A Context Aware Group Recommendation System for Concerts

Simen Fivelstad Smaaberg

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

As a group of friends going to a festival, deciding what concerts to attend can be difficult as everyone have their own preferences and wishes. In this project we present a group recommendation system for concerts that takes a user’s listening habits into account and is context aware in the sense that it is aware of a concerts time and location. We provide a prototype that use a K-nearest-neighbour collaborative filtering approach on a constructed dataset of artists, users and their listening habits. The results after a scenario based evaluation show that significant challenges exist when using a pure user based collaborative filtering approach to provide concert recommendations for a group of people. Therefore further investigation is needed to find solutions to these problems.
Contents

1 Introduction 3

2 State of the art 5
   2.1 Recommendation systems 5
      2.1.1 Content based filtering 5
      2.1.2 Collaborative filtering 5
      2.1.3 Hybrid approaches 6
      2.1.4 Social media filtering 6
   2.2 Group recommendation systems 7
   2.3 Context awareness 8
   2.4 Related work 9

3 Requirements 11

4 Design 12
   4.1 Define context 12
   4.2 Filter dataset 13
   4.3 Relaxing of Context 14
   4.4 Individual recommendations 14
   4.5 Aggregate Individual preferences into one recommendation list 16

5 Implementation 17
   5.1 Dataset 17
   5.2 Environment 20
   5.3 Algorithm 21
      5.3.1 Offline calculation 21
      5.3.2 Online calculation 21

6 Results 29
   6.1 Discussion 35

7 Conclusions and future work 41
1. Introduction

We live in an online world where the demand for, and availability of online content is ever increasing and navigating it can be difficult. Making it easier for users to find content they are interested in is therefore important for any internet actor wanting to increase their customers’ satisfaction. One method that may be used to accomplish this is to create a recommendation system for their content that provides recommendations for items the user might find interesting. Recommendation systems have been developed for everything from movies, music and books to news and vacations and are used heavily by big companies such as Amazon, IMDb and Facebook to provide relevant recommendations for their users. Also recommendation systems for music have increasingly over the past years been introduced in and become an integral part of online music services such as Spotify\(^1\), last.fm\(^2\) and iTunes\(^3\). Users will in these applications be provided with song, album and artist recommendations based on their previous listening habits and user profiles.

Another area that recommendation systems have been applied to is recommendations for groups of people instead of the single user you will find in a traditional recommendation system. In many situations, decisions within a group of people have to be taken. For example when a group of friends want to watch a movie or go to a concert, finding one that everyone is happy with might be difficult. Systems that provide recommendations in such situations have been developed, e.g. for movies \(^{18}\), tourist activities \(^9\) and restaurants \(^{14}\).

The group part adds significant complexity to the recommendation system. Examples of additional questions one could ask when it comes to a group recommendation system includes: How is the group formed? Are the users a random group of people? Or is the group predefined? How should the special properties that comes with the forming of a group (for example group dynamics) be handled? For example should a person with much knowledge of the domain area have a bigger influence on the result than the others? How can we find and combine each of the users’ preferences to provide recommendations for the group as a whole?

Group recommendation systems for music aren’t as widespread as such

---

systems for a single user, however research on them exist. McCarthy et al. [15] deployed a prototype at a training center, choosing songs to play based on the preferences of the users currently training. Chao et al. [6] presents an adaptive radio that plays music in a shared environment using negative preferences of users. Zhiwen et al. [28] presents an adaptive In-Vehicle multimedia recommender for groups of people. To date not much work have been done on how such group recommendation systems can be used to recommend concerts where additional context such as the location and time of the concert must be taken into account.

The objective of this research is to modify and apply known algorithms and techniques for group recommendation systems to create a framework that can provide concert recommendations for groups of people.

This report starts with a state of the art in section 2 describing different types of recommendation systems, group recommendation systems and context awareness, in addition to a look into what the most known music recommendation services. Section 3 outlines a scenario for the use of a concert recommendation system and from it extracts requirements for such a system. Section 4 propose an algorithm that can be used to provide recommendations of concerts based on well-known methods for recommendation systems. In section 5 a prototype is presented implementing this algorithm in addition to a constructed dataset. In section 6 the prototype is evaluated using a set of scenarios and the outcome of these discussed.
2. State of the art

2.1. Recommendation systems

A recommendation system can be described as a system that outputs the top \( N \) items out of a set of items that best fits with a user’s preferences. Usually the preferences given are in the form of ratings of a subset of the items in question. Two approaches much used for recommendation systems are content based filtering and collaborative filtering.

2.1.1. Content based filtering

In content based filtering, a user profile describing what types of items a user likes and descriptions of items are used to provide recommendations [21]. Items are compared to the user profile to find suitable ones. Examples of the descriptions used in a content based filtering approach could be the genre, lyrics and the users rating of a song if you are recommending songs to a user. The use of content based filtering have been explored thoroughly by e.g. Mooney et al. [16] where it is used in recommendation of books.

2.1.2. Collaborative filtering

The general idea of collaborative filtering is to identify users similar to the user in question and use their preferences to find items that also would fit with the interests of that user. Two much used methods for collaborative filtering are memory-based and model-based collaborative filtering.

*Memory-based collaborative filtering.* When memory-based collaborative filtering is used, recommendations are created based on the whole dataset of users, ratings and items. In many approaches, a neighborhood of users that previously have rated similar items as the user is created [22]. These neighborhoods can be formed e.g. with nearest neighbor algorithms such as a K-Nearest Neighbor algorithm using Cosine Similarities or Pearson Correlation to find similarities between users. The ratings of this neighborhood are then aggregated and compared to provide a list of top-N items for the user.

*Model-based collaborative filtering.* In a model-based approach, a model of user ratings is created that is used to predict a user’s rating to items [22] based on previous ratings. The model can be created using machine learning algorithms such as Bayesian classification and different clustering algorithms.
2.1.3. Hybrid approaches

Another popular method of providing recommendations are the use of a hybrid approach of the above two. As [1] explains you can either implement collaborative and content based systems separate and combine the results, incorporate characteristics from one of them into the other, or create a unifying model that has characteristics from both.

