Johannes Singer, BSc

Algorithms for Recommending Music to Groups

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Graz, September 2019

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

Abstract

In recent years the way of listening to music has changed due to streaming services. This made it necessary to develop systems which can recommend music out of a huge amount of songs. There are often situations in which it is useful to recommend songs for more than one person. For example, on a road trip, in a learning group or on a party it is great to have a playlist matching the preferences of a whole group. The focus of this thesis is to create a system which is accessible for many people, which is easy to use and which creates playlists all group members are happy with. The thesis starts with the theoretical background followed by the algorithms to recommend music to groups. Furthermore, the app *M'Usic* is introduced, which is the implementation of the defined goals. An evaluation of the app is also part of this thesis. The conclusion sums up the findings and contains an outlook of the future work in this topic.

Contents

List of Figures

1. Introduction

This thesis is related to the topic of recommending music to groups. An environment to recommend items is called recommender system (RS). Such systems suggest items based on available information to single users or groups. Jannach (2011) and Ricci, Rokach, and Shapira (2015) are great references for RSs as an introduction to the topic and also provide detailed information in several topics of RSs. Whereas systems recommending items to single users are wide-spread in the daily life, group recommender systems (GRSs) are not according to Felfernig, Stettinger, et al., 2018. Especially in the domain of music, environments for suggesting music are getting more and more important with the increase of streaming services. Although there are many situations where people wishing to listen to music have a similar taste in music, there are still often situations where people with different music preferences want to listen to music that everyone likes. Examples for such situations are friends meeting together up, colleagues working or studying together, or the family who is driving in the car. Furthermore, there are situations in which group members are not knowing each other like customers in a shop or people exercising in a gym. Unfortunately, all the existing services just provide recommendation techniques for single users. In an entrepreneurial environment it is easy to solve this problem: An employee creates a playlist which should go well with the preferences from most customers. Within a group of friends or within the family selecting music can be more frustrating caused by different tastes in music and the decision which group member is allowed to select the tracks. The solution is a system recommending music to the whole group's taste in music. As the system needs to know the preferences of the users, former RSs asked the users about them via ratings, which can be annoying. Nowadays, many people use music streaming services for listening to their favorite songs. The outcome is a big data collection about the preferences of users. This leads to the idea to create a RS which connects users with other users and creates playlists that correlate to the preferences of the whole group. The aim for the system, implemented within the extent of this thesis, are the following:

- Create a system to recommend music to groups.
- The system should be available for a large number of users.
- The recommendations should have a high value of accuracy.
- Groups are able to get music suggestions without answering explicit questions about their preferences.
- Suggested playlists should be playable in a straightforward fashion.
- The system should work without the need of any music licenses or fees.

To achieve these aims the current research in GRSs and recommending music has to be analyzed and improved. After this, a software has to be developed, which implements a group recommendation approach and fulfills the aims defined above. A survey should be conducted to evaluate the software's performance.

The thesis is structured as follows:

In Chapter 2 techniques to develop a RS are described. Starting with different basic approaches to suggest items, the chapter continues with data mining techniques. Further, the special characteristic of recommending music is discussed. There is also described how to evaluate a RS. In the final section the adoptions to recommend items to groups are explained.

Chapter 3 considers the possibilities to use the previously described techniques to create a system which recommends music to groups. It starts with existing approaches and how they are implemented. The following sections describe the procedure from a request for a recommendation to the final playlists including the approaches which are developed for the app *M'Usic*. In the last section the developed algorithm's properties are discussed.

In Chapter 4 the developed system, which should agree with the defined aims, is presented. It is divided into the sections *functionality*, where the use cases of the system are explained, the section *architecture*, where the components of the systems and their connections are described, the section *data source*, where the handled data is explained, and finally the section *testing and monitoring*, where the development approach is discussed in detail.

Chapter 5 presents the algorithm's evaluation. It starts with the presentation of the evaluation's environment, where the used technique, the study participants, the execution environment, and the questions are introduced, it continues with the description of the study execution and finally presents the evaluation's results in detail.

Chapter 6 consists of the résumé of the study including positive and negative aspects of the music recommender system for groups discovered during the development of the thesis, and additionally, includes thoughts about issues for future work.

