uniform Interface	A RESTful application must have a standard, uniform interface between components.
Layered System	A RESTful web-service must be designed so that components (such as a proxy server or load balancers) can be placed between the client and the server in a way that the client won’t notice that it is there.

Table 5.1: REST characteristics as described by Fielding [44]

5.2.2.2 Jersey

The back end is written in Java, and is exposing a REST interface utilizing the Jersey REST framework[8]. Jersey supports and extends JAX-RS[9] and provides a simple syntax for writing RESTful web-services. Listing 5 and 6 show the prototype REST support for retrieving user-data and creating a new user respectfully. The *@GET/@POST* annotation specifies that this method should be activated only on HTML *GET/POST requests. @Path* specifies what URL should trigger the method. *@Produces* specifies what sort of data the response contains. *@Consumes* specifies what format the data in the request body is in. Jersey allows for automatically parsing of the data in the body of the request, so in the example in Listing 6 where it is specified that the request body should contain JSON data (through *@Consumes*), Jersey will automatically parse this JSON data into a *UserWrapper* and it will be given as an argument for the method.

```java
/** RestService.java */

/**
 * Method that handles GET requests for a user based on a username.
 * @return Response that contains a user parsed as JSON.
 */
@GET
@Path("/user")
@Produces(MediaType.APPLICATION_JSON)
public Response getUser(@QueryParam("username") String username) {
    try {
        UserWrapper u = userProvider.getUser(username);
        if (u == null)
            return Response
                    .status(404)
                    .entity("User with username: '" + username +
                            " could not be found")
                    .build();
        return Response.status(200).entity(u).build();
    } catch (Exception e) {
        e.printStackTrace();
        return responseError;
    }
}
```

Listing 5: REST interface for retrieving a user

[8]https://jersey.java.net/
[9]https://jax-rs-spec.java.net/

```java
/** RestService.java */

/**
 * Method that handles POST requests to create a user.
 * @return Response status 200 if user was successfully created
 */
@POST
@Path("/createuser")
@Consumes(MediaType.APPLICATION_JSON)
public Response createUser(UserWrapper s) {
    try {
        userProvider.saveUser(s);
        System.out.println("CREATED USER"+s.getUsername());
        return Response.status(200).entity(s.getUsername()).build();
    } catch (Exception e) {
        e.printStackTrace();
        return responseError;
    }
}
```

Listing 6: REST interface for retrieving a user

5.3 Dataset

A dataset was created for the purpose of this project using data from the popular music service Last.fm[10]. The data was fetched using Last.fm's publicly available API[11]. The dataset, as seen in Table 5.2, consists of 2891 concerts in Vancouver, New York, London, Oslo, and surrounding areas; between 18. February 2014 and 6. June 2014; the artists that is performing at these concerts; users that have listened to these artists; and the top artists these users have listened to.

The dataset was built by first fetching concerts within a 100km radius from the specified cities using the *Geo.getEvents(location, radius)* call. Then, information about the artists performing at those concerts was fetched using the *Artist.getInfo(artist)* call. Users that have listened to the artists found, were then fetched using the *Artist.getTopFans(artist)* call, before the 30 most listened to artists for each user are fetched and saved using the *User.getTopArtists(username)* call. In addition to these data, information about the venue each concert is held at, and the most used tags for each artist were added using similar calls. Usage of the algorithms presented in Section 4.6 created additional data present in the dataset: 17025096

[10]http://www.last.fm
[11]http://www.last.fm/api

Property	Column	Count
Users	user	25720
Artists	artist	80877
Concerts	concert	2891
Listening counts	artistlistenings	769370
Tags	tag	159348
Tags for artists	artisttag	1358715
Artist concert participation	concertparticipation	6845
User similarities	usersimilarity	17025096
Venues	venue	596
User features	userfeatures	17025096
Artist features	artistfeatures	5085312

TABLE 5.2: Dataset properties

cosine similarities between users for the K-nearest neighbor algorithm; and 1648320 user features and 5085312 artist features for the MF algorithm.

The dataset was created using a MYSQL database with the database schema seen in Figure 5.4.

The dataset used is not distributed as part of this project due to LastFm's licensing agreement, however code to build a similar dataset is provided (see Appendix B).

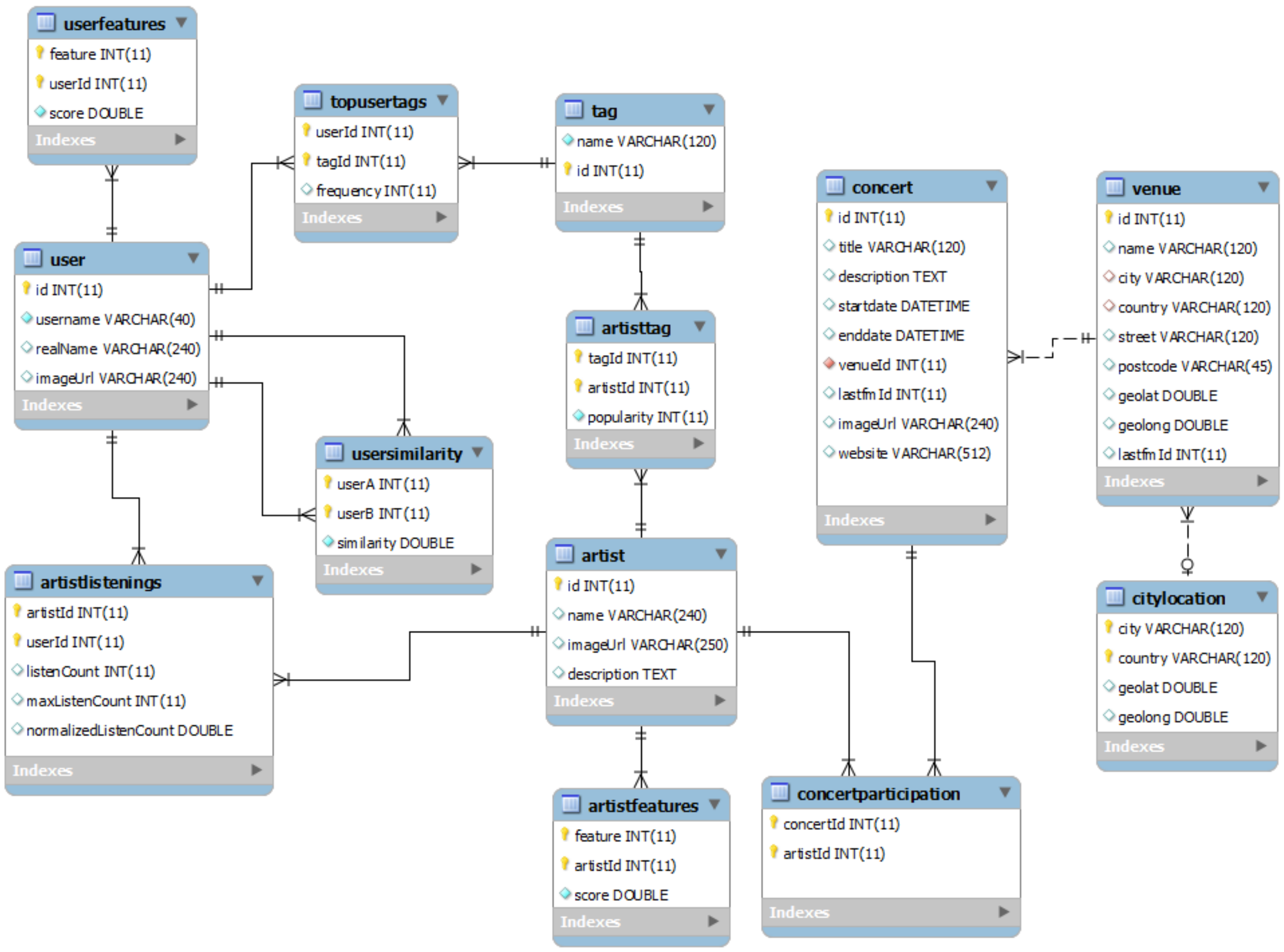


FIGURE 5.4: Dataset schema

5.4 Recommendation Algorithm Implementation

The k-Nearest Neighbor and Matrix Factorization algorithms described in Sections 4.6.1 and 4.6.2 both have parts that are resource intensive. Therefore, the algorithm described in Section 4.2 is split into two parts, an offline resource intensive part, and an online part performed on the fly. The main workflow of the algorithm can be seen in Figure 5.5. Normalization of ratings, calculation of similarities between users for the kNN algorithm, and calculation of user/artist features for the Matrix Factorization algorithm, are performed offline; whereas the actual prediction of ratings is performed online on the fly. The algorithms have their own pre-calculation and rating-prediction steps, the other steps are similar for both algorithms.

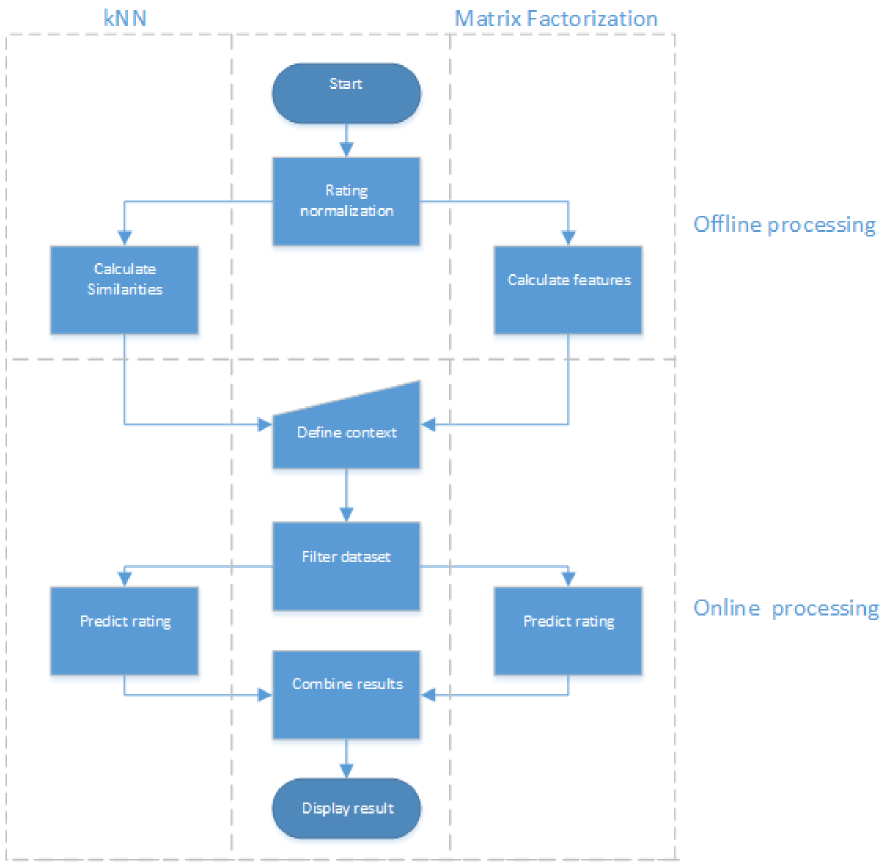


FIGURE 5.5: Algorithm flow chart

5.4.1 Rating Normalization

The rating normalization part explained in Section 4.4, is performed using the code provided by Celma [17] (the original code can be found on GitHub[12]). This code is written in the *R* statistical programming language [13].

To be able to use the $F(x)$ function, values to use for α, β and $N50$ have to be estimated. This is done by fitting the listen counts for the user to $F(x)$, using the Gauss-Newton non-linear least squares regression model *nls* in ***R***, as seen in Listing 7.

```r
#vector with the cummulative sums of listenCounts
cum = cumsum(as.numeric(listenCounts))
#vector with the cummulative sums of listenCounts as percentage of total
cum_pcnt = cum / cum[length(cum)] * 100

startN50 = N50(cum_pcnt)
startBeta = 1
startAlfa = 0.75

rank = 1:length(listenCounts)

dataset = data.frame(cum_pcnt)

#Fit cum_pcnt to the CDF function for heavy-tailed curves, F
fit = nls( cum_pcnt ~ F(rank, n50, beta, alfa), data = dataset,
           start = list(n50=startN50, beta=startBeta, alfa=startAlfa) )
```

Listing 7: Fitting listen counts to F(X) with non-linear least squares in R (based on Celma [17])

5.4.2 User Similarity Calculation for k-Nearest Neighbor

The offline part of the kNN algorithm consists of calculating user similarities. This is done as described in Section 4.6.1.2, by calculating the cosine similarity between their respective listen counts. The database stores all similarities greater than zero. A missing similarity record can therefore be seen as an indication that the similarity between the two users is zero. It is important to note here that similarities between users are stored twice, similarities between users a and b are

[12]https://github.com/ocelma/long-tail-model
[13]www.r-project.org/

stored both as $(a,b,similarity)$ and $(b,a,similarity)$. This is done for simplicity and performance reasons, so that when you query for the users similar to a user, you only have to perform the query on one column. Listing 8 shows how the cosine similarity between two vectors is implemented.

```java
/**
* Calculates the Cosine Similarity between vecA and vecB
*/
public static double calculateCosineSimilarity(double[] vecA, double[] vecB) {
    if (vecA.length != vecB.length)
        return 0.0;

    double dotProduct = 0;
    double magnA = 0;
    double magnB = 0;

    for (int i = 0; i < vecA.length; i++) {
        dotProduct += vecA[i] * vecB[i];
        magnA += Math.pow(vecA[i], 2);
        magnB += Math.pow(vecB[i], 2);

    }
    magnA = Math.sqrt(magnA);
    magnB = Math.sqrt(magnB);

    if (magnA != 0.0 || magnB != 0.0) {
        return dotProduct / (magnA * magnB);
    } else {
        return 0.0;
    }

}
```

Listing 8: Calculating cosine similarities

5.4.3 Feature calculation for Matrix Factorization

The offline part of the MF algorithm consists of extracting the latent features of the listening counts in the dataset as explained in section 4.6.2. The implementation is based on Timely Developments[14] implementation of the algorithm [45]. The implementation of the feature calculation step can be seen in Appendix B.1. The calculated features are then saved in the $userfeatures$ and $artistfeatures$ tables of the database.