2.1.4. Social media filtering

With an ever more social web a new phenomenon has become more and more popular, namely social tagging. Social tags are basically “free text labels that are applied to items such as artists, albums and songs”. [12]. The tags can be everything from the genre of the song or artist, the record company of the artist or the mood the listener was in when listening to the song. By using these social tags you can get a much more complete picture of an artist than solely looking at the usual single genre specification in for example iTunes. In a collaborative environment these tags shouldn’t be used alone to classify such properties, as usually anyone can give a tag to a song. They have to be seen together with how frequently that tag has been used for that specific song and how much it is used overall to single out tags not appropriate for the item.

For example an artist mainly considered as a Rock artist, could also have influences of Jazz or Pop. In a social collaborative setting where a lot of people are tagging songs, this might get picked up and tagged by multiple people. By analyzing these tags you could therefore get a more complete and correct picture of the artist. This availability of music with social tags have recently inspired a number of music recommendation systems based on them. Enrich, Braunhofer and Ricci [8] presents a cross-domain recommendation technique that utilizes social-tags to overcome the well known cold-start problem of collaborative filtering recommendation systems. Kaminskas and Ricci [11] uses social tags when providing music recommendations based on a user’s location. Nanopoulos et al. [17] combines social tags and audio features to provide music recommendations. Su, Chang and Tseng [25] propose a content based solution utilizing user listening counts, item-tag and artist-tag similarities to provide recommendations.
2.2. Group recommendation systems

An important problem to solve when handling recommendations for groups is how to combine the individuals of the group so we can provide only one set of recommendations for the group as a whole. Two popular choices for handling this is aggregation of individual preferences into a common user profile which then can be used in a content-based or collaborative filtering approach, and aggregation of individual recommendations where individual recommendations for each member of the group is aggregated into a final recommendation list. Many aggregation strategies exist [13], including 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.

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 [13]. 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. However this also means that in general 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.

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 others not.

Average without misery aggregation. With an average without misery approach, items with a rating below a threshold for one of the users is 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 it 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. Context awareness

The term context is used in many situations such as in data mining, information retrieval and E-commerce Personalization [2]. Bazire and Brezillon [3] analyzes 150 definitions of context coming from different domains. Therefore it is important having a clear definition of it. In this paper 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” [3].

In many cases, additional context other than ratings might affect what a user think about an item. For example in a tourist recommendation system a user might finding 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 [2, 20, 26], however research when it comes to incorporating such contextual information into recommendation algorithms for groups isn’t as much explored. A model that supports context-aware group recommendations is presented by Stefandis, et al. [24]. 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 get. An example of such a parameter could be location who could be arranged as City ≺ State ≺ Country ≺ 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 prefer to go to indoor concerts if it is raining and outdoor concerts if it is sunny. The hierarchical structure would allow for usage of for example 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 is aggregated into a common user profile which is then used to create recommendations. This is done using a least misery approach combined with a fair design method.

Wang et al. [27] 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 charac-
teristics of a group is 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 based on the social structure of the group. These characteristics in combination with the context state inferred is used to provide recommendations.

2.4. Related work

Music recommendation is used in many big music application. You can find it in well-known services such as Spotify, Last.fm, Itunes, Grooveshark and Pandora. A lot of research on such systems exists however not much information about the algorithms behind the big services are publicly available. However it is assumed that the iTunes genius feature, Last.fm and Pandora recommendations uses versions of collaborative filtering. Pandora is a music streaming service that makes a personalized radio channel based on what you have listened to before and what kind of music you initially chose when creating the new channel. In Pandora each song is manually classified using a set of musical classifiers describing for example the beat, rhythm and instrumentation of a song, and then compares these classifiers to the type of the radio station currently playing to find songs that could fit with the one currently playing.

Spotify is a music streaming service that will grant you access to millions of songs. It also features a discovery page where you can find personalized music recommendations. For this discovery page, Spotify uses collaborative filtering as their basis for recommendations \[4\]. They use a collaborative filtering approach based on implicit feedback, meaning they utilize data implicitly gathered about a user such as the listening count for a specific song instead of explicitly specified data such as a preference for rock music \[10\]. A $N \times M$ matrix of songs and items is created where $N$ is the number of items in the database and $M$ is the number of users. The listen count for each of the item, user pairs respectively is put into the matrix. After this, vectors for each user and items are created using matrix factorization such as Probabilistic Latent Semantic Indexing (PLSA) \[7\]. These matrix factorizations are performed using Hadoop MapReduce jobs. The vectors produced can then

\[4\]http://www.pandora.com/
\[5\]https://www.spotify.com
be used to compute similarity between users and items using for example dot product to find items similar to what a user already has listened to and cosine similarity to find similarities between items.
3. Requirements

Scenario. Let’s say a group of friends is at a one-day festival with a lot of different bands having concerts. Their taste in music is quite different, so choosing concerts to attend is a challenge. Also a lot of the bands playing here are not very familiar to all of them. 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. One of the group members heard that other bands were playing nearby (but not as part the festival) so they would also like to get recommendations for nearby concerts not part of the festival. The group is in town for a whole week, so they would like to get recommendations for both the day of the festival and for the rest of the week.

Requirements. From the above scenario the following requirements for the system are extracted:

1. Concert recommendations should be provided based on what type of music a user has listened to before
2. The system should take the location of a concert into account (concerts close to a user are preferred).
3. The system should be able to provide recommendations for a more general location if not enough concerts are found for the given location.
4. The system should take time into account, it should not recommend concerts already taken place or concerts too far ahead in time. It should recommend concerts within the user’s provided parameters if such is provided.
5. If not enough concerts are found within the given time interval, the time constraint should be relaxed.
4. Design

A recommendation systems for concerts (CRS) differentiates themselves from traditional music recommendation systems. For a CRS to provide relevant recommendations the location of the concert and the time have to be taken into account as well. For example if you want to go to a concert the next weekend in London, you probably don’t want to get recommendations for concerts that happened in Los Angeles in 2003. You might want concerts that adhere to the constraints given. It can therefore be said that a CRS implicitly has to take context into account to provide relevant recommendations for a user.

The approach taken in this project consists of four main phases. First the context that is to be applied, which consists of two properties namely location and time, is defined. Next, the context defined is applied to the dataset to filter out irrelevant data. Then concert recommendations for each of the members of the group are calculated using a K-nearest neighbor algorithm. In the last phase, these individual recommendations are aggregated into one single recommendation list using average aggregation.