2. Techniques for Recommender Systems

In this chapter techniques for RSs are described. It starts with the basic recommendation approaches *Collaborative Filtering* 2.1 (see Goldberg et al., 1992), *Content-Based Filtering* 2.2 (see Pazzani and Billsus, 1997), and *Context-Aware Recommender Systems* 2.3 (see Gediminas Adomavicius, Sankaranarayanan, et al., 2005) describing different possibilities to find preferences with the given information and trying to find items according to the preferences. In *Data Mining* 2.4, supporting techniques for RSs are introduced. The section *Music Recommender Systems* 2.5 is related to the singularity of recommending music. In *Evaluating Recommender Systems* 2.6, techniques for evaluating RSs and attributes which play a role in an evaluation are described. Finally, the Section *Group Recommender Systems* 2.7 explains the case of recommending items to groups including the aggregation challenge, special techniques for group recommendations and additional appraisal criteria on evaluating GRSs. This chapter should give an insight and at the same time an overview of recommending items and it should create the basis for recommending music to groups, which is explained in detail in the next chapter.

2.1. Collaborative Filtering

Collaborative filtering (CF) is a recommendation technique based on the assumption that there are users with the same preferences on items. In terms of music, this means that if both, user A and user B, have liked the same songs in the past, and then user A likes a new song, the RS suggests user B this song subsequently. This is one of the most popular technologies used for RSs. On the one hand, pure CF can be implemented without the knowledge of any information of the item. On the other hand, item information can be used to improve the recommendation. These types of systems have been researched over the last twenty years and so their advantages, performances, and limitations are well-known. The result of a collaborative recommendation is either a prediction rating of a specific item or a list of recommended items. Jannach, 2011, p. 13

CF is widely known from amazon.com[1], where suggestions are made, based on products which other selected customers have bought. These customers are the ones who bought similar products as the users in the past.

The background data to calculate a recommendation are a set of users and a set of items. Mostly, the connection between these two groups is expressed by terms of ratings. These ratings have been made by users in the past. As a matter of course, not every connection between users and items is rated. The CF RS tries to predict the unknown rating of a user for a specific item I_a. The first step in this prediction is to find the "nearest neighbors" which are the ones with similar ratings compared to the user. The ratings of the similar users for the item I_a are used to predict the rating of the user. Felfernig, Jeran, et al., 2014, pp. 2–3

CF is typically separated in the two categories, such as user-based CF and item-based CF. The aim in user-based CF is to find users who have correlating preferences to a specific user (so called "nearest neighbors" of the user). The proposed items are the ones the neighbors has liked. Whereas in item-based CF, the intention is to find items with the same ratings ("neighbors" of the item). In both cases the main task is to measure the similarity of users or items. Y. Cai et al., 2014, p. 766

2.1.1. User-Based Nearest Neighbor Recommendation

As mentioned above, the aim is to find the *nearest neighbors* of the user who has had similar preferences in the past. The RS tries to find the rating for each unrated item according to the *nearest neighbors*. The basic assumption

[1]https://www.amazon.com

of this method is that the preferences of users do not change over time. Jannach, 2011, pp. 13–18

Jannach (2011, p. 14) calculates the similarity between user $\mathrm{U_a}$ and another user with the Pearson correlation coefficient shown in Formula 2.1.

$$similarity(U_a, U_x) = \frac{\sum_{i \in I}(r_{a,i} - \overline{r_a})(r_{b,i} - \overline{r_b})}{\sqrt{\sum_{i \in I}(r_{a,i} - \overline{r_a})^2}\sqrt{\sum_{i \in I}(r_{b,i} - \overline{r_b})^2}} \tag{2.1}$$

In this formula, I is the set of items both users have rated. $r_{a,i}$ is the rating of user U_a for the item i and $\overline{r_a}$ is the average rating of user U_a. The result of this formula is a value between -1 and 1. The higher the result, the more similar are the preferences of the users. This leads to the *nearest neighbors*. Felfernig, Jeran, et al. (2014, p. 4) assume that at least two items per user pair should be rated to get a convincing value of similarity.

To get the predicted rating for an *item* of the user U_a, only *nearest neighbors (NN)* with a rating for this item can be used. With the ratings of these users, Jannach (2011, p. 16) tries to predict the rating with the following Formula 2.2.

$$prediction(U_a, item) = \overline{r_a} + \frac{\sum_{U_j \in NN} similarity(U_a, U_j) \times (r_{j,item} - \overline{r_j})}{\sum_{U_j \in NN} similarity(U_a, U_j)} \tag{2.2}$$

The highest prediction of an item is the one to be recommended.