[14]http://www.timelydevelopment.com/demos/NetflixPrize.aspx

Number of features	Max iterations	γ	λ	Minimum improvement
64	120	0.0009	0.3	0.000001

Table 5.3: Choice of constants for MF implementation

5.4.3.1 Choice of constants

The MF algorithm uses, as described, constants that should be decided based on the dataset in question. Both λ; γ; *maxIterations*,the maximum amount of iterations before the feature calculation terminates; *minImprov*, the minimum RSME you can see before the feature calculation terminates because of convergence; can be adjusted. To determine what value these constants should have, a 10-fold cross validation was performed on the dataset. First, all the (user, artist, listenCount) triples in the dataset were split into 10 equal parts. Then, in turn (fold) one of the parts was used as a test set whereas the other 9 were used as a training set. This means that the MF algorithm was performed on the training set to calculate features. Then these features were used to predict the ratings of the (user, artist) part of the triples in the test set. These predicted ratings was then compared to the actual ratings of the triple to calculate the RMSE between them. The average of these 10 RMSE values were saved as a score for the given parameters.

The RMSE values can be seen as a measurement on how well the algorithm predicts the ratings already present in the dataset. Therefore, the parameters with the lowest RMSE score were chosen to be used for the prototype. The parameters chosen after running the cross validation on multiple possible parameters can be seen in Table 5.3. These parameters yielded a RMSE score of 2.94, and an average error in rating prediction of 2.48 out of 10.

5.4.4 Context definition

The definition of context consists of defining a group to calculate recommendations for, in addition to define where and in what timespan the recommended concerts should be in. This is all performed by the user in a web browser. Groups are defined by searching for users and clicking on them as seen in Image 6.2. Timespan and location are selected by using the controls provided as seen in Image 6.4 and 6.5.

5.4.5 Filter dataset

As explained in Chapter 4.5, the dataset is filtered before predicting ratings with the MF and kNN algorithms, so that data irrelevant to the results can be left out.

5.4.5.1 Filter concerts

After the context defined arrives at the backend-server through its REST interface, the context is applied to the dataset to filter out irrelevant data. First, concerts happening within the given timespan and at, or close to the location specified, are fetched with the SQL query in Listing 9.

```
SELECT conc.id, cp.artistId, conc.distance
FROM ((
   SELECT c.id AS id, cp.artistId AS artistId,
      DISTANCE() AS distance
   FROM concert c, citylocation cl, venue v
   WHERE  c.startdate>=START_DATE
      AND c.startdate<=END_DATE
      AND v.city!=LOCATION
      AND cl.city=LOCATION
      AND v.id=c.venueId
      AND(enddate IS NULL OR enddate>=?)
   ORDER BY distance ASC LIMIT 100
   ) conc
)
LEFT JOIN concertparticipation AS cp ON conc.id=cp.concertId;
```

Listing 9: Filter dataset based on distance

5.4.5.2 Relaxing of context

To enable for relaxation of the location parameter, the 100 concerts closest to the location specified is also fetched. This is done by utilizing the Haversine formula as seen in Listing 10 (DISTANCE()). This formula can be used to estimate the shortest distance between two points on the earth surface[46]. In addition, concerts within 5 days of the date range specified is fetched to support relaxing of the date range parameter.

```
SELECT (6371 *
     ACOS(COS(RADIANS(v.geolat)) * COS(RADIANS(cl.geolat))
   * COS(RADIANS(cl.geolong)     - RADIANS(v.geolong))
   + SIN(RADIANS(v.geolat))      * SIN(RADIANS(cl.geolat)))
AS distance
FROM venue v, concertlocation cl
```

Listing 10: The DISTANCE() function, calculating distance between two points on the earth surface.

After these concerts have been fetched, the kNN and MF algorithms will predict ratings for these concerts and select the k concerts with the highest predicted ratings.

5.4.6 Predict ratings with kNN

As specified in section 4.6.1.2, the outcome of kNN is based on what ratings the most similar users to the user in question have given to the artists performing at the relevant concerts. Therefore, the users with a similarity to the user in question greater than 0, and that have listened to one or more of the artists performing at the concerts found before, are fetched for each of the members of the group. This is done with the SQL query in listing 11.

```
SELECT a.userId, a.artistId, a.listenCount, us.similarity
FROM artistlistenings a, usersimilarity us
WHERE  us.userA=GROUP_USER
   AND a.userId=us.userB
   AND artistId IN LIST_OF_ARTIST_IDs
```

Listing 11: Fetch artist listeners

For each of the artists found in the previous step, a rating is predicted using the code in Appendix B.2. For each of the artists, the k most similar users to the user being processed that also have listened to the artist, is selected. Each of these k users contribute to the final rating for the artist with its rating of the artist times its similarity to the user being questioned squared, divided by the total similarity of the k users ($sim^2/totSim$). The division ensures that the most similar users contribute most to the final rating. By squaring the similarities, this contribution from the most similar users is even greater.

After each of the artists have got a predicted rating, a final rating for each of the concerts is predicted by using the average predicted rating from all of the artists performing at that concert.

The steps above (starting from section 5.4.6) is then repeated for all the members of the group.

When all of the members of the group have got a predicted rating for each of the concerts, the final combined predicted group ratings for the concerts are calculated by using the average rating for that concert.

5.4.6.1 Choice of k

Choosing k in a k-Nearest Neighbor algorithm has a great impact on the predictions given. A choice of k too high might result in recommendation of only popular items. A k too low can introduce noise data into the recommendations as only a small set of people decide the prediction.

By observation, a fixed k doesn't work well in the case of a CRS, where the number of items that can be recommended is greatly reduced based on the context applied. Firstly, all of the k users have to have listened to one or more of the same artists as the user asking for recommendations. Secondly, all of the k users have to have listened to artists performing at the concert a rating is being predicted for. For lesser known artists, the number of users that meet these criteria can be low, and thus k users cannot be found for the artist. This will lead to an extra emphasis on well known artists, however when no well known artists are performing within the given time and location constraints, potentially no concerts will get a final rating.

This shows the need for an adaptive k for each artist. Therefore, in this thesis, k is set as $\sqrt{n}$, where n is the number of people having listened to the artist being evaluated, a, and also have a similarity >0 with the user the recommendations are for, u. By choosing this k, noise data is removed when predicting ratings for well known artists as k is likely to be high, but at the same time, it allows for recommendation of concerts that otherwise wouldn't be recommended because they wouldn't have k users that fit the criteria.

$$k_{ua} = \sqrt{n} \tag{5.1}$$

5.4.7 Predict ratings with Matrix Factorization

In the MF approach, a rating for each of the artists found in section 5.4.5 is first predicted. This is done by performing the dot product between the latent feature vector for the user being processed and the latent feature vector for the artist as explained in section 4.6.2. An implementation of this can be seen in Listing 12.

```
double predRating = rating.pseudoAvg;
for (int f = 0; f < maxFeatures; f++) {
    predRating += artistFeatures[f][artistId] * userFeatures[f][userId];
    if (predRating > MAX_RATING)
           predRating = MAX_RATING;
       if (predRating < MIN_RATING)
           predRating = MIN_RATING;
}
return predRating;
```

Listing 12: MF calculation of artist score

Similarly to the kNN approach, the predicated rating for a concert from a user is the average of the predicted ratings for the artists playing at that concert. These steps are also here repeated for all the members of the groups. The final combined predicted group ratings for the concerts are also here the average rating predicted for that concert.

5.4.8 Merge results (Hybrid Recommendation)

As explained in Section4.7, after each concert have got a predicted rating from both the MF and the kNN algorithm, the final rating for a concert is defined as the average between the results from the two algorithms. These ratings makes up the rating given by the Hybrid algorithm.

The 10 concerts with the highest predicted rating within the given context, the 10 top concerts from the 100 extra concerts fetched based on distance, and the top 10 concerts from the extra concerts based on date are finally returned to the front end for viewing by a user.

5.5 New users

The algorithm outlined above has an offline part performed when deemed necessary, and an online part that is performed on the fly. When a new user registers, data for that user has to be calculated and inserted into the database to be able to receive recommendations. However, since the offline part of the algorithm is not frequently run, the new user wouldn't be able to get recommendations until it is run again. This is not feasible for instant recommendations as this is a very time consuming process. For the $\boldsymbol{k}$NN part of the algorithm, this is overcome by calculating the similarity between the new user and the users already in the database, using the artists the user specified on the registration page. When these similarities have been calculated, $\boldsymbol{k}$NN can be used as normal also for that user. For the MF part of the algorithm, the latent factors of the most similar user to the user created based on the artists specified, is used in the recommendation process until the offline part of the algorithm is run again.

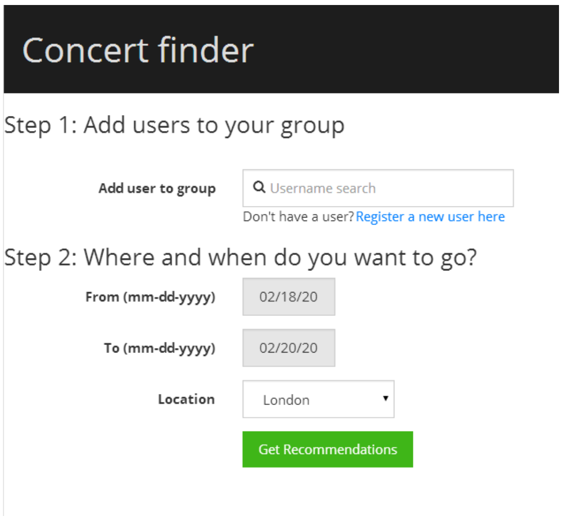
Chapter 6

Prototype

In this chapter, the main functionality of the prototype and how it can be used in practice, is presented in addition to how the prototype meets the requirements identified in Chapter 4.1. It shows how users can be created (Section 6.2), how groups can be formed (Section 6.1), how context can be specified (Section 6.1), how the recommendation process can be initialized, and what the results from the recommendation process look like (Section 6.4). It also shows how the context can be relaxed by a user (Section 6.4.1).

6.1 Main View

Image 6.1 shows the main view of the prototype. A user can here specify which users should be added to the group that is used for recommendations, and specify the context (location and time) for the recommended concerts. Users can be added to a group as seen in Image 6.2 and 6.3. Dates for the concerts can be chosen via a simple date picker (see Image 6.4), and locations can be selected from a dropdown list (see Image 6.5). This shows that the prototype takes the location and time into account when providing recommendations as requirements R2 and R4 specifies.


Concert finder

Step 1: Add users to your group

Add user to group | Username search

Don't have a user? Register a new user here

Step 2: Where and when do you want to go?

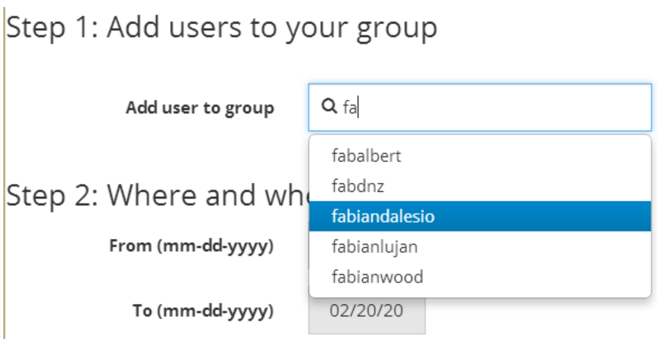
From (mm-dd-yyyy) 02/18/20

To (mm-dd-yyyy) 02/20/20

Location London

Get Recommendations

FIGURE 6.1: Prototype main view


Step 1: Add users to your group

Add user to group | fa

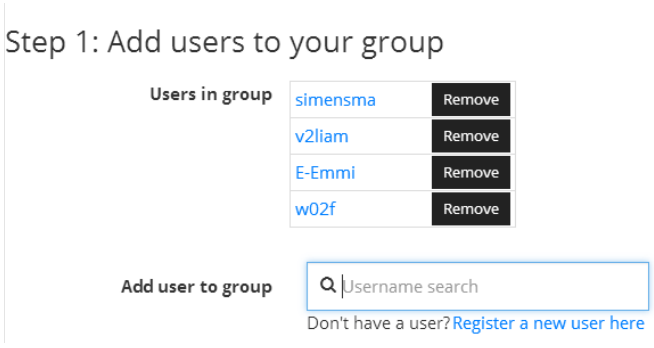
fabalbert
fabdnz
fabiandalesio
fabianlujan
fabianwood

Step 2: Where and wh

From (mm-dd-yyyy)