Definition 1. **Problem definition** In a recommendation system for concerts (CRS) a set of concerts $C$, artists $A$ and users $U$ are used to provide concert recommendations for a user $u$. Each of the concerts have a set of artists playing. Each of the users in $U$ have given a rating to a subset of the artists. A group $G \subseteq U$ consists of one or more users. The goal of the recommendation system is to provide the top $N$ most relevant concerts $C' \subseteq C$ for $G$ as a whole that adhere to the context applied $CP$ and the ratings $R$ each of the members of $G$ previously have given.

4.1. **Define context**

When wanting to attend a concert, two major variables have to be considered, location and time. Without them recommendations would possibly be meaningless, recommending concerts that already took place or in a location too far away. It is therefore crucial to define the context variables $CP = \text{location, time}$ so the rest of the algorithm can use these.
4.2. Filter dataset

In this phase the dataset is filtered based on the context parameters defined and other assumptions that can be done about it.

**Definition 2.** The function \( \text{filter}(\text{Con} \subseteq C, CP) \) returns the concerts \( C' \subseteq C \) that adhere to each of the context parameters \( CP \) given.

**Definition 3.** The function \( \text{listenedTo}(u' \in U, a \in A) \) returns whether or not the user \( u' \) has listened to the artist \( a \).

The first step of this phase is to reduce the data set of concerts to only those concerts in \( C \) that adhere to the context parameters in \( CP \).

\[
C' \subseteq C = \text{filter}(C, CP)
\]  

(1)

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. This is because only users that have listened to one or more of the artists playing will be considered in the collaborative filtering approach taken.

This gives the following reduced set of artists \( A' \) to consider for the recommendation phase.

\[
A' = \{a : \forall c \in C' \forall a \in c\}
\]  

(2)

In a \( K \)-nearest-neighbor collaborative filtering 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 \( A' \), can’t be considered for the \( K \) nearest neighbor 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 he might as well be left out (see section 4.4).

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 has listened to one or more of the artists in \( A' \).

\[
U' = \{u' : \forall u' \in U \exists a \in A' \text{listenedTo}(u', a)\}
\]  

(3)
4.3. Relaxing of Context

If not enough items are found, then the context should be relaxed in the following order until enough items are found.

1. Location
2. Time

This means first adding locations nearby the specified city, and then dates close to the one in question if not enough are found. How this is done, is left for the implementation phase, as no fitting way to fit it into the algorithm here was found.

4.4. Individual recommendations

For individual recommendations a collaborative filtering approach is used utilizing a K-nearest neighbor algorithm. In a K-nearest neighbor 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.

In this project K is defined as \( K = \sqrt{n} \), where n is the number of people having listened to the artist being evaluated.

The similarity measure chosen is the cosine similarity between two user’s listening count for each artist. The cosine similarity between two vectors \( x \) and \( y \) is defined as:

\[
sim(x, y) = \cos \theta = \frac{x \cdot y}{\|x\| \times \|y\|}
\]

Definition 4. The function \( \text{listenCount}(u' \in U, a \in A) \) returns the amount of times \( u' \) have listened to \( a \).

The user vector \( w_i \) for a user \( u_i \in U \) is defined as the vector of the users listening counts to each of the artists in \( A \).

\[
w_i = \{\text{listenCount}(i, a) : a \in A\}
\]

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
\[\text{sim}(u_1, u_2) = \text{sim}(w_1, w_2) = \frac{w_1 \cdot w_2}{\|w_1\| \times \|w_2\|} = \frac{\sum_{i=1}^{n} w_{1i} \times w_{2i}}{\sqrt{\sum_{i=1}^{n} (w_{1i})^2} \times \sqrt{\sum_{i=1}^{n} (w_{2i})^2}}\] (6)

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 in \(C'\) have 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. \(\text{totalSimilarity}\) 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 \(\frac{\text{sim}(u_i, u)}{\text{totalSimilarity}}\). The actual contribution is influenced by the rating given to \(a\) by \(u_i\), so this is multiplied with \(\text{rating}(u_i, a)\). The predicted rating for an artist \(i\) will then be:

\[\text{artistRating}_i = \sum_{j=1}^{n} \frac{\text{sim}(u_j, u) \times \text{listenCount}(u_j, a_i)}{\text{totalSimilarity}}\] (7)

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

\[\text{rating}^u_c = \frac{\sum_{k=1}^{m} \text{artistRating}_k}{m}\] (8)
4.5. Aggregate Individual preferences into one recommendation list

The algorithm has only utilized implicit gathered data about a user, its listening counts to different artists. In such datasets, it is hard to take things like misery into account as there is no way for a user to state negative preferences. A listening count of 0 for an artist doesn’t necessarily mean that the user doesn’t like the artist. Therefore an average approach is used to aggregate recommendations. We aggregate the recommended lists for each of the users, taking the average of the predicted scores between them. So for each concert $c$ in $C'$, the final score for the concert $totalRating_c$ is the average of the predicted scores from the people in the group for that concert.

$$totalRating_c = \frac{\sum_{u=1}^{n} rating_u^c}{n}$$  \hspace{1cm} (9)

The final predictions returned is the top $N$ concerts in $C'$ with the highest rating $totalRating_c$. 
Table 1: Dataset properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Column</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>user</td>
<td>10820</td>
</tr>
<tr>
<td>Artists</td>
<td>artist</td>
<td>65416</td>
</tr>
<tr>
<td>Concerts</td>
<td>concert</td>
<td>1024</td>
</tr>
<tr>
<td>Listening counts</td>
<td>artistlistenings</td>
<td>543390</td>
</tr>
<tr>
<td>Tags</td>
<td>tag</td>
<td>138633</td>
</tr>
<tr>
<td>Tags for artists</td>
<td>artisttag</td>
<td>1135341</td>
</tr>
<tr>
<td>Artist concert participation</td>
<td>concertparticipation</td>
<td>2458</td>
</tr>
<tr>
<td>User similarities</td>
<td>usersimilarity</td>
<td>50280000</td>
</tr>
<tr>
<td>Venues</td>
<td>venue</td>
<td>251</td>
</tr>
</tbody>
</table>