2.1.2. Item-Based Nearest Neighbor Recommendation

User-based CF was widely used in e-commerce as the most successful technology for building RSs. But with the growth of the world wide web and its millions of users, unfortunately the computational complexity of user-based CF methods grew linearly with the number of customers. This lead to the development of scalable item-based recommendation algorithms.

In such algorithms, the first step is to find similarities between items, and then to identify the set of items recommended to the user. Deshpande and Karypis, 2004, p. 143

Jannach (2011, p. 19) used the *adjusted cosine* measure to calculate the similarity shown in Formula 2.3.

$$similarity(I_a, I_x) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_b})^2}} \tag{2.3}$$

U is the set of all users that have rated both the items. $r_{u,a}$ is the rating of user u to item a, and $\overline{r_u}$ is the average rating of user u. As in the Pearson correlation coefficient, the result of this formula can be between -1 and 1.

To predict the rating for user U_a for an *item*, Jannach (2011, p. 20) calculates as follows:

$$prediction(U_a, item) = \frac{\sum_{i \in NN} similarity(item, i) \times (r_{a,i}}{\sum_{i \in NN} similarity(item, i)} \tag{2.4}$$

The set of items with a similar rating pattern compared to the *item* are the *nearest neighbors (NN)* which are the basis for the prediction of the rating. In this case *NN* only represents items which have already been rated by user U_a. Felfernig, Jeran, et al., 2014, pp. 5–6

2.2. Content-Based Filtering

Content-based filterings (CBFs) are RSs with one of the most successful outcome. The similarity between items are calculated based on their contents. The item information is represented as attributes. To recommend items, the CBF uses the description of items and the user's preferences. For each active user the CBF employs a personal profile to generate accurate recommendations. Through that there is always a result, even if there are no ratings of other users. The disadvantages of CBFs are the necessity of ratings of the current user and the high effort to extract the information of an item. Son and Kim, 2017, p. 404

Such RSs can recommend items only with the description of the item characteristics and a description of the interests of a user (*user profile*) in terms of preferred item characteristics. The suggested items are the ones with the best match to the user's preferences. In practice, there are two types of attributes. On the one hand, *technical* descriptions such as the list of actors in a movie or the genre of a music track which are widely available. On the other hand, there are subjective *qualitative* features. For example, there is not always a connection between the items that a user should like based on their characteristics and those that a user actually likes. Such information is difficult to obtain, and the expenditure is often not affordable. Typical examples of CBFs recommend text documents because the extraction of the content and its description is easily automatically realizable. Jannach, 2011, pp. 51–79

In terms of music, CBFs try to find similarities in tracks the user has rated high in the past such as the genre or the interpreter, and recommend songs with these attributes. This also shows the problem of this type of RSs: the overspecialization. A user who always listens to the same genre would never get a recommendation of an other. A way to solve this issue is to implement functions to recommend random items of other genres. Furthermore, items which are too similar to others in the recommendations should be ignored. For example, if a recommended song is available as an album version and as a radio edit, only one should be in the final recommendation list. G. Adomavicius and Tuzhilin, 2005, pp. 735–737

2.3. Context-Aware Recommender Systems

Both already introduced types of RSs have in common that only users and items are observed to recommend items. Context-aware recommender systems (CARSs) include, in addition, the context to provide recommendations. This allows to vary the recommendations depending under certain *circumstances*. For example, songs recommended for playing during working time should be other ones than playing at a party. That requires to gain *contextual information* to improve the recommendation process. Such information in terms of music can be the mood of the user like “calm”, “dark”, “energetic”, “positive”, etc. In this case, CARSs only recommend songs according to the chosen mood. G. Adomavicius and Tuzhilin, 2015, pp. 191–192