To (mm-dd-yyyy) 02/20/20

FIGURE 6.2: Prototype user search view


Step 1: Add users to your group

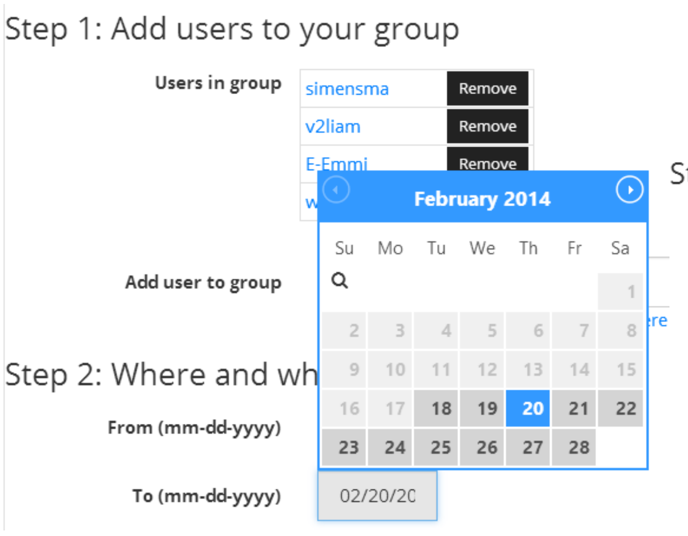
Users in group

simensma	Remove
v2liam	Remove
E-Emmi	Remove
w02f	Remove

Add user to group | Username search

Don't have a user? Register a new user here

FIGURE 6.3: Prototype group view


Step 1: Add users to your group

Users in group

simensma Remove
v2liam Remove
E-Emmi Remove

February 2014

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	

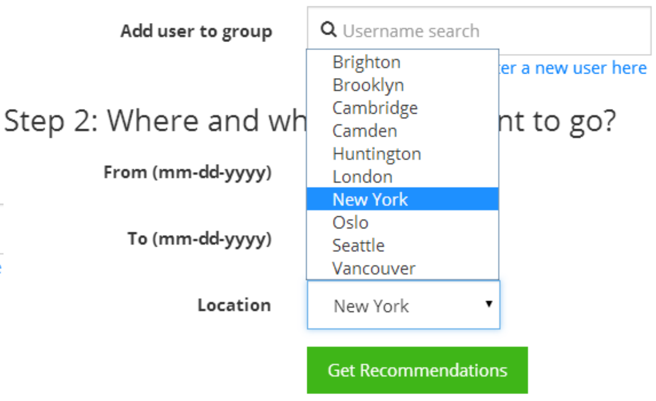
Add user to group

Step 2: Where and wh

From (mm-dd-yyyy)

To (mm-dd-yyyy) 02/20/20

FIGURE 6.4: Prototype date selection view


Add user to group | Username search

Step 2: Where and wh … nt to go?

From (mm-dd-yyyy)

To (mm-dd-yyyy)

Brighton
Brooklyn
Cambridge
Camden
Huntington
London
New York
Oslo
Seattle
Vancouver

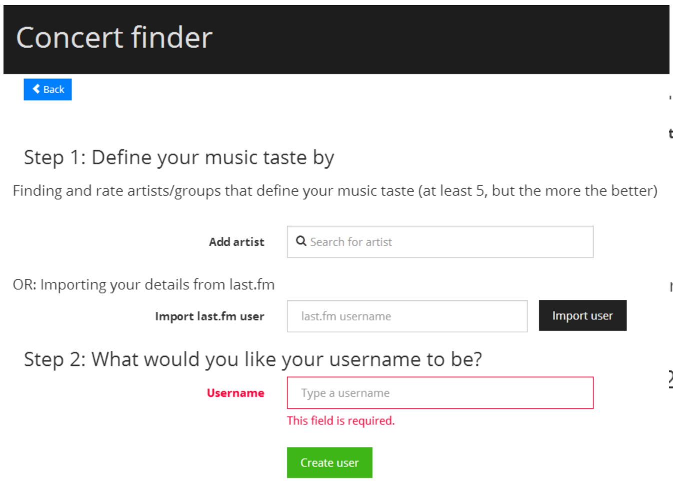
Location New York

Get Recommendations

FIGURE 6.5: Prototype location selection view

6.2 User Creation View

A new user can be created in the user creation view (see Image 6.6) by specifying a unique username, and rating at least 5 artists that defines the new users music taste. By requiring the user to specify these artists, the well known cold start problem of collaborative filtering algorithms (see Section 2.1.6.1) is mitigated. The user specifies these artists by searching amongst all the artists in the Last.fm dataset(see Image 6.7) which then are imported into the dataset if necessary.



Concert finder

Back

Step 1: Define your music taste by

Finding and rate artists/groups that define your music taste (at least 5, but the more the better)

Add artist: Search for artist

OR: Importing your details from last.fm

Import last.fm user: last.fm username — Import user

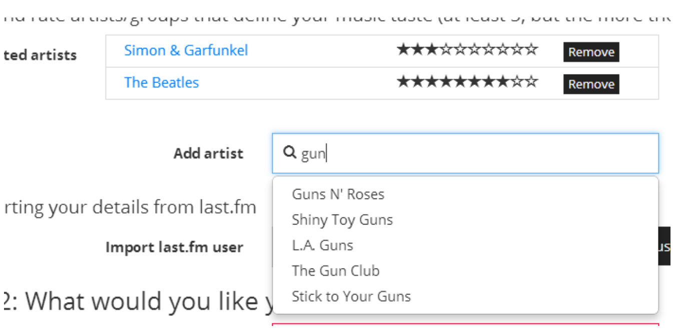
Step 2: What would you like your username to be?

Username: Type a username

This field is required.

Create user

FIGURE 6.6: Prototype create user view



ted artists

Simon & Garfunkel	★★★☆☆☆☆☆☆☆	Remove
The Beatles	★★★★★★★★☆☆	Remove

Add artist: gun

Guns N' Roses
Shiny Toy Guns
L.A. Guns
The Gun Club
Stick to Your Guns

rting your details from last.fm

Import last.fm user

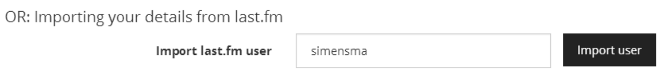
2: What would you like y

FIGURE 6.7: Prototype artist selection view

Requirement R1 stated that concert recommendations should be based on a user's listening habits. The prototype fulfils this by requiring new users to specify and rate their favorite artists. This data is the basis for the kNN and MF algorithms as explained in Chapter 4.2.

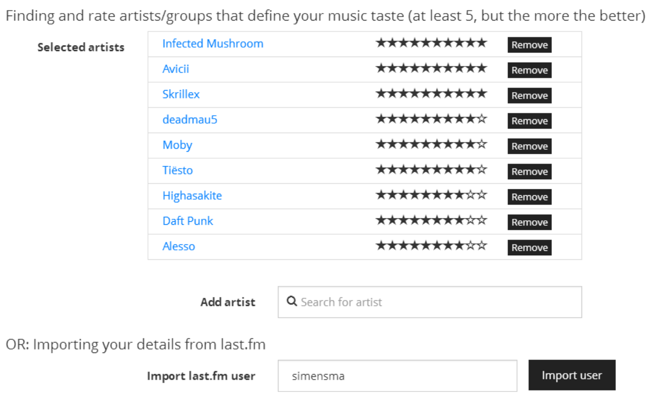
6.2.1 Import User from Last.fm

Instead of specifying artists manually, a user can also import his listening details from Last.fm by specifying his Last.fm username and clicking the *Fetch user* button as seen in Image 6.8. His listening habits will then be fetched from Last.fm and added to the above list as seen in Image 6.9.


OR: Importing your details from last.fm

Import last.fm user	simensma	Import user

FIGURE 6.8: Import of user from last.fm


Finding and rate artists/groups that define your music taste (at least 5, but the more the better)

Selected artists

Infected Mushroom	★★★★★★★★★★	Remove
Avicii	★★★★★★★★★★	Remove
Skrillex	★★★★★★★★★★	Remove
deadmau5	★★★★★★★★★☆	Remove
Moby	★★★★★★★★★☆	Remove
Tiësto	★★★★★★★★★☆	Remove
Highasakite	★★★★★★★★☆☆	Remove
Daft Punk	★★★★★★★★☆☆	Remove
Alesso	★★★★★★★★☆☆	Remove

Add artist Search for artist

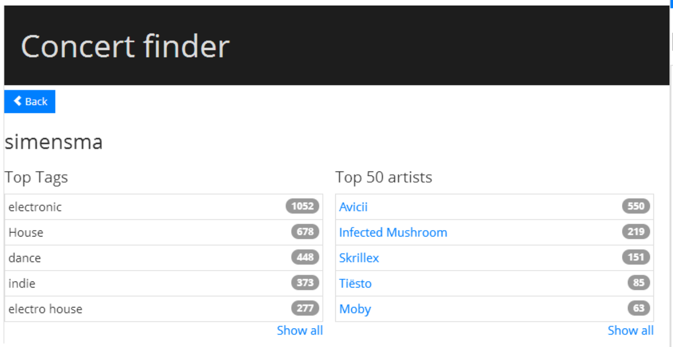
OR: Importing your details from last.fm

Import last.fm user simensma Import user

FIGURE 6.9: Artist-listenings for an imported user

6.3 User and Artist view

By clicking on a user link, the artists most listened to, and the tags most used for those artists, are displayed as seen in Image 6.10. Similarly, if an artist link is clicked, information about the artist is displayed in addition to the tags most used to described the artist (see Image 6.11).

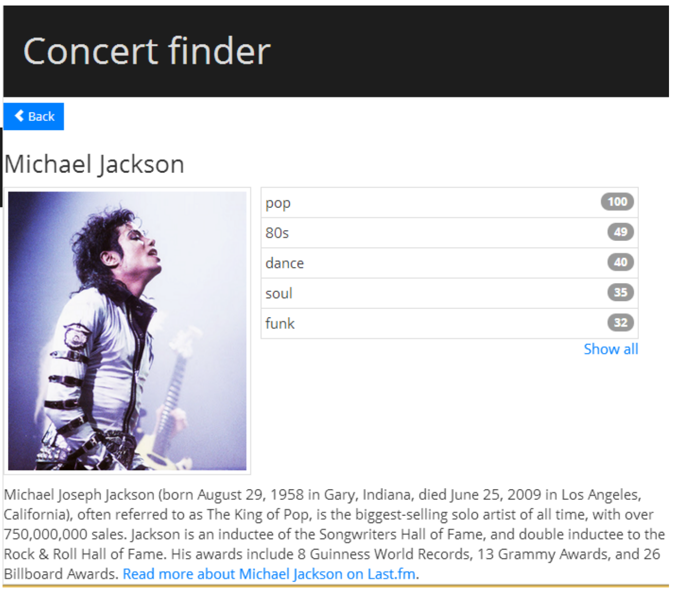

Concert finder

Back

simensma

Top Tags		Top 50 artists	
electronic	1052	Avicii	550
House	678	Infected Mushroom	219
dance	448	Skrillex	151
indie	373	Tiësto	85
electro house	277	Moby	63
Show all		Show all	

FIGURE 6.10: Prototype user view


Concert finder

Back

Michael Jackson

pop	100
80s	49
dance	40
soul	35
funk	32

Show all

Michael Joseph Jackson (born August 29, 1958 in Gary, Indiana, died June 25, 2009 in Los Angeles, California), often referred to as The King of Pop, is the biggest-selling solo artist of all time, with over 750,000,000 sales. Jackson is an inductee of the Songwriters Hall of Fame, and double inductee to the Rock & Roll Hall of Fame. His awards include 8 Guinness World Records, 13 Grammy Awards, and 26 Billboard Awards. Read more about Michael Jackson on Last.fm.

FIGURE 6.11: Prototype artist view

6.4 Result view

By clicking the "Get recommendation" button in the main view after all the fields are filled out, the view switches to the result view (see Image 6.12), where the recommended concerts can be seen in ascending order. The predicted rating of a concert can be seen in the upper hand corner of a concert tile (yellow star). The left side of the concert tile shows the artists performing at the concert, sorted in descending order based on the highest predicted rating. The right hand of the concert tile shows the top tags for the artists playing at the concert, in descending order based on tag frequency.