5. Implementation

5.1. Dataset

For the purpose of this project, a dataset was created using data from the popular music streaming service Last.fm\(^6\). The dataset was created using Last.fm’s publicly available API\(^7\). To discover users, the ‘User.getNeighbors’ call was used using algorithm\(^1\). It discovers users recursively by traversing a user’s neighborhood. The random part of the algorithm was included to get a more diverse set of users. For each of the users added to the dataset, the top 50 artists that user had listened to was added to the dataset using the ‘User.getTopArtists’ call. Concerts were found using the Geo.getEvents call, searching for concerts in a 100km radius around London and New York between 3. November 2013 and 14. December 2013. For the concerts found that contained artists that hadn’t been listened to by any users in the dataset, the top listeners to that artist were added to the dataset using the ‘Artist.getTopFans’ call. This was done to ensure that all concerts (even the small ones) would have a chance to be picked up by the recommendation algorithm. Also the dataset contains cosine-similarities between users based on their listening habits. Using this approach in addition to adding the relevant tags for each artist and venues for each concert, resulted in a dataset with counts as seen in table\(^1\). The dataset was created

\(^6\)http://www.last.fm/
\(^7\)http://www.last.fm/api
Algorithm 1 Finding users

\[\begin{aligned}
\text{u} \leftarrow \text{simensma} \\
\text{n} \leftarrow \text{max number of users} \\
\text{i} \leftarrow 1 \\
\text{users} \leftarrow \emptyset \\
\text{while} \ true \ do \\
\text{tmp} \leftarrow \text{User.getNeighbors(u)} \\
\text{for} \ \text{user} \ \text{in} \ \text{tmp} \ \text{do} \\
\text{rand} \leftarrow \text{Math.Random()} \\
\text{if} \ \text{rand} < 0.5 \ \&\ \text{user} \ \text{not in} \ \text{users} \ \text{then} \\
\text{users} \leftarrow \text{users} \cup \text{user} \\
\text{i} \leftarrow \text{i} + 1 \\
\text{if} \ \text{i} == \text{n} \ \text{then} \ \text{return} \ \text{users} \\
\text{end if} \\
\text{end if} \\
\text{end for} \\
\text{u} \leftarrow \text{tmp.getRandom()} \\
\text{end while}
\end{aligned}\]

as a MySQL database with a schema like the one in figure 1.
Figure 1: Database diagram for dataset
5.2. Environment

A prototype was developed implementing the algorithm in section 4. The prototype features a front end in HTML/Javascript based on the Knockout.js framework (see figure 2). The back end is developed in Java utilizing a RESTful architecture, providing support for fetching of information in the format of JSONP\(^8\). The prototype is available at github\(^9\).

---

\(^{8}\)http://json-p.org/

\(^{9}\)https://github.com/simensma/GroupRec
5.3. Algorithm

The algorithm from section 4 was split into two parts, one resource intensive part that is performed offline, and one part performed online on the fly.

5.3.1. Offline calculation

Transformation of listening counts. In a traditional recommendation system, ratings of items are used as a basis for recommendations. In the dataset built, only listening counts for each of the artists are available, so the algorithm is based on implicit user feedback. One problem with using listening counts only is that they don’t capture negative ratings. A listening count of 1 doesn’t necessary mean that the user doesn’t like the song. Another problem is that listening counts aren’t normalized, so there is no boundaries on the values they can contain. A new user have only listened to some artists a couple of times, whereas an old user might have listened to an artist thousands of times. Therefore the listening counts for an artist by a user are normalized by the user’s total listening counts, this is seen in the \textit{maxListenCount} field of the \texttt{artistlistenings} column.

Calculation of similarities. The first part performed offline is the calculation of similarities between users. Using the cosine similarity approach described in section 4.4 the similarity between all users is calculated. If the similarity between two users is greater than 0, it is stored in the \textit{usersimilarity} table of the database. A missing record of similarity between users are therefore to be seen as a similarity of 0. It is important to note here that similarities between users are stored twice, for example similarities between users \texttt{A} and \texttt{B} are stored both as (\texttt{a,b,similarity}) and (\texttt{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.

5.3.2. Online calculation

Definition of group. The definition of the group is performed by the user, and in the prototype this is performed in the main view by searching for usernames (see figure 3).

Definition of context. Context is also defined by the user. The user provides a date interval and location for the concert as seen in figure 4 and figure 5.
Figure 3: Prototype user search

Figure 4: Prototype user date

Figure 5: Prototype user location
Filter dataset. The above context parameters are sent to the server through a REST API, and processed there. First all concerts and the artists playing at that concert are fetched from the database using a SQL query like the one in listing 1.

**Listing 1** Filter dataset query
```sql
SELECT c.id AS id, cp.artistId AS artistId, DISTANCE() AS distance
FROM concert c, concertparticipation cp, citylocation cl, venue v
WHERE cp.concertId=c.id AND c.startdate>=START_DATE
    AND c.startdate<=END_DATE AND v.id=c.venueId
    AND v.city=LOCATION AND cl.city=v.city;
    AND (enddate IS NULL OR enddate>=?)
```

**Listing 2** Filter dataset based on distance
```sql
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;
```

The DISTANCE() function as seen in listing 2 calculates the distance in kilometers between the geolocation of a concert and the geolocation of the city given from the user. The Haversine formula is used which can be used to estimate the shortest distance between two points on the earth’s surface [5].
The DISTANCE() function using the Haversine formula

```sql
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
```

Relaxing of context To support relaxing of context as specified in section 4.3 also concerts within 5 days (on both sides of the date range in question) are fetched in addition to the 100 concerts closest to the location given. Extra concerts based on date are fetched by adding 5 days on both sides to the start date given by the user. Selecting the 100 concerts closest to the location given is done modifying the query in listing 1 to take distance into account. This query can be seen in listing 2. The results are then merged with the results of the filtered dataset based on the original context variables (previous paragraph).

Fetch artist listeners

```sql
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
```

After this, the users that have listened to these artists and their similarities to each of the members of the group specified are fetched, using the SQL query in listing 4 for each user in the group.

Predict listening frequency for each artists. For each user in the group we predict the listening frequency (listening count for the artist divided by the total listening counts of the user) he would have for each of the artists if the user hasn’t listened to the artist before. So for each of the users in the group, each of the artists found in the previous step are given a predicted listening frequency using a method similar to the one in listing 5. As we can see, each of the users that have listened to the artist, will contribute
to the total prediction for the artist by its listening frequency to the artist times its percentage of total similarity to the group member being evaluated (sim/tot). This will ensure that the most similar users to the group member will contribute most to the prediction.