2.3.1. Context in Recommender Systems

Context, in term of RSs, are circumstances which affect the preferences of the users. Contexts need to have a structure. For example, the location in the user’s country can be classified. This is needed to categorize items in these *contextual factors*. One task of the CARS is to define such factors. Contextual factors can be classified into three categories, such as *fully observable*, *partially observable* and *unobservable*. Factors are fully observable if all information about the context, which are needed for the recommendation, is available at this time. They are partially observable when only a part of the information is known explicitly. And, if factors are unobservable, there is no information of the context available. If there are partially unobservable factors, they can be predicted with methods like hidden Markov models. For unobservable factors, the only way to model context is by using latent variables. Another way to categorize contextual factors is to classify their importance or their structure for a certain period of time. This means that contextual factors also can be divided in *static* and *dynamic* factors. Static factors are stable over the lifetime of a RS. For example, genres such as rock, pop, etc., do always have the same structure, now and in the future. Dynamic factors can change their structure. For example, the RS realize that the structure of the time should change from weekday and weekend to the time of the day (morning, midday, afternoon, evening and night). Gediminas Adomavicius, Mobasher, et al., 2011, pp. 67–69

2.3.2. Representing Contextual Information in Recommender Systems

Rating-based algorithms typically use existing ratings to predict how a user votes an item. The rating function R is defined as follows:

$$R : User \times Item \rightarrow Rating \tag{2.5}$$

As soon as the function *R* is defined, the RS can predict all unknown ratings. These systems are called *two-dimensional* because only *Users* and *items* are considered as dimensions. Rating-Based CARSs add contextual attributes to this function. They affect the ratings and are defined as sets. The precondition is the knowledge of the contextual information. The function *R* could be represented as following:

$$R : User \times Item \times Context \rightarrow Rating \tag{2.6}$$

Context in this case defines the application's associated known contextual information. This leads to a multidimensional contextual model, because each contextual attribute is represented as a single dimension. Gediminas Adomavicius, Mobasher, et al., 2011, pp. 70–71

2.3.3. Integrating Contextual Information into Recommender Systems

Basically, there are two ways of using contextual information in RSs. *Context-driven querying and search* is used widely in travel/tourist RSs, for example to recommend near restaurants. And secondly, there are CARSs that use *contextual preference elicitation and estimation*. On the contrary to the first approach, the second mentioned system try to model and learn preferences of the user based on contextual information. For two-dimensional RSs, the recommendation process can be split in three parts. The first part is the data part which represents the user preferences as its *input*. The second component is the two-dimensional RS called *function*, and the third component is the recommendation list called *output*. CARSs based on *Contextual preference elicitation and estimation* influence one of these components. Depending on the component, the three forms are called:

- *Contextual pre-filtering*. In this case, only relevant data according to the contextual information is collected. The rest of the recommendation process is unchanged to two-dimensional RSs.
- *Contextual post-filtering*. The process starts like a two-dimensional RS and results in a set of recommendations. The contextual information is afterwards used to adjust the result set.
- *Contextual modeling*. In this process, the two-dimensional recommendation function is exchanged by a multi-dimensional recommendation function to include the contextual information in the rating estimation.

Techniques, which are based on contextual user preferences, observe users and their interaction with the system to figure out the user's preferences and model them. This context-sensitive preferences and the resulting recommendations are calculated either by adopting CF, or CBF methods or by intelligent techniques to analyze data from machine learning or data mining. G. Adomavicius and Tuzhilin, 2015, pp. 206–209

2.4. Data Mining

Due to the exploding amount of data available for RSs, it is getting more and more important to find techniques to deal with this. The aim is to exploit the information and model it. The models can be used for learning preference models. Gemmis et al., 2009

RSs tend to adopt techniques of neighboring areas. To handle the amount of data, these systems use algorithms stemming from the field of data mining. The steps to get preference models out of data are called *Data Preprocessing*, *Model Learning* and *Result Interpretation*. Amatriain and Pujol, 2015, p. 227

2.4.1. Data Preprocessing

Before starting to preprocess data, the task is to *gather user feedback*. Feedback can happen in two ways. Either as *Explicit Rating* where the user rates an item on a defined range, or *Implicit Rating*. Implicit feedback is gathered passively by monitoring the user interactions in a specific environment with the system. In order to get implicit ratings, the user interests from the feedback must be estimated on a value in a predefined scale. Gemmis et al., 2009, pp. 8–10

Due to the amount of dimensions and scales of the explicit and implicit ratings, in real-life situations the data has to be preprocessed to match the prerequisites of machine learning techniques. Preprocessing techniques are, for example, *similarity measures* or *dimensionality reducing* and, in this area, especially the *Matrix Factorization and Singular Value Decomposition*. Amatriain and Pujol, 2015, p. 229

Similarity Measures

Similarity measurement is used to calculate the distance between two items or two users. It is more a tool than a step in preprocessing data. There are several methods to calculate the similarity and the following are the most common ones:

- *Euclidean distance*. This is probably the most common and simplest example.