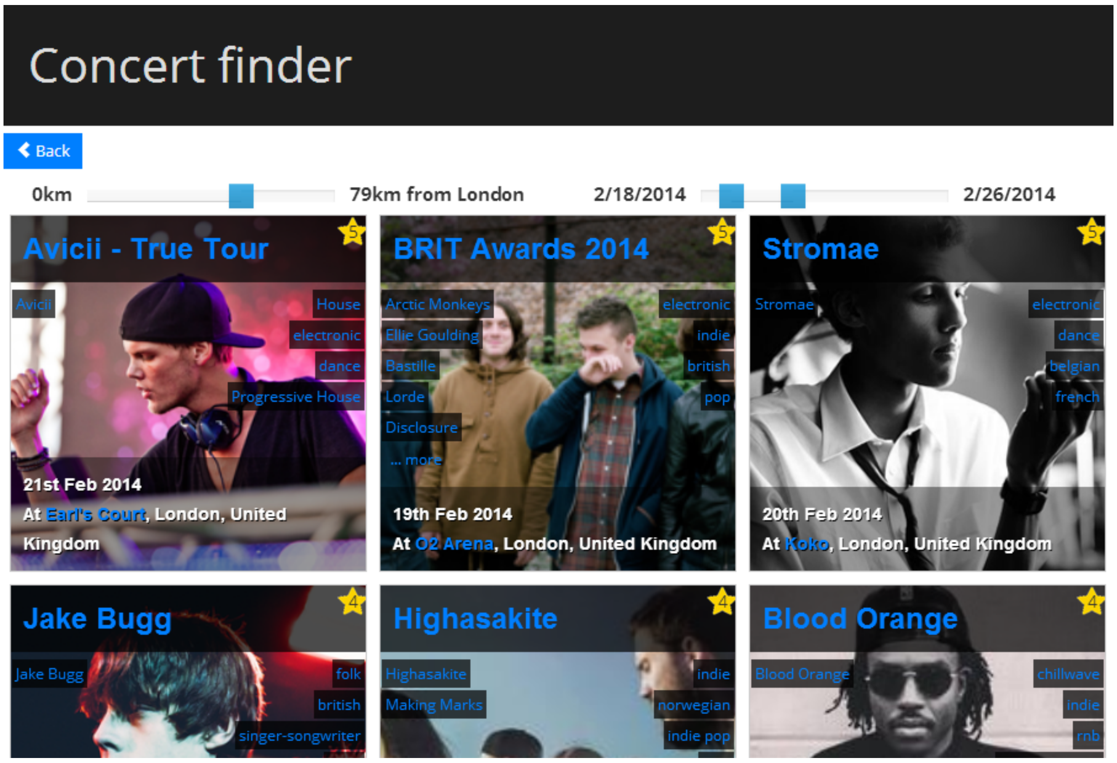


FIGURE 6.12: Prototype result view

6.4.1 Context relaxation

The application automatically fetches concerts in the areas surrounding the specified location so that the user easily can relax the location parameter and find extra concerts if wanted. This can be achieved by using the slider on the top left of the page (see Image 6.13), that dynamically shows or hides extra concerts within the radius specified by the slider.

Similarly to location relaxation, the application automatically fetches concerts in the dates surrounding the specified dates with the date relaxation feature. A user can then easily relax the date parameters to show concerts for date ranges smaller or bigger than the one originally specified. This is done by utilizing the sliders on the top right of the page (see Image 6.14). New concerts are then dynamically added or hidden based on the new date range specified by the sliders.

As seen, a user can relax the location and time of the concerts by utilizing the sliders provided. This shows that requirements R3 and R5 are fulfilled.

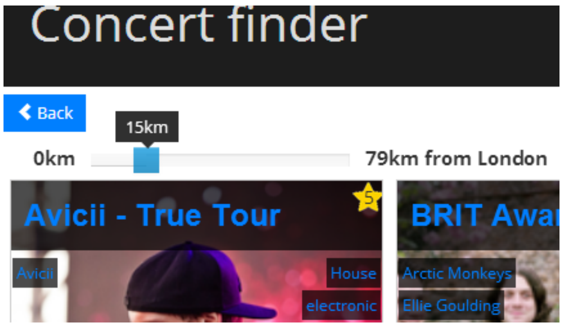


FIGURE 6.13: Relaxing location parameter

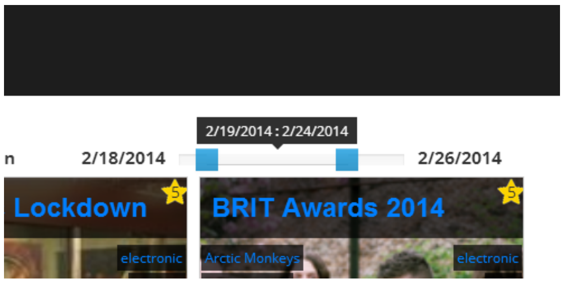
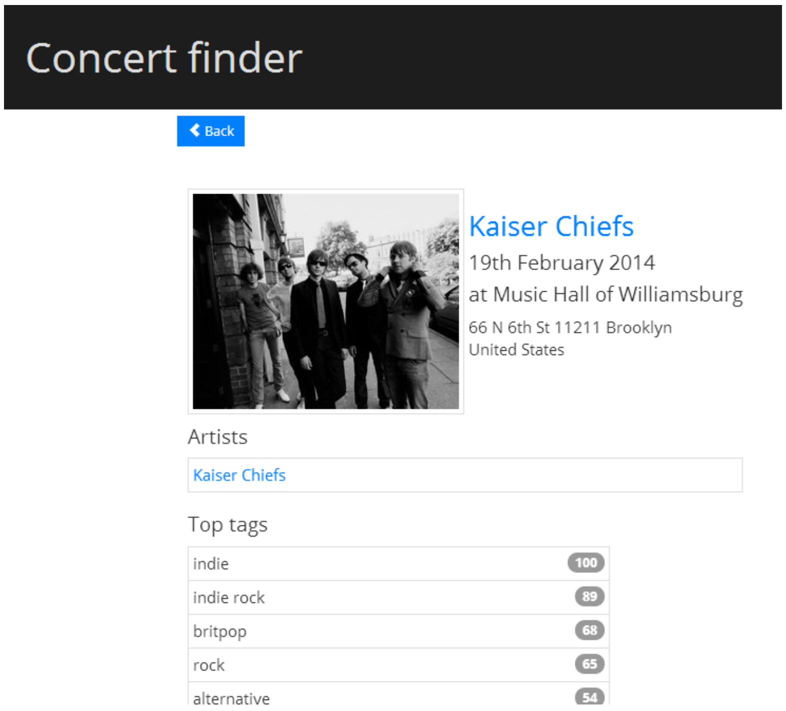


FIGURE 6.14: Relaxing date parameters

6.5 Concert view

By clicking one of the concert tiles, a view of the concert will be visible (see Image 6.15). Here, information such as the location and time, a description of the concert, the top tags of the artists playing at the concert, and links to ticket distributors (if present) will be displayed.


Concert finder

Back

Kaiser Chiefs
19th February 2014
at Music Hall of Williamsburg
66 N 6th St 11211 Brooklyn
United States

Artists

Kaiser Chiefs

Top tags

indie	100
indie rock	89
britpop	68
rock	65
alternative	54

FIGURE 6.15: Prototype concert view

Chapter 7

Evaluation

Evaluation of the prototype was performed in two phases. After the first period of development, the usability of the prototype was evaluated using the System Usability Scale and an Application specific Survey. After the second period of development, the Quality of the Recommendations given was evaluated with a user centered method.

7.1 Usability evaluation results

To evaluate the usability of the prototype, 15 users were asked to perform the steps outlined in Chapter 3.3.1.

7.1.1 Number of test subjects

15 users participated in the usability survey performed after the first period of development. Nielsen [47] suggests that 5 users are enough to find the majority of usability problems of a system, those 5 participants could reveal about 80% of all usability problems. This can be achieved by running many small tests and iterations. Faulkner [48] states that this number is dependent on the types of users participating. It was shown that in some cases, 5 participants found 99% of the

Question	Item	N	%
Gender	Female	2	13
	Male	13	86.7
Age	15	1	6.67
	22	1	6.67
	23	6	40
	24	3	20
	25	2	13.33
	27	1	6.67
	57	1	6.67
Occupation	Student	14	93.33
	Teacher	1	6.67
Device	Smart-phone	2	13
	Tablet	1	7
	Laptop	10	67
	Desktop Computer	2	13

Table 7.1: Usability survey demographics

problems and in other cases only 55%. With 10 users participating, the minimum of problems found was 82% and with 15 participant the minimum number of problems found was 90%. In general, one should run usability tests with as many participants that schedules, budgets, and availability allow.

7.1.2 Demographics

Table 7.1 lists some demographic information about the participants, including gender, age, occupation and what device the survey was undertaken on. The table shows that 13 of the participants were male (86.7%), and 2 of them female (13%). The age of the participants varied between 15 and 57 years, with a mean of 27.6 years, median of 24 years and standard deviation of 13.5 years. Most participants listed *Student* as their occupation (93% of the participants). 67% of the participants used a laptop when performing the survey.

7.1.3 User Background Survey results

The participants were asked to answer four simple questions to determine if they had previous knowledge of other music recommendation systems, and how satisfied they were with the recommendations provided from them. The results from this survey can be seen in Table 7.2.

Question	Item	N	%
BI1: How often do you use a recommendation service for concerts such as LastFm, Eventful or Bandsintown?	Every day	0	0
	A few times a week	0	0
	A few times a month	1	7
	Less often	3	20
	Never	11	73
	Don't know	0	0
BI2: In general, how satisfied are you with the quality of recommendation from these services?	Very dissatisfied	0	0
	Dissatisfied	1	7
	OK	3	20
	Satisfied	0	0
	Very satisfied	0	0
	Don't know	11	73
BI3: How often do you listen to music recommended to you by services such as Spotify, ITunes, LastFm or Pandora?	Every day	0	0
	A few times a week	7	47
	A few times a month	6	40
	Less often	2	13
	Never	0	0
	Don't know	0	0
BI4: In general, how satisfied are you with the quality of recommendation from these services?	Very dissatisfied	1	7
	Dissatisfied	0	0
	OK	7	47
	Satisfied	6	40
	Very satisfied	1	7
	Don't know	0	0

Table 7.2: Usability survey user background

The results from question BI1 shows that 73% of the participants never had used a concert recommendation service before. The ones that had tried it before, used it a few times a month or less often. 3 out of the 4 participants that had tried such a service before, said that they were *OK* with the recommendations provided of the service, and 1 participant was *dissatisfied*. The fact that only 4 out of 15 participants had tried a concert recommendation service for concerts, might indicate that either the demand for such a service is not there, or the quality of existing services are not appealing to the users. The results from question BI3 shows that all the participants had tried a music recommendation service before. 47% of them are using it a few times a week, 40% a few times a month and 13% less often. 87% of the participants answered either *OK* or *Satisfied* when asked how satisfied they were with the quality of recommendation from these music recommendation services. From these results it is apparent that the participants in general use music recommendation systems from time to time, however there are still some to improve the recommendations given.

7.1.4 System Usability Scale results

Results, question by question, from the SUS survey can be seen in Table 7.3. Utilizing the method described in Section 3.3.1.1, the results yield a SUS score of 79.83 as seen in Table 7.4. According to the adjective rating scale outlined, a SUS score of 79.83 would be rated as somewhere between *Good* and *Excellent*. There is no absolute when it comes to usability, but a score of 79.83 is a good indication on that the users found the usability of the prototype satisfactory.

	Strongly disagree		Disagree		Undecided		Agree		Strongly agree	
Question	N	%	N	%	N	%	N	%	N	%
SUS1	2	13	2	13	8	53	3	20	0	0
SUS2	7	47	6	40	2	13	0	0	0	0
SUS3	0	0	0	0	2	13	6	40	7	47
SUS4	15	100	0	0	0	0	0	0	0	0
SUS5	0	0	1	7	2	13	8	53	4	27
SUS6	6	40	5	33	4	27	0	0	0	0
SUS7	0	0	0	0	0	0	6	40	9	60
SUS8	7	47	6	40	2	13	0	0	0	0
SUS9	0	0	2	13	4	27	5	33	4	27
SUS10	12	80	2	13	0	0	1	7	0	0

TABLE 7.3: SUS results

Question	Average SUS Score
SUS 1: I think that I would like to use this system frequently	2.8
SUS 2: I found the system unnecessarily complex	1.67
SUS 3: I thought the system was easy to use	4.33
SUS 4: I think that I would need the support of a technical person to be able to use this system	1
SUS 5: I found the various functions in this system were well integrated	4
SUS 6: I thought there was too much inconsistency in this system	1.87
SUS 7: I would imagine that most people would learn to use this system very quickly	4.6
SUS 8: I found the system very cumbersome to use	1.67
SUS 9: I felt very confident using the system	3.73
SUS 10: I needed to learn a lot of things before I could get going with this system	1.33
SUS score	79.83

Table 7.4: SUS scores

7.1.5 Application Specific Survey results

Table 7.6 shows the results from the Application Specific Survey (AS). The participants were asked to answer four question specific to the prototype. 66% of the participants checked 4 or 5 when asked if they believe they could get use of this application in the future. When asked if the application provided recommendations relevant for the user, the average response was 3.27 on a scale from 1 to 5 where 1 is defined as *strongly disagree* and 5 as *strongly agree* as seen in Table 7.5. The average response when asked if a participant found it easy to find recommendations with a location close to the location specified was 4.1 on the same scale. The average response when asked if a participant found it easy to find recommendations on days close to the date range specified was 4.27.

From these results it is apparent that the participants had mixed feelings about how relevant the recommendations given were, which is the main focus of the second period of development. Average scores of over 4.1 for both AS3 and AS4 indicate that the participants were satisfied with how the current context relaxation options works.

Question	Average	Median	Standard deviation
AS 1	3.73	4	1.33
AS 2	3.27	3	1.16
AS 3	4.1	4	1.1
AS 4	4.27	5	1.1

Table 7.5: Application specific survey result properties

Question	Item	N	%
AS1: I believe I could get use of this application in the future	1	2	13
	2	0	0
	3	3	20
	4	5	33
	5	5	33
AS2: The application provided recommendations relevant for me	1	1	7
	2	3	20
	3	4	27
	4	5	33
	5	2	13
AS3: I found it easy to find recommendations with a location close to the location I specified	1	1	7
	2	0	0
	3	2	13
	4	6	40
	5	6	40
AS4: I found it easy to find recommendations on days close to the date range I specified	1	1	7
	2	0	0
	3	1	7
	4	5	33
	5	8	53

Table 7.6: Application specific survey results

7.1.5.1 Usability problems

From the free-text answers to AS5, AS6 and AS7 seen in Appendix A.4, some usability challenges were identified:

- No information about dates could be found when clicking on a concert.
- No ability to view all the concerts in a location or timespan
- No genres are displayed on concerts, so it's impossible to say anything what type of music new artists play
- Change color of text, blue is not very good for reading.
- Scary to leave group setup view to register new users, as there is no indication that the state will be kept
- Lack of clear and easy to read messages

Solutions to Item 1 and 3; missing information about dates and no genres on concerts; were in the second period of development implemented as seen in Section 6.5 and Section 6.4 respectively. However, as the second development period focused on improving the recommendation algorithm, the rest is left for future work.