**Listing 5 Calculation of artist score**

/* Contains similarities (Double) between all the users (Integer) that have listened to some of the artists found and the user being processed. */
Map<Integer,Double> userSimilarities;

/* Artist being processed */
int artistId;

/* 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>> artistListeners;

/* Sum of the similarities to the given user for all the users that have listened to the artist being processed. */
double tot;
double score = 0;

for (Tuple<Integer, Double> tuple : artistListeners) {
    double sim = userSimilarities.get(tuple.getX());
    double lFreq = tuple.getY();
    score += lFreq * sim / tot;
}
addScore(artistId, score);

**Prediction for concert.** A concert can, as defined, have multiple artists playing. Therefore the final score of a concert is the average of the predicted listening frequencies for the artists playing. This is then calculated for each user so that each concert will have a rating from each of the users in the group.

**Combine results.** For each of the concerts, the ratings from the individual users are so combined into one rating for the concert. This is done by taking
the average of these ratings (see section 4.5).

*Filter results.* The above implementation yields a list of concerts with a combined predicted rating for each of them. The 10 concerts with the highest predicted rating and that fits with the context specified, the 10 concerts with the highest ratings amongst those 100 extra fetched based on distance, and the 10 top concerts for the extra concerts based on date, are returned to the client for viewing.

*View results.* When the recommended concerts are returned, the web page displays some details about the concerts such as location information and who is playing as seen in figure 6. You can also view extra information about a user (listening count) figure 7 and artists (social tags and their frequency of use) figure 8 by respectively clicking on them.

Also a user gets an option to relax the context by adding the additional found concerts (by distance and date) to the view. A link to the extra concerts by distance and dates are provided as seen in 9. By clicking these links, the respective concerts will be added to the view (seen by the light blue concerts in figure 10).
### Recommended concerts

Show 30 extra concerts found between 3.5 and 86.2 km from Brighton
Show 1 extra concerts found the 16th Nov 2013
Show 1 extra concerts found the 17th Nov 2013
Show 5 extra concerts found the 18th Nov 2013
Show 1 extra concerts found the 19th Nov 2013
Show 2 extra concerts found the 20th Nov 2013

<table>
<thead>
<tr>
<th>Concert</th>
<th>Venue</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Crystal Fighters</strong></td>
<td><strong>Concorde 2</strong></td>
<td>0.5</td>
</tr>
<tr>
<td>21st Nov 2013</td>
<td>Brighton</td>
<td></td>
</tr>
<tr>
<td>Crystal Fighters (0.5)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Woe, Is Me</strong></td>
<td><strong>The Haunt</strong></td>
<td>0.3</td>
</tr>
<tr>
<td>29th Nov 2013</td>
<td>Brighton</td>
<td></td>
</tr>
<tr>
<td>Our Last Night (0.3), Woe, Is Me (0.0), Empires Fade (0.0)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Mindless Self Indulgence</strong></td>
<td><strong>Concorde 2</strong></td>
<td>0.2</td>
</tr>
<tr>
<td>28th Nov 2013</td>
<td>Brighton</td>
<td></td>
</tr>
<tr>
<td>Mindless Self Indulgence (0.2), The Red Paintings (0.0)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Electric Six</strong></td>
<td><strong>Concorde 2</strong></td>
<td>0.2</td>
</tr>
<tr>
<td>10th Dec 2013</td>
<td>Brighton</td>
<td></td>
</tr>
<tr>
<td>Electric Six (0.2)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Oh Land</strong></td>
<td><strong>The Haunt</strong></td>
<td>0.1</td>
</tr>
</tbody>
</table>

Figure 6: Result view
Figure 7: User view

Figure 8: Artist view

Figure 9: Relaxing of context view

Figure 10: Results after relaxing of context (additional concerts highlighted)
6. Results

Evaluating a music recommendation system can be very difficult as taste in music are highly subjective, and a lot of factors impact the final results. In this section, three scenarios that describe how the application can be used in a real setting, followed by a discussion about the scenarios and their output. The three scenarios show how the requirements of section 3 are met by the prototype.

Requirement 1 specified that concert recommendations should be provided based on what type of music a user has listened to before. From the implementation details in section 5.3.2 we can see that the prototype does this by basing the recommendations on a user’s previous artist listening counts.

Scenario 1. Simen, David and Marie are in London together for the weekend 15.11.13 - 17.11.13 and they want to find a concert to attend on either of these days. They access the application and is presented with the view as seen in figure 11. They so add themselves to the group by searching for their LastFm usernames as seen in figure 12. Then they put in the dates they want to search for as seen in figure 13. At last they choose London as the location they want to get recommendations for (figure 14), and press the big blue button. This gives them the view of recommended concerts as seen in figure 15. None of the members of the group have heard about any of the artists suggested, except for Simen which have at least heard about KT Tunstall. They don’t want to travel outside London either, so they decide to go with the KT Tunstall concert.

This scenario shows how requirements 2 and 4 from section 3 are met. The system clearly takes the location and time of a concert into account when they specify the context, and this can also be seen in the results in figure 14.
Figure 11: Scenario 1 - Main page of the prototype

Figure 12: Scenario 1 - Adding user to the group

Figure 15: Scenario 1 - Viewing results
Scenario 2. Simen stays at home in London after a long weekend and meets up with his friend Oskar. Simen and Oskar share the same taste in music and they plan on going to a concert within the next 14 days in London. They fill in the data as seen in figure 16. This gives the concert recommendations as seen in figure 17. They aren’t really satisfied with the concerts recommended, so they decide to add the concerts for 13th of November. These are easily identified by being highlighted in light blue (figure 18). None of them stood out either, so they decide to give up and go bowling instead.

In this scenario we can see that the prototype covers the needs of requirement 5. Simen and Oskar don’t find enough concerts that fits with their wishes within the given time interval, so they get the option to show extra concerts in the days before and after what they specified (the time constraint can be relaxed).
Figure 16: Context for scenario 2
### Recommended concerts