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{2.7}$$

- *Cosine similarity*. The cosine of the angle of two vectors, which is representing the items in an n-dimensional space, describes the similarity of two objects.

$$cos(x,y) = \frac{(x \bullet y)}{\|x\|\|y\|} \tag{2.8}$$

- *Pearson correlation*. The Pearson correlation measures the similarity of ratings by removing the effects of mean and variance.

$$PC(U_a, U_x) = \frac{\sum_{i \in I}(r_{a,i} - \overline{r_a})(r_{b,i} - \overline{r_b})}{\sqrt{\sum_{i \in I}(r_{a,i} - \overline{r_a})^2}\sqrt{\sum_{i \in I}(r_{b,i} - \overline{r_b})^2}} \tag{2.9}$$

It depends on many aspects which similarity measure function performs best, and additionally, differences are often not measurable. Amatriain and Pujol, 2015, pp. 229–230

Matrix Factorization and Singular Value Decomposition

Reducing dimensionality is used when the data set includes too many features leading to a high-dimensional space combined with a limited number of values of this features per object. This usually happens when explicit feedback is not, or hardly available, and implicit feedback is used to get a user's preferences. Implicit ratings are commonly represented as a dense matrix. Y. Koren, Bell, and Volinsky, 2009, pp. 31–32

CF systems compare users and items to achieve recommendations. This can be done in two different ways. The classical *neighborhood approach* where relationships between users or between items are computed based on the ratings. The alternative approach is the *latent factor model*, such as Singular Value Decomposition (SVD), where items and users transformed to the same latent factor space to make them comparable. The entries in the latent space represents the rating calculated from the factors of the user feedback. In the field of music recommendation, possible factors for dimensions can be obvious, like *genre* or *interpret* up to total uninterpretable dimensions. Yehuda Koren, 2008, p. 426

Matrix factorization is the base for many, very successful, latent factor model realizations. The mapping of users and items in an f-dimensional latent factor space allows the matrix factorization to model the user-item interactions as inner products in the space. The vectors for the item i and the user u in the space are $q_i \in \mathbb{R}^f$ and $p_u \in \mathbb{R}^f$. The vector q_i shows the match to all factors both positive and negative and the vector p_u indicates the preferences of the user to these factors. The dot product of these two vectors represents the interaction between the item and the user, and is defined as $q_i^T p_u$. Predicted values are notated as $\hat{r}_{ui}$, known rating as r_{ui}. The hardest part is to compute all mappings between p_u and q_i. Afterwards, the estimated ratings can be calculated with the previously mentioned Equation 2.10. Y. Koren, Bell, and Volinsky, 2009, p. 32

$$\hat{r}_{ui} = q_i^T p_u \tag{2.10}$$

The described model is very similar to SVD. In the domain of CF, the SVD factorizes the user-item matrix. As in conventional SVD missing information in the matrix leads to an undefined status, earlier systems made assumptions accompanied by a heavy increase of the amount of data and the possibility of distortion of the underlying data. For this reason, current works suggest focusing on the observed data only. These systems learn the factor vectors (q_i and p_u) by minimizing the regularized squared error of the known ratings:

$$\min_{p*,q*} \sum_{(u,i)\in\mathcal{K}} (r_(ui) - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.11}$$

$\mathcal{K}$ is the set of user-item pairs of which the rating is known. This is seen as the training set. λ is a constant to control the extension of regularization. It is usually determined by cross-validation. Y. Koren, Bell, and Volinsky, 2009, p. 32

Matrix factorization contains the benefit of including different aspects into the system. This allows to add biases in the calculation. *Biases* are characteristics of users or items. For example, the average rating of users differs. The approximate formula for the bias b_{ui} of the rating r_{ui} is as follows:

$$b_{ui} = \mu + b_i + b_u \tag{2.12}$$

μ represents the overall average rating, b_u and b_i are the deviation of the item i and user u from the average. Including the biases in the formula to minimize the squared error function comes to following:

$$\min_{p*,q*,b*} \sum_{(u,i)\in\mathcal{K}} (r_(ui) - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2) \tag{2.13}$$