7.1.5.2 Feature suggestions

From the free-text answers to AS5, AS6 and AS7 as seen in Appendix A.4, some suggestions to new functionality were identified:

- Plot concert location on map
- Import user details from Last.fm
- Import user details from Spotify or other similar services
- Edit user functionality
- Add new locations
- Filter concerts based on genre
- Finding friends through Facebook
- Ability to exclude concerts or genres from results.

The functionality for importing user details from Last.fm was implemented in the second period of development (see Section 6.2.1). The rest of the suggested features are left for future work.

7.2 Recommendation Quality Evaluation

For the purpose of Quality Evaluation, the result view of the prototype was slightly adjusted. Now, it displays three different paragraphs with recommendations (cases) as seen in Image 7.1. One paragraph contains the results when running the k-NN algorithm separately, one paragraph contains the results when running the MF algorithm separately, and one paragraph contains the results when running the hybrid version of the two. Each of the paragraphs are given "random" case ids and placed in a random order, however which paragraph contains the results from what algorithm can be deduced by looking at the source code of the result page.

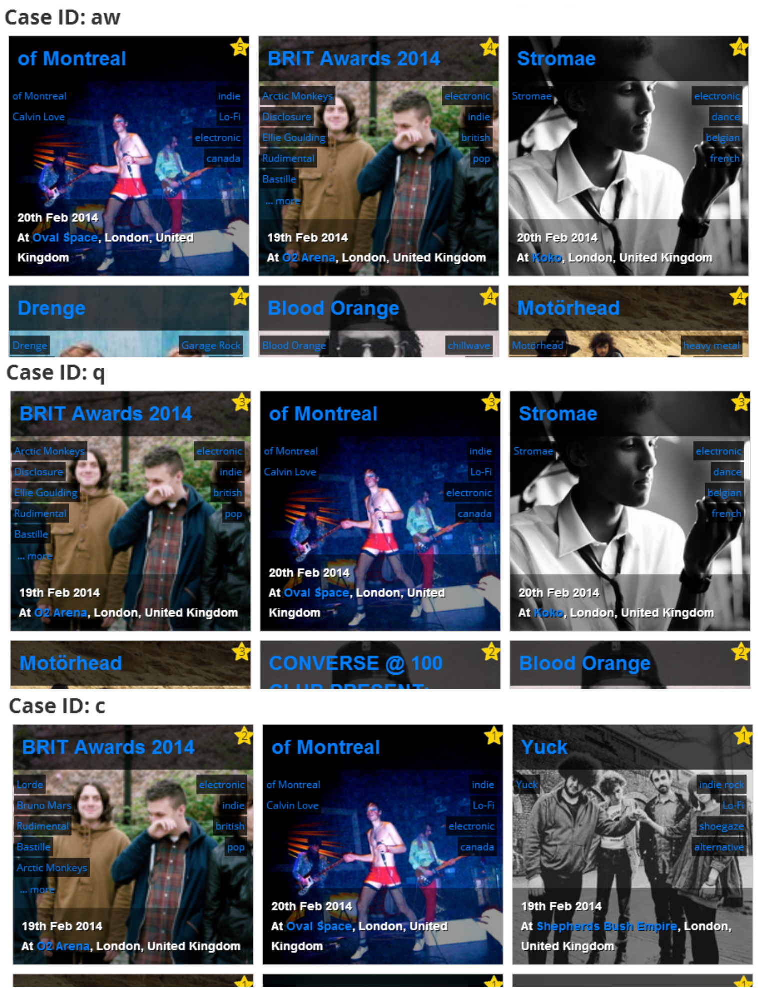


FIGURE 7.1: Result View of prototype for result evaluation

To evaluate the prototype, the steps outlined in Section 3.3.2 were followed. Two groups consisting of 2 and 3 people respectively were asked to find recommendations individually, in a group of two people, and in a group of three people, for two different timespans (18/02/2014-03/03/2014 and 05/03/2014 and 09/07/2014), and two different cities (London and New York). When the second group was asked to find recommendations for a group of 3 people, a user from the first group was added to the recommendation process.

For each step, the participants rated each of the algorithms on how satisfied they were with the recommendations given on a scale from 1-5, where 1 is *Very satisfied* and 5 is *Very dissatisfied*.

7.2.1 Results

The raw data from the Quality Evaluation can be found on Github (see Appendix C.1).

As seen in Table 7.7, the MF algorithm was overall picked as giving the most appealing results 7 out of 40 times, the ***k***NN algorithm 16 out of 40 times, and the hybrid approach in 17 out of 40 cases. Overall, the ***k***NN algorithm received an average rating of 2.28 in the 40 responses, the Hybrid approach 2.38, and the MF algorithm an average of 3.13 as seen in Table 7.8. Table 7.9 shows the average ratings for each of the algorithms when the recommendation process was performed with users that was created with 5 and 10 of their favorite artists respectively. Going from 5 to 10 users resulted that the average rating given to the ***k***-Nearest Neighbor algorithm went from 2.5 to 1.8. For the other algorithms, no significant changes were observed.

As seen in Table 7.10, when going from one person in a group to two and three, the average ratings given for the ***k***NN and Hybrid approaches increased. For the MF approach however, the average ratings decreased slightly from 3.3 for one user to 2.9 for three users in a group. The average ratings for the ***k***NN approach increased from 1.8 to 2.3 for two users and to 2.5 for three users. The average

Algorithm	Number of selections	Percentage
Matrix Factorization	7	17.5%
$\boldsymbol{k}$-Nearest Neighbor	16	40.0%
Hybrid approach	17	42.5%

Table 7.7: Preferred algorithm selection by users

Algorithm	Average rating	Variance	Standard Deviation	n
Matrix Factorization	3.13	0.73	0.85	40
$\boldsymbol{k}$-Nearest Neighbor	2.28	0.92	0.96	40
Hybrid approach	2.38	0.75	0.87	40

Table 7.8: Overall Average statistics per algorithm

ratings for the Hybrid approach increased from 2.3 for one user, and 2.4 for two users to 2.6 for three users. The same can be said for the Variance and Standard Deviation of ratings given to each algorithm. For the $\boldsymbol{k}$NN algorithm, the variance increased from 0.62 for one user in a group to 1.17 for three users in a group, and the standard deviation from 0.78 to 1.08. The variance for the hybrid approach increased from 0.68 to 0.93 and the standard deviation from 0.82 to 0.97.

These results show a clear trend that the $\boldsymbol{k}$NN and the Hybrid approach tend to produce more satisfying recommendations than the MF approach as the average ratings given to the two are generally lower, and they were picked as the favorite algorithms significantly more. An overall average rating of 2.28 and 2.38 out of 5 from the $\boldsymbol{k}$NN and Hybrid approaches respectively, indicates that the participants were reasonably satisfied with the results given, however with some room for improvement.

In general, recommendations given for users created based on 10 of the user's favorite artists, produced more satisfying results than for when 5 artists were used in the user creation process. Going from one user in a group, to two and three users, overall led to lower satisfaction with the recommended concerts.

	Average rating		Variance		Standard Deviation		n
Algorithm	5	10	5	10	5	10	
Matrix Factorization	3.6	3.3	0.27	0.46	0.52	0.67	10
k-Nearest Neighbor	2.5	1.8	0.72	0.62	0.85	0.79	10
Hybrid approach	2.2	2.3	0.62	0.68	0.79	0.82	10

TABLE 7.9: Statistics when 5 and 10 artists were used in user creation

	Average rating			Variance			Standard Deviation		
Algorithm	1 user	2 users	3 users	1 user	2 users	3 users	1 user	2 users	3 users
Matrix Factorization	3.3	2.7	2.9	0.46	1.12	0.77	0.67	1.06	0.88
k-Nearest Neighbor	1.8	2.3	2.5	0.62	1.12	1.17	0.78	1.06	1.08
Hybrid approach	2.3	2.4	2.6	0.68	0.93	0.93	0.82	0.97	0.97

TABLE 7.10: Statistics when recommendations were given for groups consisting of 1, 2 and 3 users respectively

7.3 Discussion

7.3.1 Threat to Validity

The QE was performed with two groups of 2 and 3 people. This low number of participants means that each participant had a significant impact on the results. The statistics produced when a user was created with 5 and 10 favorite artists (Table 7.9), were based on $n = 10$ samplings each; answers for questions Q1 and Q3; and Q5 and Q7 respectively.

The same can be said with the statistics produced for the results with varying group sizes (Table 7.10). They were based on $n = 10$ samplings each; answers for questions Q5 and Q7 for groups with one user; Q9 and Q11 for groups with two users; and Q13 and Q15 for groups with three users. The overall statistics (Table 7.8) was created based on answers to questions Q1, Q3, Q5, Q7, Q9, Q11, Q13 and Q15 which for 5 users lead to a sample size of $n = 40$.

By looking at the top social tags used for the artists each of the users registered as seen in Table 7.11, it is apparent that the users' taste in music are quite different as they share few top tags amongst them (except for User 3 and 4). However,

User 1	User 2	User 3	User 4	User 5
Rock	Pop	Electronic	Electronic	Rock
Progressive metal	Hip-Hop	Electro house	House	Classic rock
Metal	Rnb	House	Dance	Classical
Alternative rock	dance	Electro	Progressive House	Instrumental
Alternative	Rock	Dubstep	Electro	Violin
Progressive rock	Electronic	Dance	Electro House	Hard rock
Sludge	Alternative	Electronica	Dubstep	Pop

TABLE 7.11: Top tags for the participating users

because of the low number of users and sample sizes, even with this diversity, it can't be said that these five users are representative for the whole potential user base, and therefore, further testing should be performed to measure the Quality of Recommendations created by the prototype.

Even though more testing is needed, there still is a strong indication that the kNN and Hybrid approaches perform better than the MF approach as suggested in Section 7.2.1 with a sample size of $n = 40$. Similarly, it can be said that the five users testing the prototype were reasonably happy with the results.

7.3.2 Novelty and Serendipity in Concert Recommendation Systems

In section 2.1.6.3, there was put an emphasis on that a recommendation system should provide novel and serendipitous recommendations. The emphasis should be put on the lesser known artists, the *long tail* of the listen count curve. However, during the development and testing of this prototype, it was observed that a full focus on this may not be the best approach for a CRS. People tend to prefer to go to concerts with artists they are already familiar with. The concert scene might not be the place were people try to be adventurous and discover new music. It is easier, more convenient, and cheaper to discover and becoming familiar with new artists first, before deciding to attend a concert with them. This might be one of the causes in why the kNN and Hybrid approaches received better ratings from the test users when it came to quality of recommendations, as CF approaches tend

***k*NN**		**MF**	
Artist	# of listeners	Artist	# of listeners
Avicii	548	Arctic Monkeys	2388
Katy Perry	676	Lorde	554
Arctic Monkeys	2388	Beyoncé	585
Disclosure	535	Metronomy	418
Kanye West	1578	Cut Copy	378
Nine Inch Nails	1270	Alkaline Trio	383
The National	1687	Panic! at the Disco	
Drake	712	Slowdive	308
Interpol	784	Katy Perry	676
Arcade Fire	2165	Pretty Lights	234
Average	**1234.3**	**Average**	**632**

Table 7.12: Number of listeners for the top artist playing at the top 10 concerts between 18/02/2014 and 17/07/2014 in London for user *simensma*

to have a popularity bias causing the more popular artists to be recommended (see Section 2.1.6.4). An example of this can be seen in Table 7.12, where the top artist and how many users have listened to them for the 10 top concerts recommended for the user *simensma* in London between 18/02/2014 and 17/07/2014, can be seen. The 5 most frequently used tags to describe *simensma*'s top artists are *electronic*,*house*,*dance*,*indie*, and *electro house*. On average, 1234 users had listened to each of the artists recommended by the ***k***NN algorithm whereas 632 users on average had listened to each of the artists recommended by the MF algorithm.

Chapter 8

Conclusion and Further Work

8.1 Further work

From this thesis, many directions for further research are possible. The results clearly shows that the MF approach underperforms compared to the $\boldsymbol{k}$NN and Hybrid approach, a further investigation into why this happens can be done. By extending the framework provided, other algorithms for recommendation systems can be implemented and compared to the existing ones to see how they perform. Another way to go from here is to have a look at the context aware part of the application. Is there any benefit in making relaxation of context an implicit part of the algorithm instead of something performed by the user explicitly? How would other context variables, such as listen recency affect the satisfactions when recommending concerts? In Section 7.1.5, some usability problems and suggestions to new functionality for the prototype were identified, and can be a good starting point when utilizing the framework provided.

8.2 Conclusion

In this thesis, a prototype of a Context Aware group Recommendation System for Concerts was presented. The prototype implemented three different algorithms, a Matrix Factorization algorithm, a k-Nearest Neighbor algorithm and a Hybrid approach of the two. The goal for the thesis, was to improve the usability and quality of recommendations given by the prototype implemented during the author's semester project fall 2013 [1].

The usability of the prototype was evaluated using the System Usability Scale (SUS) and an Application Specific Survey (AS). 15 people were asked to undertake these surveys. In total, the prototype got a SUS score of 79.83 which is a good indication on that the users found the usability of the prototype satisfactory. However, the comments from the free text answers shows that there still are room for improvements.