<table>
<thead>
<tr>
<th>Concert</th>
<th>Venue 1</th>
<th>Venue 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loom</strong></td>
<td>The Victoria</td>
<td>2.6</td>
</tr>
<tr>
<td>22nd Nov 2013</td>
<td>London</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Crow (2.6) , Loom (0.3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Alter Ego Tour</strong></td>
<td>O2 Academy</td>
<td>1.4</td>
</tr>
<tr>
<td>24th Nov 2013</td>
<td>Islington</td>
<td></td>
</tr>
<tr>
<td>Patrycja Markowska (1.4)</td>
<td>London</td>
<td>United Kingdom</td>
</tr>
<tr>
<td><strong>Arcane Roots</strong></td>
<td>XOYO</td>
<td>1.3</td>
</tr>
<tr>
<td>26th Nov 2013</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>Verses (1.3) , Arcane Roots (0.2) , Empire (0.2) , Verses UK (0.6) more</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Mindless Self Indulgence</strong></td>
<td>Koko</td>
<td>1.2</td>
</tr>
<tr>
<td>29th Nov 2013</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>Mindless Self Indulgence (1.2) , The Dead Betas (0.7) , The Red Paintings (0.5)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Disclosure</strong></td>
<td>O2 Academy</td>
<td>1.2</td>
</tr>
<tr>
<td>28th Nov 2013</td>
<td>Brixton</td>
<td></td>
</tr>
<tr>
<td>29th Nov 2013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disclosure (1.2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 17:** Results for scenario 2

<table>
<thead>
<tr>
<th>Concert</th>
<th>Venue 1</th>
<th>Venue 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blackberry Smoke</strong></td>
<td>London Bar &amp;</td>
<td>2.3</td>
</tr>
<tr>
<td>15th Nov 2013</td>
<td>Kitchen</td>
<td></td>
</tr>
<tr>
<td>Blackberry Smoke (2.3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Alter Ego Tour</strong></td>
<td>O2 Academy</td>
<td>1.4</td>
</tr>
<tr>
<td>24th Nov 2013</td>
<td>Islington</td>
<td></td>
</tr>
<tr>
<td>Patrycja Markowska (1.4)</td>
<td>London</td>
<td>United Kingdom</td>
</tr>
<tr>
<td><strong>Starlucker</strong></td>
<td>Hoxton Square Bar &amp; Kitchen</td>
<td>1.3</td>
</tr>
<tr>
<td>13th Nov 2013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starlucker (1.3) , STRFKR (0.0)</td>
<td>London</td>
<td>United Kingdom</td>
</tr>
<tr>
<td><strong>Arcane Roots</strong></td>
<td>XOYO</td>
<td>1.3</td>
</tr>
<tr>
<td>26th Nov 2013</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>Verses (1.3) , Arcane Roots (0.2) , Empire (0.2) , Verses UK (0.6) more</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Mindless Self Indulgence</strong></td>
<td>Koko</td>
<td>1.2</td>
</tr>
<tr>
<td>29th Nov 2013</td>
<td>London</td>
<td></td>
</tr>
<tr>
<td>Mindless Self Indulgence (1.2) , The Dead Betas (0.7) , The Red Paintings (0.5)</td>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td><strong>Disclosure</strong></td>
<td>O2 Academy</td>
<td>1.2</td>
</tr>
<tr>
<td>28th Nov 2013</td>
<td>Brixton</td>
<td></td>
</tr>
<tr>
<td>29th Nov 2013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disclosure (1.2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 18:** Results from scenario 2 with extra days added (highlighted in light blue)
Scenario 3. Simen is visiting Oskar in Brighton for two weekdays between the 10.-12. December 2013, and wants to go to a concert. The recommendation system gives them the results seen in figure 19. None of the concerts really stands out, and as Simen brought his car, they don’t mind travelling a bit. They click on “Show extra concerts between X and Y kilometers from Brighton”. 10 new concerts are added to the results (figure 20), highlighted in light blue. None of the concerts listed completely fits with what the users are usually listening to, however they have heard a lot about Black Sabbath, so they decide to go to that concert.

Scenario 3 shows that also requirement 3 has been attended to. Simen and Oskar didn’t find any fitting concerts in Brighton, however they get the opportunity to view concerts in close proximity to Brighton, which in terms means that they can view concerts at a more general location than the one given.
6.1. Discussion

By having a look at the results from the above scenarios, we can easily see that most of the concerts recommended are held by ‘lesser known’ artists. This is to be expected as generally the amount of "world-star" artists playing at a single location within the same small timespan are limited. Assessing how good the recommendations actually are requires some more investigation. For scenario 1 we can see the overall top tags and accompanying frequencies of occurrence between them for the artists Simen, Marie and David respectively have listened to in table 2. All tag data in the data set, are fetched from LastFm where users can provide tags to the artists. There are...
Table 2: User tags for Simen, David and Marie

<table>
<thead>
<tr>
<th>simensma</th>
<th>Freq</th>
<th>DaviDOdger</th>
<th>Freq</th>
<th>mariekevdp</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>electronic</td>
<td>100</td>
<td>alternative</td>
<td>100</td>
<td>indie</td>
<td>100</td>
</tr>
<tr>
<td>House</td>
<td>54</td>
<td>indie</td>
<td>97</td>
<td>alternative</td>
<td>96</td>
</tr>
<tr>
<td>indie</td>
<td>51</td>
<td>female vocalists</td>
<td>92</td>
<td>electronic</td>
<td>84</td>
</tr>
<tr>
<td>alternative</td>
<td>32</td>
<td>pop</td>
<td>90</td>
<td>ambient</td>
<td>75</td>
</tr>
<tr>
<td>dance</td>
<td>31</td>
<td>rock</td>
<td>80</td>
<td>rock</td>
<td>69</td>
</tr>
<tr>
<td>rock</td>
<td>29</td>
<td>80s</td>
<td>65</td>
<td>experimental</td>
<td>62</td>
</tr>
<tr>
<td>electronica</td>
<td>27</td>
<td>new wave</td>
<td>65</td>
<td>female vocalists</td>
<td>61</td>
</tr>
<tr>
<td>trance</td>
<td>21</td>
<td>electronic</td>
<td>61</td>
<td>post-rock</td>
<td>52</td>
</tr>
<tr>
<td>indie pop</td>
<td>20</td>
<td>british</td>
<td>57</td>
<td>alternative rock</td>
<td>49</td>
</tr>
<tr>
<td>pop</td>
<td>19</td>
<td>singer-songwriter</td>
<td>49</td>
<td>folk</td>
<td>43</td>
</tr>
<tr>
<td>heavy metal</td>
<td>19</td>
<td>indie rock</td>
<td>43</td>
<td>singer-songwriter</td>
<td>39</td>
</tr>
<tr>
<td>Hip-Hop</td>
<td>19</td>
<td>post-punk</td>
<td>42</td>
<td>neofolk</td>
<td>34</td>
</tr>
<tr>
<td>electro house</td>
<td>18</td>
<td>alternative rock</td>
<td>31</td>
<td>indie rock</td>
<td>31</td>
</tr>
<tr>
<td>dubstep</td>
<td>18</td>
<td>folk</td>
<td>26</td>
<td>doom metal</td>
<td>28</td>
</tr>
<tr>
<td>Progressive House</td>
<td>17</td>
<td>dance</td>
<td>24</td>
<td>shoegaze</td>
<td>28</td>
</tr>
</tbody>
</table>

no restrictions on what tags you can give to an artist, so every tag might not be accurately describing the artist. However in general, these tags can give a pretty good impression of what sort of music an artist play, and therefore can give a pretty good picture on what sort of music a user likes in general.