Such systems allow to decrease the observed signal to get a vital accurate model. Y. Koren, Bell, and Volinsky, 2009, p. 33

2.4.2. Model Learning

Learning user profiles can be categorized in two different types: Offline and online learning techniques. In *online learning* the model is build online and updated frequently to achieve real-time recommendations. *Offline learning* builds a model once and can be used in domains with slowly changing user preferences compared to the time the model building takes. User profiles are typically learned with machine learning techniques. Such a technique is used to build a *classifier* by learning from a training set to label items. For example, with c_+ and c_- to categorize items in user-likes and user-dislikes. Machine learning strategies are for example probabilistic algorithms, neural networks, decision trees and nearest neighbor algorithms. Gemmis et al., 2009, pp. 10–11

Naïve Bayes

Naïve Bayes is a probabilistic method for classification. It is usually used for recommending textual documents. The classifier determines the probability that an item i belongs to a class C_j given feature values of the item. In the field of textual documents, this represents the probability that a user is interested in a text which contains or does not contains specific words:

$$P(class_j \mid word_1 \& word_2 \& \ldots \& word_n) \tag{2.14}$$

$word_{1\ldots n}$ stand for Boolean values if the word appears in the text or not. Bayes Classifier take the assumption that the appearance of words in texts are independent events given the class of the text. This can be expressed as follows:

$$\prod_{i}^{n} P(word_i \mid class_j) \tag{2.15}$$

This leads to the probability of an item given $class_j$ is proportional to:

$$P(class_j)\prod_{i}^{n} P(word_i \mid class_j) \tag{2.16}$$

The item is assigned to the class with the highest probability. Even though the assumption of independent attributes in the field of text classification is completely unrealistic, the performance of Naïve Bayes is very well. Billsus and Pazzani, 1996, pp. 237–238

Decision Trees

Decision Trees are methods to classify attributes by making decisions in the path of a tree. The *non-leaf nodes* test a single attribute-value to determine in which sub-tree the classification continues until a *leaf node* is reached. They indicate the class to assign the item. Concrete algorithms like ID3 have features to deal with noise or unknown attribute values. Quinlan, 1986

Decision trees are implemented very easily but are not very effective in the field of RS. Although in combination with other techniques decision trees can improve recommendations efficiently or accurately. Jannach, 2011, pp. 70–72

Nearest Neighbors

The *Nearest neighbor classifier* (*k*NN) is an instance-based classifier which needs training records to predict class labels of new cases. For a new point to classify the *k*NN looks for the *k* closest points of the training set. The new point gets the class label of the class with the largest number of points in this neighborhood. The trickiest part is to choose the size of *k*. A too small size is sensitive to noise points, a too big size can include too many points from another class. Cover and Hart, 1967

The results of *k*NN are quite accurate and the algorithm is still simple and intuitive. For a long time the *k*NN was the de facto standard in CF, but is challenged by Matrix factorization nowadays because it is not protected against bias and relatively expensive in classifying new records. Amatriain and Pujol, 2015, pp. 237–238

Support Vector Machines

Support vector machines (SVMs) try to find linear hyperplanes with a maximum margin to separate data in two classes. The basic idea behind SVM is that the maximum margin decreases the probability of misclassification of unknown items. Cristianini, Shawe-Taylor, et al., 2000

Clustering

Clustering is dedicated to the category of unsupervised learning. It combines the strength of different techniques to recommend closely related individuals associated to a group. The use of clusters which represent the computed groups, enables to select the closest clusters instead of the nearest neighbors and so make the system scalable. Moreover, it allows to predict unrated items of a user by using the rating information of the user's group. A popular implementation of clustering is the K-means algorithm, where the K defines the desired number of clusters. Xue et al., 2005, pp. 114–116

2.4.3. Result Evaluation

Widely accepted as an evaluation of a RS is the *mean absolute error (MAE)*. It measures the difference between the predicted and the users actual rating. The formula to calculate is:

$$|\overline{E}| = \frac{\sum_{i=1}^{N} |p_i - r_i|}{N} \tag{2.17}$$

This allows to compare the significance of a difference between the MAEs of two RSs by well-studied statistical properties. Root mean squared error (RMSE) is related to the MAE and emphasizes the difference between the predicted and the actual rating. Larger errors result in a higher RMSE-value. Herlocker et al., 2004, pp. 20–21