The AS mainly focused on the usability of the Context relaxation part of the prototype, to find out if it was easy to find concerts close to the parameters specified when it comes to time and location. The results from the AS showed that the users in general were satisfied with how this process worked.

The goal for this prototype was to recommend concerts to a user within the location and timespan given that the user could be interested in attending. To evaluate how well this was achieved, a Recommendation Quality Evaluation(QE) was undertaken with two groups consisting of 2 and 3 people respectively. Through a range of scenarios, the groups were told to find recommendations for the dates and location asked about, and for each algorithm, rate how satisfied they were with the results. The results from the QE showed that the users generally were satisfied with the kNN implementation and the Hybrid approach, whereas they were less satisfied with the MF approach. The QE was also undertaken to see how different group sizes affected the quality of recommendations. The results showed that the users became less satisfied when the number of members in the group increased from one to two and three respectively, which is to be expected as different preferences has to be taken into account in larger groups. However, the QE was only performed with five participants, so there is a need for an evaluation with more participants to able to draw any final conclusions.

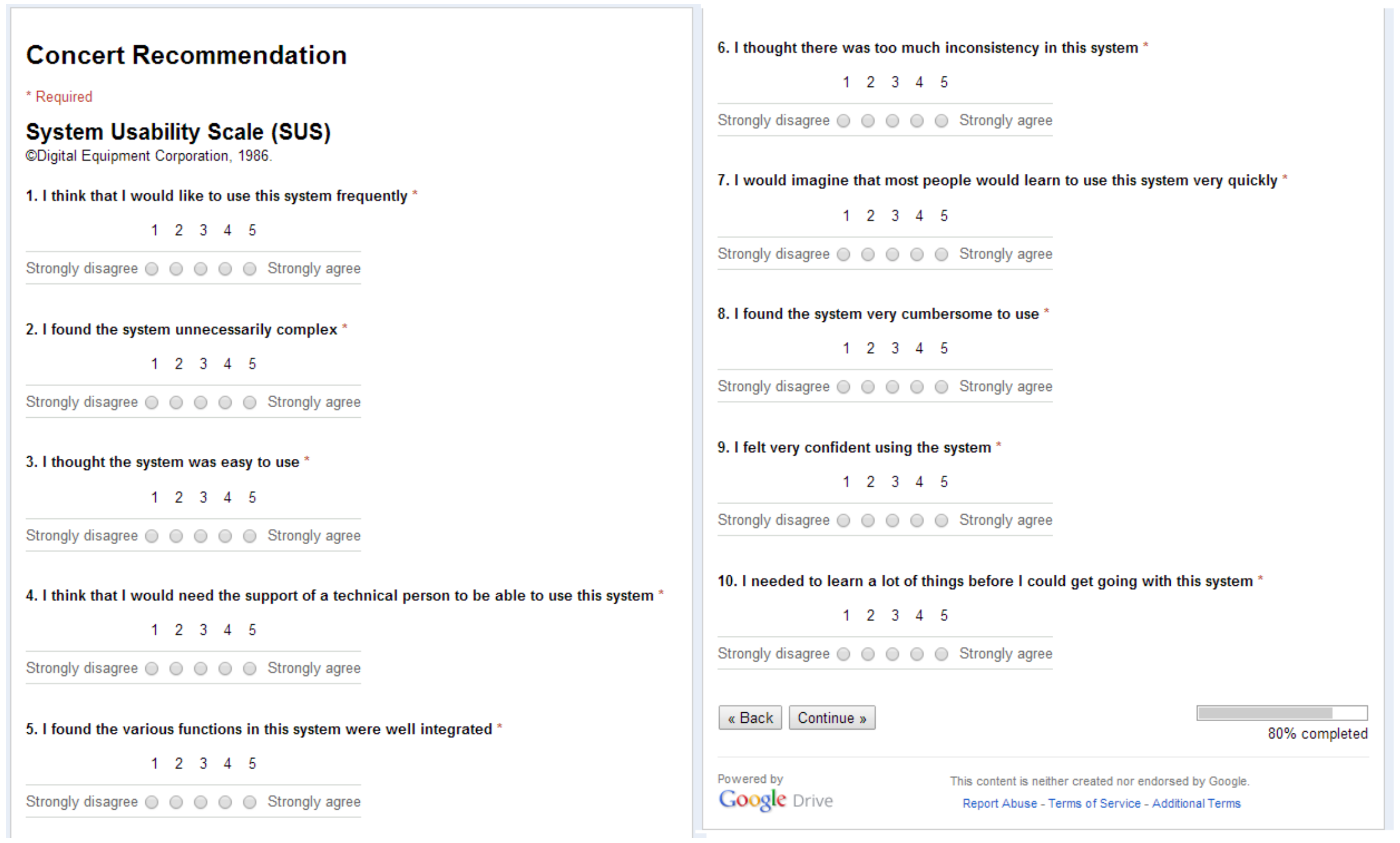
All in all, these results show that well known methods for recommendation systems can be applied successfully to a Context Aware Recommendation System for Concerts.

Appendix A

Usability Survey

Section A.1 contains the SUS schema used for usability evaluation. The AS survey used in the same evaluation can be found in section A.2. The survey used to gather background information on the subjects can be found in section A.3. Section A.4 contains the free-text answers gathered from questions AS5, AS6 and AS7.

A.1 System Usability Scale (SUS)



Concert Recommendation

* Required

System Usability Scale (SUS)
©Digital Equipment Corporation, 1986.

1. I think that I would like to use this system frequently *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

2. I found the system unnecessarily complex *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

3. I thought the system was easy to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

4. I think that I would need the support of a technical person to be able to use this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

5. I found the various functions in this system were well integrated *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

6. I thought there was too much inconsistency in this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

7. I would imagine that most people would learn to use this system very quickly *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

8. I found the system very cumbersome to use *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

9. I felt very confident using the system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

10. I needed to learn a lot of things before I could get going with this system *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

« Back Continue »

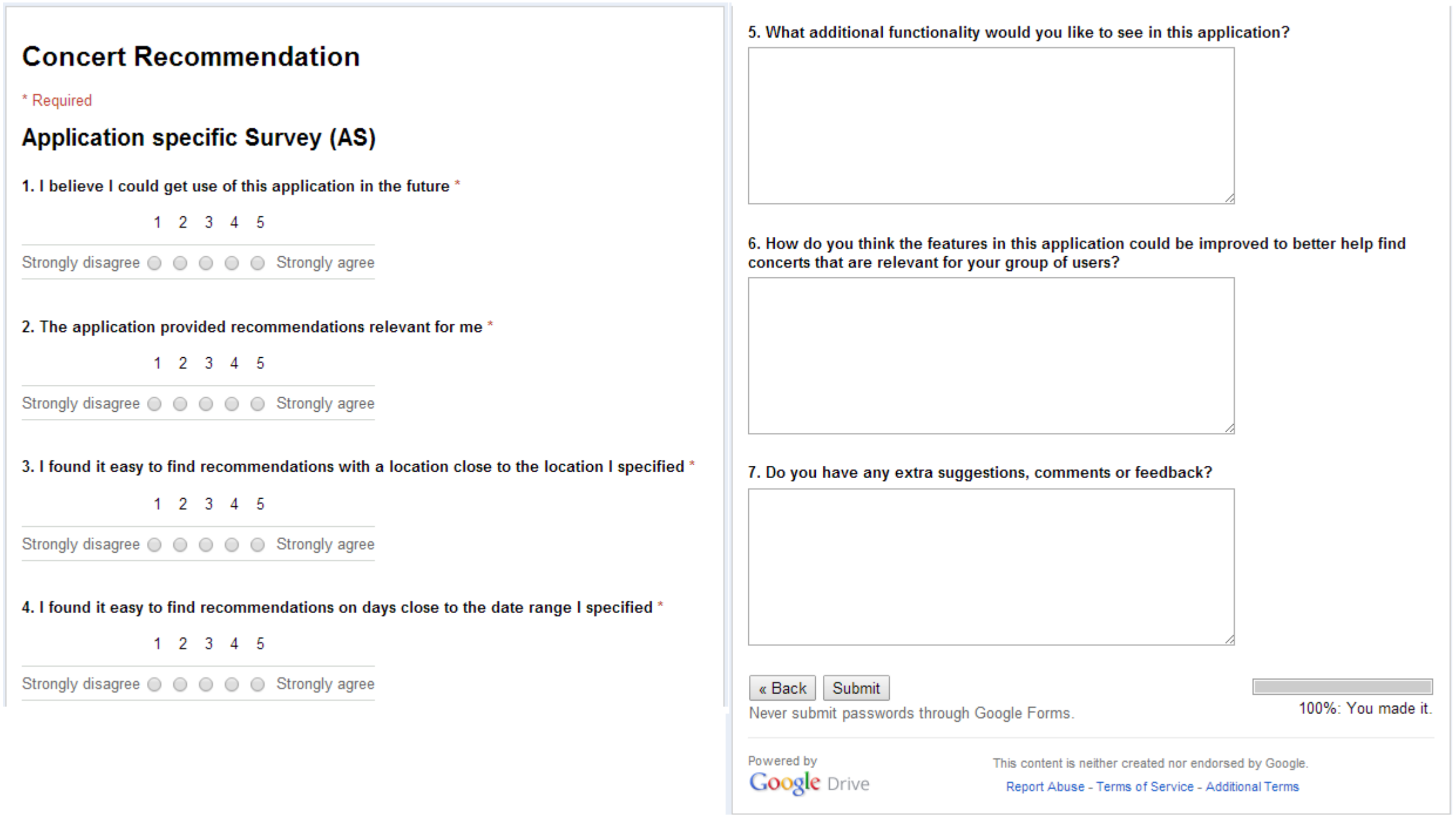
80% completed

Powered by Google Drive

This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure A.1: System Usability Scale

A.2 Aplication Specific Survey (AS)



Concert Recommendation

* Required

Application specific Survey (AS)

1. I believe I could get use of this application in the future *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

2. The application provided recommendations relevant for me *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

3. I found it easy to find recommendations with a location close to the location I specified *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

4. I found it easy to find recommendations on days close to the date range I specified *

	1	2	3	4	5	
Strongly disagree	○	○	○	○	○	Strongly agree

5. What additional functionality would you like to see in this application?

6. How do you think the features in this application could be improved to better help find concerts that are relevant for your group of users?

7. Do you have any extra suggestions, comments or feedback?

Never submit passwords through Google Forms.

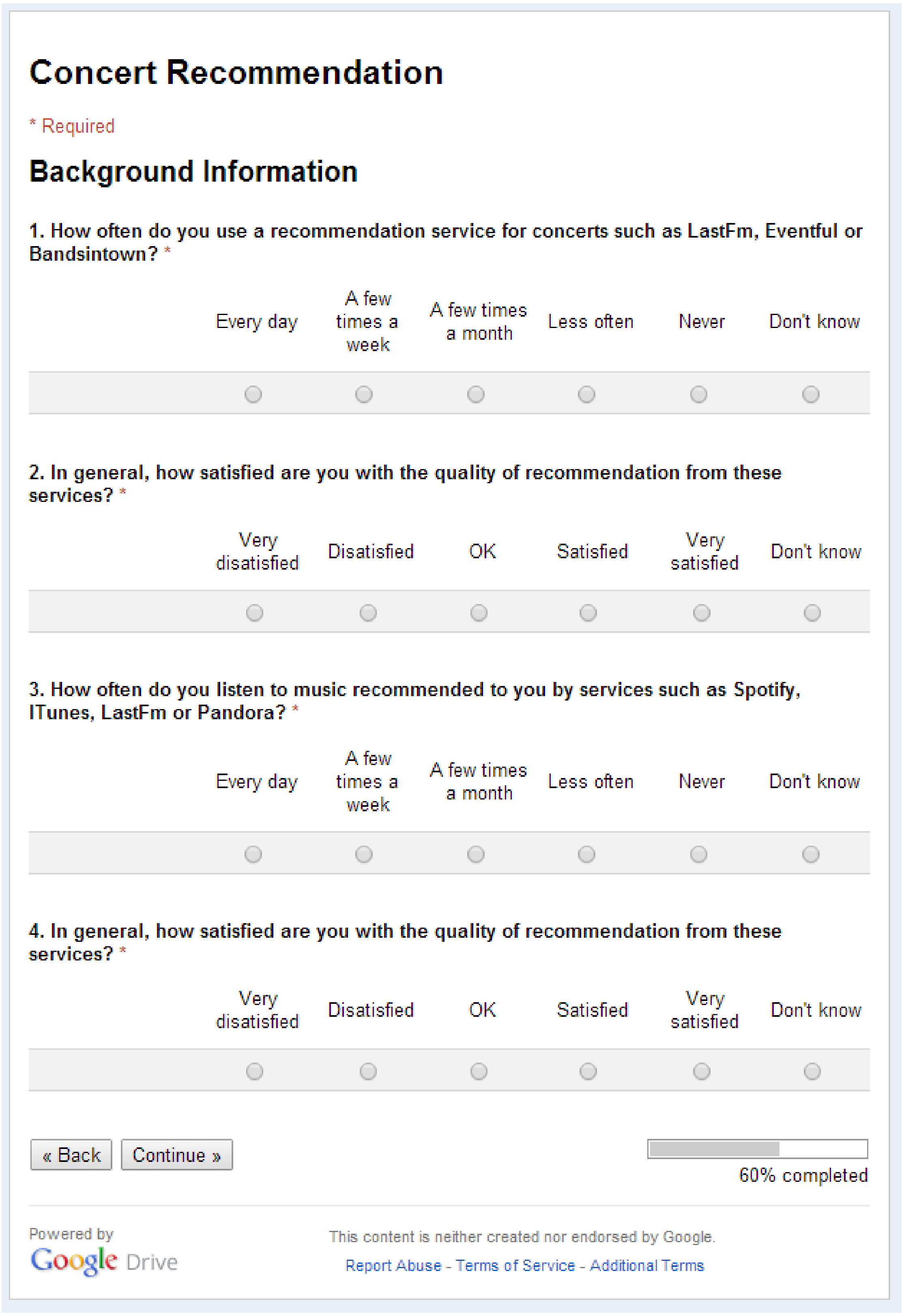
Powered by Google Drive

This content is neither created nor endorsed by Google.