From the figure we can see that Simen and Marie have a pretty similar taste in music, with a lot of the top tags being the same, whereas David listen to a more diverse kind of music. In table 3 the top tags of the artists playing at the highest scoring concerts from scenario 1 can be seen. If these tags are compared to the top tags from the three users, it is easily spotted that none of these concerts are the best match with the preferences of the users. Three of the four artists playing seems to be heavily punk influenced, which aren’t reflected by the user’s preferences.

The same can be said if we have a look at the results from scenario 2. In table 4 the top tags for Simen and Oskar can be seen. By comparing their top tags it is safe to assume that they have a similar taste in music, with a lot of the tags with the highest frequency being the same. If we compare these to the tags from the top four recommended concerts in table 5 these concerts don’t seem to be the best fit for the users either. The top portions
of the user tag and concert tag lists have few tags in common, which is an indicator of this.

Again for scenario 3 the artists recommended don’t seem to be the best fit for the users. By comparing the top user tags in table 4 and the top tags for the artists playing at the top four recommended concerts in table 6, we can see that the lists have few common tags.

The outcome of the scenarios described above in general seem to indicate that the recommended concerts don’t seem to be the best fit with the preferred type of music the members of the groups listen to. There are multiple possible reasons why this is happening.

Data sparsity. One factor that might affect the results given is the known problem of data sparsity in collaborative filtering [1]. A user has generally listened to only a small subset of all the artists in the dataset. In the dataset created, the 50 top artists for a user is listed, which is less than 0.08% of the total artists in the dataset. Generally a lot of the artist have only been listened to by a few people, which means they rarely will be recommended because connecting them to a user will be difficult. When we in addition have to take context into account, problems might occur. In a usual K-nearest neighbor approach you pick your K-nearest neighbors from all the existing
<table>
<thead>
<tr>
<th>simensma</th>
<th>Freq</th>
<th>DaviDOdger</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>electronic</td>
<td>100</td>
<td>electronic</td>
<td>100</td>
</tr>
<tr>
<td>House</td>
<td>54</td>
<td>pop</td>
<td>82</td>
</tr>
<tr>
<td>indie</td>
<td>51</td>
<td>indie</td>
<td>80</td>
</tr>
<tr>
<td>alternative</td>
<td>32</td>
<td>Hip-Hop</td>
<td>55</td>
</tr>
<tr>
<td>dance</td>
<td>31</td>
<td>dance</td>
<td>53</td>
</tr>
<tr>
<td>rock</td>
<td>29</td>
<td>alternative</td>
<td>47</td>
</tr>
<tr>
<td>electronica</td>
<td>27</td>
<td>rock</td>
<td>45</td>
</tr>
<tr>
<td>trance</td>
<td>21</td>
<td>rap</td>
<td>41</td>
</tr>
<tr>
<td>indie pop</td>
<td>20</td>
<td>House</td>
<td>38</td>
</tr>
<tr>
<td>pop</td>
<td>19</td>
<td>female vocalists</td>
<td>34</td>
</tr>
<tr>
<td>heavy metal</td>
<td>19</td>
<td>british</td>
<td>29</td>
</tr>
<tr>
<td>Hip-Hop</td>
<td>19</td>
<td>indie pop</td>
<td>28</td>
</tr>
<tr>
<td>electro house</td>
<td>18</td>
<td>alternative rock</td>
<td>27</td>
</tr>
<tr>
<td>dubstep</td>
<td>18</td>
<td>rnb</td>
<td>26</td>
</tr>
<tr>
<td>Progressive House</td>
<td>17</td>
<td>indie rock</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 4: Most used tags for Simen and Oskar (scenario 2)