Report Abuse - Terms of Service - Additional Terms

FIGURE A.2: Application specific survey

A.3 Background information (BI)



Concert Recommendation

* Required

Background Information

1. How often do you use a recommendation service for concerts such as LastFm, Eventful or Bandsintown? *

	Every day	A few times a week	A few times a month	Less often	Never	Don't know
	○	○	○	○	○	○

2. In general, how satisfied are you with the quality of recommendation from these services? *

	Very disatisfied	Disatisfied	OK	Satisfied	Very satisfied	Don't know
	○	○	○	○	○	○

3. How often do you listen to music recommended to you by services such as Spotify, ITunes, LastFm or Pandora? *

	Every day	A few times a week	A few times a month	Less often	Never	Don't know
	○	○	○	○	○	○

4. In general, how satisfied are you with the quality of recommendation from these services? *

	Very disatisfied	Disatisfied	OK	Satisfied	Very satisfied	Don't know
	○	○	○	○	○	○

« Back
Continue »
60% completed
Powered by Google Drive
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure A.3: User background information

A.4 Feedback

AS5. What additional functionality would you like to see in this application?
”When clicking on a concert, I would like to see the date of the concert. I would also like to see website and a google map of the location. It should connect your user to your user on applications such as last.fm, instead of having to create a new user if I already have one on a different site.”
Seeing ticket prices or adjusting the search for ticket prices
”Ability to choose genres. Also some information about what it means when you add a person to the group. I felt the information about the features were lacking in general. Explain what users you are adding from what service.”
User preferences imported directly from my last.fm or Spotify account.
Where to find concerts sitt favourite artists not included in the reconnendation.
”En måte å endre brukerens band/artistforslag.Edit user rett og slett ;) Og det burde kanskje bli lagt til flere locations å velge mellom.” (Translated as: A way to change a users group/artist recommendation. Simply put edit user functionality :) Also, maybe more locations to choose from should be available)
I would like to be able to see all the concerts in the specifide location and dates
show genres for concerts. Also an overview of all concerts ata place
It would be nice to be able to filter concerts based on genre

Table A.1: AS5 comments

AS6. How do you think the features in this application could be improved to better help find concerts that are relevant for your group of users?
Some people live in places where there’s not that many options for concerts. Maybe have an option that lists all concerts within an area on a given date(s)?
Not sure :P
”Maybe ability to find ones friends through facebook or the underlaying user base. Also, maybe the ability to exclude some results or genres. Say ””hide”” or ””I don’t wan’t to go to this concert””.”
1. Possibly include some hint of genre in the initial concert overview. Though I hadn’t heard about several of the artists before, I might still be interested. Having an idea of the genre in the initial overview would allow me to choose which concerts to consider more carefully, even when I don’t know the band.
”Question like: If you like .....you might like......”

Table A.2: AS6 comments

AS7. Do you have any extra suggestions, comments or feedback?
Looks good, and the developer is handsome.
”there are severe delays when getting recommendations.
the recommendations did not show any performers known to me, and can therefore not be particularly relevant.”
I love you <3 :)
”Maybe change the color of the text. Blue is not very good for reading. Make it a lighter shade of blue.
Very cool animations and a cool project in general! :)”
It’s a bit scary leaving the group setup view in order to register new users. If it was somehow emphasized that I would not lose my current state I would feel more comfortable.
Use clear and easy messages in the startup.
I generally think music recommendation are a quite waste of time. It is hard to predict what people like. For instance, one person may like one song of an artist on a album, while it does not like another song on the same album. So it is extremely hard for computers to know such things.
It looks nice!

Table A.3: AS7 comments

Appendix B

Code

All the code produced while developing the prototype presented in this thesis is available at https://github.com/simensma/GroupRec. It is released under a GNU General Public License, version 2[1]. Due to copyright restrictions, the dataset used for evaluation purposes is not part of the distributed code. However code to reconstruct a similar one is provided.

Section B.1 contains the code used for calculation of features for the MF algorithm. Section B.2 shows how ratings are predicted for an artist.

[1]http://www.gnu.org/licenses/gpl-2.0.html

B.1 Matrix factorization

```java
public void calculateFeatures(int maxFeatures, int maxIterations,
        double lambda, double gamma, double minImprov) {

    double rmse = 2.0;
    for (int feature = 0; feature < maxFeatures; feature++) {
        double rmse_last = Double.POSITIVE_INFINITY;

        for (int iter = 0; (iter < maxIterations)
            || (rmse <= rmse_last - minImprov); iter++) {
            double squareError = 0;
            rmse_last = rmse;

            for (int i = 0; i < ratings.length; i++) {
                ArtistlisteningWrapper rating = ratings[i];

                int artistId = rating.getArtistId();
                int userId = rating.getUserId();

                double predictedRating = predictRating(artistId, userId,
                        feature, maxFeatures, rating.getCache(),true);
                double error = (1.0 * rating.getListenCount() - predictedRating);
                squareError += Math.pow(error, 2);

                userFeatures[feature][userId] += gamma
                        * (error * artistFeatures[feature][artistId]
                        - lambda * userFeatures[feature][userId]);
                artistFeatures[feature][artistId] += gamma
                        * (error * userFeatures[feature][userId]
                        - lambda * artistFeatures[feature][artistId]);
            }

            rmse = Math.sqrt(squareError / ratings.length);
        }
        for (int i = 0; i < ratings.length; i++) {
            ratings[i].setCache(predictRating(ratings[i].getArtistId(),
                    ratings[i].getUserId(), feature, maxFeatures,
                    ratings[i].getCache(), false));
        }
    }
}
```

Listing 13: Calculating latent features (based on Development [45])

B.2 Artist rating prediction with k-Nearest Neighbor

```java
/**
 * artistListenings: Map of list of tuples that contains (userId, rating)
 * pairs for the artists that should get a predicted rating.
 *
 * userSimilarities: Contains similarities (Double) between all the users
 * (Integer) that have listened to some of the artists found and the user
 * being processed.
 *
 * artists: Artists ids that a rating should be predicted for
 *
 * returns: Predicted ratings for the artists being processed.
 */
protected Map<Integer, Double> calculateArtistScores(
        Map<Integer, List<Tuple<Integer, Double>>> artistListenings,
        Map<Integer, Double> userSimilarities, List<Integer> artists, int k) {

    Map<Integer, Double> artistScores = new HashMap<Integer, Double>();
    for (Integer artistId : artists) {
        /*
         * List of tuples, (userId, listeningFrequency), which describes the
         * listening frequency to the artist being processed for all the
         * users that have listened to that artist.
         */
        List<Tuple<Integer, Double>> listens = artistListenings
                .get(artistId);

        // Find the k most similar users
        k = (int) Math.sqrt(listens.size());

        Collections.sort(listens, new Comparator<Tuple<Integer, Double>>() {

            @Override
            public int compare(Tuple<Integer, Double> t1,
                    Tuple<Integer, Double> t2) {

                return t2.getY().compareTo(t1.getY());
            }
        });

        listens = listens.subList(0, k);

        double sum = 0;
        double totSim = 0;
        for (Tuple<Integer, Double> tuple : listens) {
            totSim += Math.pow(userSimilarities.get(tuple.getX()), 2);
        }

        // Calculate a rating for artistId
        // Each user in listens contributes to the final rating with
        // rating*sim/totSim
        for (Tuple<Integer, Double> tuple : listens) {

            double sim = Math.pow(userSimilarities.get(tuple.getX()), 2);
            double rating = tuple.getY();
            sum += rating * sim / totSim;

        }

        artistScores.put(artistId, sum);
    }
    return artistScores;
}
```

Listing 14: kNN calculation of artist score

Appendix C

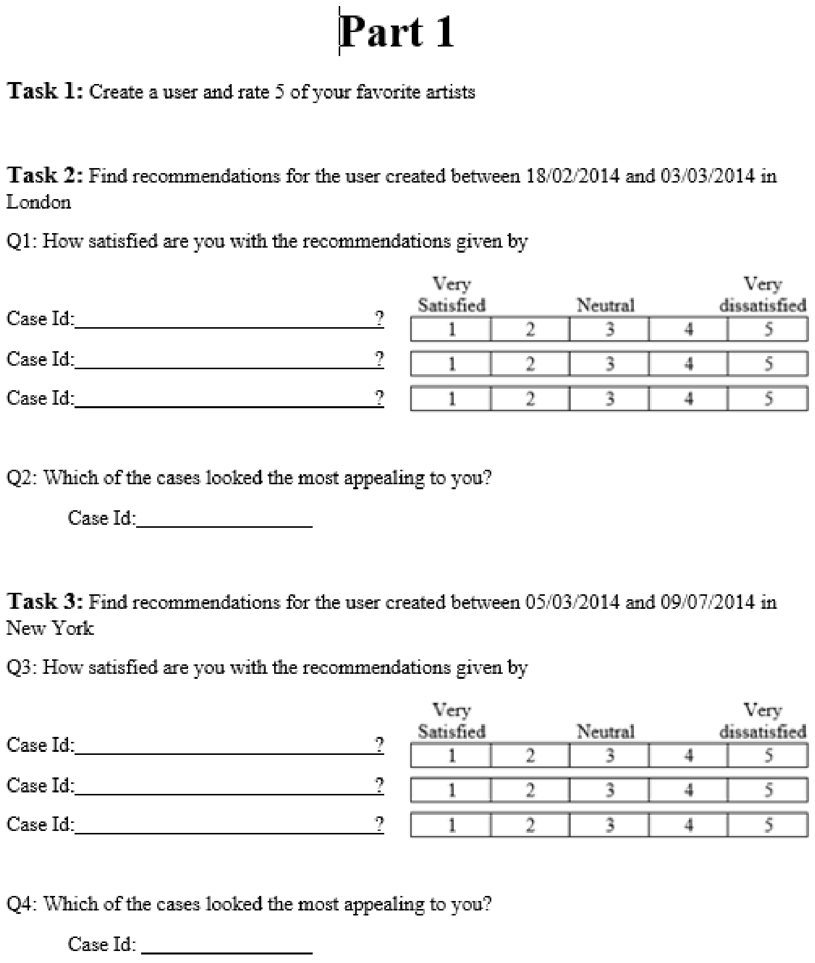
Quality Survey

Section C.2 contains the 4 schemas used for the Quality Survey.

C.1 Results

The raw results from the Quality Evaluation can be found in this projects Github repository, at `https://github.com/simensma/GroupRec/blob/master/evaluation/Evaluation.xlsx`.

C.2 Quality Survey



Part 1

Task 1: Create a user and rate 5 of your favorite artists

Task 2: Find recommendations for the user created between 18/02/2014 and 03/03/2014 in London

Q1: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:________________?	1	2	3	4	5
Case Id:________________?	1	2	3	4	5
Case Id:________________?	1	2	3	4	5

Q2: Which of the cases looked the most appealing to you?

Case Id:____________

Task 3: Find recommendations for the user created between 05/03/2014 and 09/07/2014 in New York

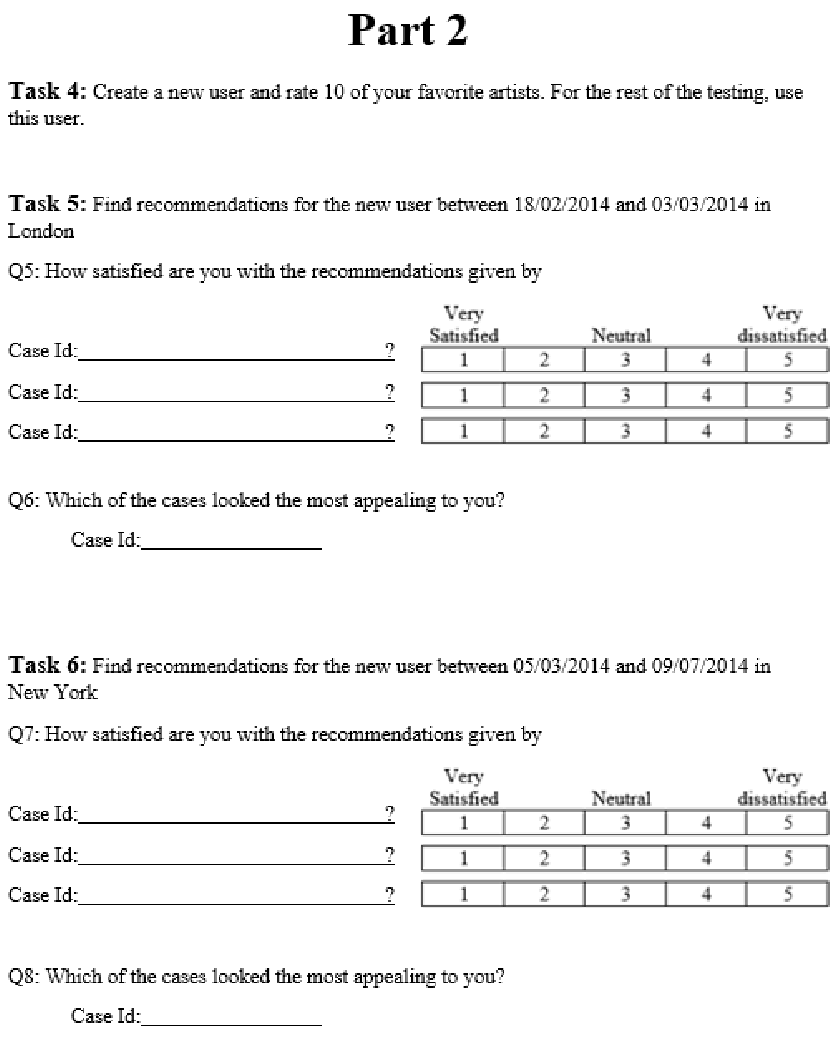
Q3: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:________________?	1	2	3	4	5
Case Id:________________?	1	2	3	4	5
Case Id:________________?	1	2	3	4	5

Q4: Which of the cases looked the most appealing to you?

Case Id: ____________

Figure C.1: System Quality Survey Schema part 1



Part 2

Task 4: Create a new user and rate 10 of your favorite artists. For the rest of the testing, use this user.

Task 5: Find recommendations for the new user between 18/02/2014 and 03/03/2014 in London

Q5: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5

Q6: Which of the cases looked the most appealing to you?

Case Id:____________

Task 6: Find recommendations for the new user between 05/03/2014 and 09/07/2014 in New York

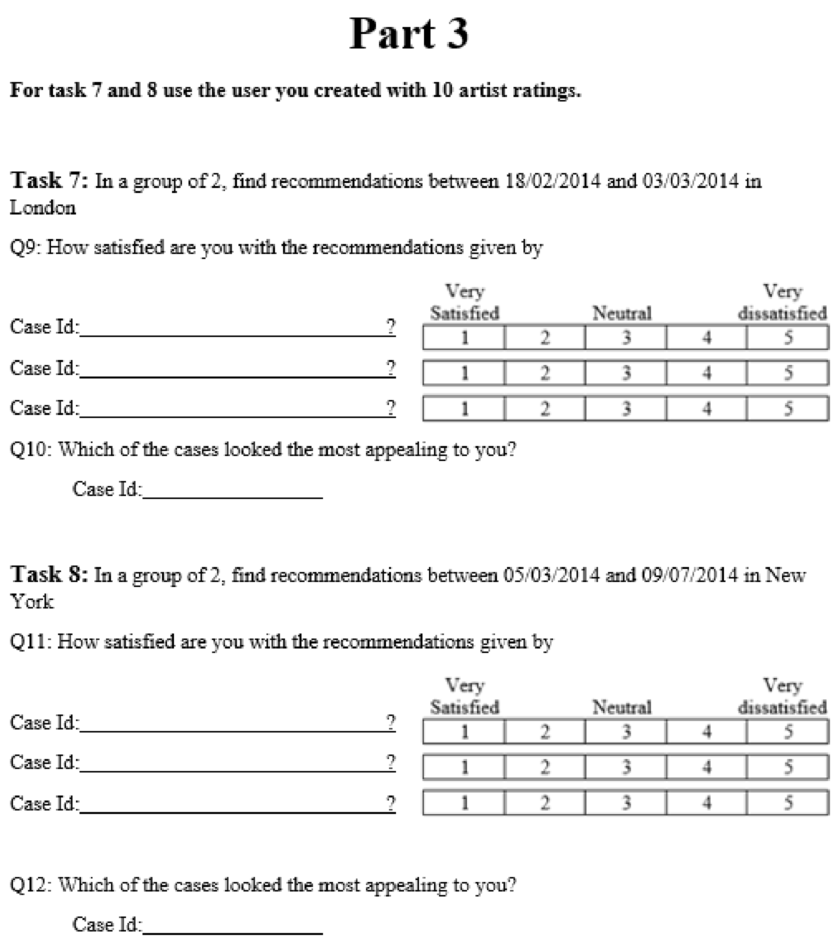
Q7: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5

Q8: Which of the cases looked the most appealing to you?

Case Id:____________

FIGURE C.2: System Quality Survey Schema part 2



Part 3

For task 7 and 8 use the user you created with 10 artist ratings.

Task 7: In a group of 2, find recommendations between 18/02/2014 and 03/03/2014 in London

Q9: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:______________________?	1	2	3	4	5
Case Id:______________________?	1	2	3	4	5
Case Id:______________________?	1	2	3	4	5

Q10: Which of the cases looked the most appealing to you?

Case Id:____________

Task 8: In a group of 2, find recommendations between 05/03/2014 and 09/07/2014 in New York

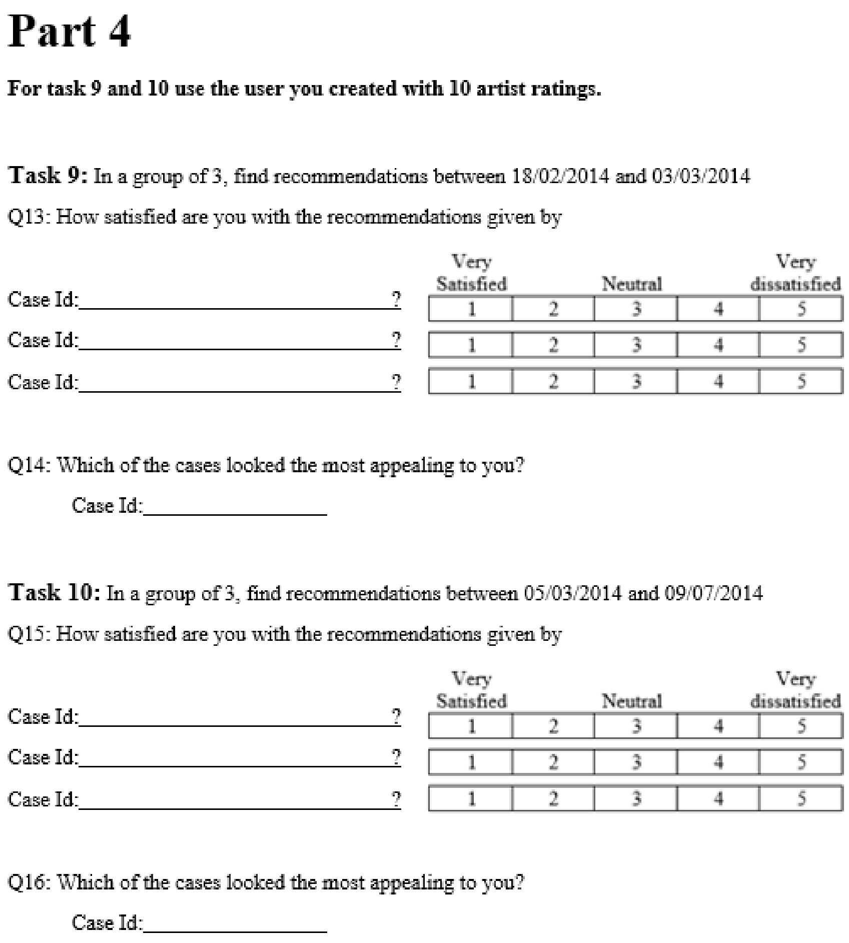
Q11: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:______________________?	1	2	3	4	5
Case Id:______________________?	1	2	3	4	5
Case Id:______________________?	1	2	3	4	5

Q12: Which of the cases looked the most appealing to you?

Case Id:____________

Figure C.3: System Quality Survey Schema part 3



Part 4

For task 9 and 10 use the user you created with 10 artist ratings.

Task 9: In a group of 3, find recommendations between 18/02/2014 and 03/03/2014

Q13: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5

Q14: Which of the cases looked the most appealing to you?

Case Id:____________

Task 10: In a group of 3, find recommendations between 05/03/2014 and 09/07/2014

Q15: How satisfied are you with the recommendations given by

	Very Satisfied		Neutral		Very dissatisfied
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5
Case Id:____________________?	1	2	3	4	5

Q16: Which of the cases looked the most appealing to you?

Case Id:____________

Figure C.4: System Quality Survey Schema part 4

Bibliography

[1] Simen F Smaaberg. A context aware group recommendation system for concerts. Specialization project in Computer Science, Department of Computer and Information Science, Norwegian University of Science and Technology, 2013.

[2] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

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

[4] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[5] 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*, pages 285–295. ACM, 2001.

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

[7] ON Osmanli and IH Toroslu. Using tag similarity in svd-based recommendation systems. In *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, pages 1–4. IEEE, 2011.

[8] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

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

[10] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[11] Mary Bazire and Patrick Brézillon. Understanding context before using it. In *Modeling and using context*, pages 29–40. Springer, 2005.

[12] Umberto Panniello, Alexander Tuzhilin, Michele Gorgoglione, Cosimo Palmisano, and Anto Pedone. Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 265–268. ACM, 2009.

[13] Mark Van Setten, Stanislav Pokraev, and Johan Koolwaaij. Context-aware recommendations in the mobile tourist application compass. In *Adaptive hypermedia and adaptive web-based systems*, pages 235–244. Springer, 2004.

[14] Kostas Stefanidis, Nafiseh Shabib, Kjetil Nørvåg, and John Krogstie. Contextual recommendations for groups. In *Advances in Conceptual Modeling*, pages 89–97. Springer, 2012.

[15] Jing Wang, Hui Li, and Hui Zhao. The contextual group recommendation. In *Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on*, pages 127–131. IEEE, 2013.

[16] 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, pages 1097–1101. ACM, 2006.

[17] Oscar Celma. *Music recommendation and discovery: The long tail, long fail, and long play in the digital music space.* Springer, 2010.

[18] Maciej Pacula. A matrix factorization algorithm for music recommendation using implicit user feedback. 2009.

[19] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, 2009.

[20] Judith Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction*, 14(1): 37–85, 2004.

[21] Brian Whitman. How music recommendation works — and doesn't work. `http://notes.variogr.am/post/37675885491/how-music-recommendation-works-and-doesnt-work`, 2013. Accessed: 2013-04-27.

[22] Jean-Julien Aucouturier and Francois Pachet. Music similarity measures: What's the use? In *ISMIR*, 2002.

[23] Patrik N Juslin and John A Sloboda. *Music and emotion: Theory and research.* Oxford University Press, 2001.

[24] Lie Lu, Dan Liu, and Hong-Jiang Zhang. Automatic mood detection and tracking of music audio signals. *IEEE Transactions on audio, speech, and language processing*, 14(1):5–18, 2006.

[25] Yazhong Feng, Yueting Zhuang, and Yunhe Pan. Music information retrieval by detecting mood via computational media aesthetics. In *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, pages 235–241. IEEE, 2003.

[26] Diyi Yang, Tianqi Chen, Weinan Zhang, Qiuxia Lu, and Yong Yu. Local implicit feedback mining for music recommendation. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 91–98. ACM, 2012.

[27] David Jennings. *Net, blogs and rock'n'roll: how digital discovery works and what it means for consumers, creators and culture.* Nicholas Brealey Publishing, 2007.

[28] Alan Hevner and Samir Chatterjee. *Design research in information systems: theory and practice*, volume 22. Springer, 2010.

[29] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[30] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:194, 1996.

[31] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[32] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[33] Kalevi Kilkki. A practical model for analyzing long tails. *First Monday*, 12 (5), 2007.

[34] Simon Funk. Netflix update: Try this at home. `http://sifter.org/simon/journal/20061211`, 2006. Accessed: 2014-03-15.

[35] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *Computer Vision—ECCV 2002*, pages 707–720. Springer, 2002.

[36] Gruntjs. Grunt: The javascript task runner. `http://gruntjs.com/`, 2014. Accessed: 2014-02-30.

[37] John Gossman. Introduction to model/view/viewmodel pattern for building wpf apps. `http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx`, 2005. Accessed: 2014-04-02.

[38] Addy Osmani. Understanding mvvm, a guide for javascript developers. `http://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/`, 2012. Accessed: 2014-03-02.

[39] Blue Spire. Single page apps done right. `http://durandaljs.com/`, 2014. Accessed: 2014-03-02.

[40] RequireJS. Require.js, a javascript module loader. `http://requirejs.org/`, 2014. Accessed: 2014-03-02.

[41] Knockoutjs.com. Simplyfy dynamic javascript uis with the model-view-view model (mvvm) pattern. `http://knockoutjs.com/`, 2014. Accessed: 2014-02-30.

[42] Knockoutjs.com. Knockout: Introduction. `http://knockoutjs.com/documentation/introduction.html`, 2014. Accessed: 2014-03-02.

[43] Roger L Costello. Building web services the rest way. `http://www.xfront.com/REST-Web-Services.html.UltimaConsulta`, 2007. Accessed: 2014-04-02.

[44] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, 2000.

[45] Timely Development. Netflix price. `http://www.timelydevelopment.com/demos/NetflixPrize.aspx`, 2007. Accessed: 2014-04-15.

[46] Christopher A Cassa, Karin Iancu, Karen L Olson, and Kenneth D Mandl. A software tool for creating simulated outbreaks to benchmark surveillance systems. *BMC Medical Informatics and Decision Making*, 5(1):22, 2005.

[47] Jakob Nielsen. *Usability engineering.* Elsevier, 1994.

[48] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3):379–383, 2003.