<table>
<thead>
<tr>
<th>Crows</th>
<th>Freq</th>
<th>BlackBerry</th>
<th>Freq</th>
<th>Patrycja</th>
<th>Freq</th>
<th>Starfucker</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoke</td>
<td></td>
<td>Southern Rock</td>
<td>100</td>
<td>polish</td>
<td>100</td>
<td>electronic</td>
<td>100</td>
</tr>
<tr>
<td>Smoke</td>
<td>100</td>
<td>blues rock</td>
<td>41</td>
<td>female vocalists</td>
<td>37</td>
<td>indie pop</td>
<td>80</td>
</tr>
<tr>
<td>Smoke</td>
<td>41</td>
<td>country rock</td>
<td>35</td>
<td>rock</td>
<td>28</td>
<td>indie</td>
<td>69</td>
</tr>
<tr>
<td>Smoke</td>
<td>35</td>
<td>rock</td>
<td>29</td>
<td>pop</td>
<td>21</td>
<td>portland</td>
<td>64</td>
</tr>
<tr>
<td>Smoke</td>
<td>29</td>
<td>classic rock</td>
<td>23</td>
<td>pop rock</td>
<td>8</td>
<td>Belgium</td>
<td>64</td>
</tr>
<tr>
<td>Smoke</td>
<td>23</td>
<td>American</td>
<td>23</td>
<td>polska przeprasza</td>
<td>6</td>
<td>synthpop</td>
<td>64</td>
</tr>
<tr>
<td>Smoke</td>
<td>23</td>
<td>American</td>
<td>5</td>
<td>patrycja markowska</td>
<td>5</td>
<td>indietronica</td>
<td>6</td>
</tr>
<tr>
<td>Smoke</td>
<td></td>
<td>country</td>
<td>17</td>
<td>female vocalist</td>
<td>5</td>
<td>american</td>
<td>5</td>
</tr>
<tr>
<td>Smoke</td>
<td>17</td>
<td>rock atual foda</td>
<td>3</td>
<td>PL</td>
<td>3</td>
<td>electropop</td>
<td>5</td>
</tr>
<tr>
<td>Smoke</td>
<td>3</td>
<td>BlackBerry</td>
<td>17</td>
<td>Pop-Rock</td>
<td>3</td>
<td>electronica</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5: Top tags for artists playing at the top concerts from scenario 2
<table>
<thead>
<tr>
<th>Haim</th>
<th>Freq</th>
<th>Black Sabbath</th>
<th>Freq</th>
<th>Electric Six</th>
<th>Freq</th>
<th>White Lies</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>indie</td>
<td>100</td>
<td>heavy metal</td>
<td>100</td>
<td>rock</td>
<td>100</td>
<td>indie rock</td>
<td>100</td>
</tr>
<tr>
<td>indie pop</td>
<td>80</td>
<td>hard rock</td>
<td>57</td>
<td>alternative</td>
<td>71</td>
<td>post-punk</td>
<td>82</td>
</tr>
<tr>
<td>female vocalists</td>
<td>80</td>
<td>classic rock</td>
<td>45</td>
<td>indie</td>
<td>55</td>
<td>british</td>
<td>82</td>
</tr>
<tr>
<td>indie rock</td>
<td>65</td>
<td>metal</td>
<td>42</td>
<td>electronic</td>
<td>52</td>
<td>indie</td>
<td>75</td>
</tr>
<tr>
<td>soul</td>
<td>43</td>
<td>rock</td>
<td>28</td>
<td>Disco</td>
<td>49</td>
<td>alternative</td>
<td>61</td>
</tr>
<tr>
<td>pop</td>
<td>22</td>
<td>doom metal</td>
<td>19</td>
<td>alternative rock</td>
<td>17</td>
<td>rock</td>
<td>21</td>
</tr>
<tr>
<td>soft rock</td>
<td>19</td>
<td>british</td>
<td>8</td>
<td>indie rock</td>
<td>17</td>
<td>alternative rock</td>
<td>16</td>
</tr>
<tr>
<td>american</td>
<td>15</td>
<td>70s</td>
<td>6</td>
<td>dance</td>
<td>12</td>
<td>Post-punk revival</td>
<td>11</td>
</tr>
<tr>
<td>alternative</td>
<td>13</td>
<td>Black Sabbath</td>
<td>4</td>
<td>american</td>
<td>8</td>
<td>new wave</td>
<td>9</td>
</tr>
<tr>
<td>rock</td>
<td>9</td>
<td>Stoner Rock</td>
<td>4</td>
<td>funk</td>
<td>6</td>
<td>london</td>
<td>4</td>
</tr>
<tr>
<td>acoustic</td>
<td>6</td>
<td>classic metal</td>
<td>3</td>
<td>comedy</td>
<td>5</td>
<td>UK</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6: Top tags for artists playing at the top concerts from scenario 3

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>87</td>
<td>55</td>
</tr>
<tr>
<td>Median</td>
<td>18</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 7: Number of unique users used for recommendation per concert

<table>
<thead>
<tr>
<th>simensma</th>
<th>mariekevdp</th>
<th>DavidDOdgers</th>
<th>oskars26</th>
</tr>
</thead>
<tbody>
<tr>
<td>2584</td>
<td>2965</td>
<td>3532</td>
<td>3183</td>
</tr>
</tbody>
</table>

Table 8: Number of users with a similarity >0
users. With context, the number of available users is reduced drastically. When combined with the fact that most concerts are held by 'lesser known' artists with few people having listened to them, this can lead to that users with a low similarity to yourself are picked (as the only option available), and recommendations that in fact aren’t what you originally wanted will be returned. For example in scenario 2, the average number of users that have listened to an artist playing at a concert and also have a similarity to one of the users greater than 0, is on average 55 and the median for the scenario is 9 as seen in table 7. This means that the K-nearest neighbor algorithm would pick its top \( K \) users out of a pool of 55 users on average in this scenario. In comparison, a traditional approach to a K-nearest neighbor collaborative filtering algorithm, can utilize the pool of all similarities to a user. As seen in table 8 these numbers varies between 2500-3200 for the users in these scenarios.

Choice of \( K \). In a \( K \)-nearest-neighbor algorithm, the choice of \( K \), the number of users that should contribute to the score for an artist, has a great impact on the outcome of the algorithm. When \( K \) is too high, the boundaries between objects get smoother \cite{23}, it gets difficult to separate them, and we possibly end up with a bunch of recommendations with almost the same predicted value. With a \( K \) too low, the results will be heavily influenced by noise, single users can have too much impact on the result. In this paper \( K = \sqrt{n} \) where \( n \) is the number of people having listened to the artist being evaluated. However, this might not necessarily be the optimal choice of \( K \).

Implicit dataset. In section 5.3.1 a naive approach to normalizing listening counts was introduced to tackle the problem of using implicit datasets. Both \cite{19} and \cite{10} identifies the need for a more thorough approach. \cite{19} states that ”in an explicit dataset, the distribution of ratings are not heavily skewed towards one end or another”. In an implicit dataset on the other hand, most artist have a rating (play frequency) close to 0. This will cause most of the predicted ratings for an artist to be low when just using the naive approach. This can also be one reason to why the recommendations aren’t as accurate as wished.
7. Conclusions and future work

In this project, a prototype of a context aware group recommender system for concerts has been developed in addition to a dataset that makes this possible. The prototype is using a plain user-based collaborative filtering approach, using a slightly modified K-nearest neighbor algorithm.

The results found through the definition of 3 scenarios, shows that the final recommendations provided by the algorithm aren’t as accurate as wanted. Some possible sources of this error have been identified and a deeper investigation into them is needed in order to improve the outcome it.

Several lines of work emerge from the work done in this project. From the results it is apparent that there is a need to improve the results of the recommendations. Future work here include investigating how the data-sparsity challenge can be overcome with the extra challenges with context sensitive data. Also finding a more optimal choice of $K$ might be worth investigating. Also an investigation into if the use of implicit ratings is a source for the unwanted results should be performed. Is it enough to optimize the current algorithm to overcome these needs? or is there a need to investigate different approaches than a pure collaborative approach based on listening counts?

There is also a need for a more thorough evaluation of the prototype in general. Both when it comes to the accuracy of the recommendations and testing the prototype in a real life environment to see how it is perceived by possible users.

Another possible way to go from here, is to have a more in depth look at the context part of the prototype. Is there a wish to add more context parameters such as weather? Are other methods, such as the use of GPS or linking it with Facebook groups for defining the context parameters, suitable for such a group recommender system?
References


44