There are *classification accuracy metrics* to measure the correctness of decisions a classifier makes. Such metrics do not measure the predicted rating, but rather qualify a classification as correct if the item has assigned to the correct class. In a RS where items can be classified in relevant and not-relevant, and the RS select several items as recommendation, the *precision* is defined as follows:

$$P = \frac{N_{rs}}{N_s} \tag{2.18}$$

The result of this formula is the probability that a recommended item is relevant. Whereas *recall* is the probability, recommendation contains a relevant item:

$$R = \frac{N_{rs}}{N_r} \tag{2.19}$$

Depending on the separation of the items in relevant and non-relevant, the precision and recall value can be different. Herlocker et al., 2004, pp. 22-24

2.5. Music Recommender Systems

To enable users of online music services the experience to have a huge amount of titles under control, it is necessary to provide music recommender systems. The unique characteristic of recommending music is that consuming one track is very short, additionally, a user can decide within seconds whether a song is satisfying. Furthermore, songs are listened repeatedly and typically recommendations contain a set of tracks where many of them are listened to. Due to the low number of ratings per song, a music recommender tends to use content-based and context-based techniques. A. Schedl et al., 2015, pp. 453–455

In the following sections techniques for music recommendations are described, and, additionally, ways to order the recommended tracks as playlists are mentioned.

2.5.1. Content-Based Music Recommendation

With the availability of millions of songs in internet stores it becomes important to offer RSs dealing with this. The problem in CF is that the number of rated songs by a user compared to the available songs is extremely small. This leads to recommend items only out of a small popular track set and fewer known songs are ignored. Instead, content-based music recommendation (CBMR) recommend the full set of available tracks by finding similar songs compared to a user-model. D. Bogdanov, Haro, and Fuhrmann, 2011, p. 249

This leads to another disadvantage of CF, the so called cold-start problem. New items are not part of the possible recommended items until there are ratings for the item. And non-recommended items are less likely to get rated. McFee, Barrington, and Lanckriet, 2012, p. 2207

Content Creation

In CBMR the first step is to create structured meta-data out of the unstructured music-source. One way is to extract automatically semantic information and relations from music-sources based on arbitrary textual sources. The relation between entities can be, for example, the track, artist or genre. Especially in the domain of music the number of possible relations is very high. In web-based sources, containing meta-data and the song-text, it is possible to create models of the items with the help of linguistic features. Knees and M. Schedl, 2011, pp. 18–19

Another way to extract information out of music is to analyze the audio signal. This allows to get timbral, temporal, and tonal features as high-level semantic descriptions. With the use of classification tasks, it is possible to get information about rhythm, tempo, instrumentation, genres, musical culture and moods. This enables to create a model for recommendation.D. Bogdanov, Haro, and Fuhrmann, 2011, pp. 249–250

On this approach Soleymani et al. (2015) developed a system including five attributes known as Mellow, Unpretentious, Sophisticated, Intense and Contemporary (*MUSIC*) which results are much better compared to CF recommendations when no user ratings are available and this increases the possibility that songs, that are less popular but appealing to the user will be recommended.

Preference Set

To create a user-model, it is necessary to know the preferences of a user. There are different ways to get this information. Either the user can create a preference set with several tracks representing the taste of the user. Or the preferences of a user's most popular songs can be get out of the ratings a user made on tracks. Another way is to simply take a user's top tracks which could be the most played tracks in a music system, or the one marked as favorites. D. Bogdanov, Haro, and Fuhrmann, 2011, pp. 249–250

Model Building

The recommendation task itself contains two steps. The first step is to create a user model and subsequently finding similar songs to this user model. There are several ways modeling the preferred recommendation outcome. One way is to create a model as the mean point of all items in the preference set. The recommended songs are the nearest to this point. Another way is, that every item of the preference set is a single model and the recommendation would be the item closest to one of these models. A third way is to build clusters with a Gaussian mixture model using the preference set as the training set. Clusters containing more training items are bigger, and so the probability a recommended item is part of this clusters is higher. Dmitry Bogdanov et al., 2013, pp. 19–21

Similarity Measurement

To calculate the similarity the model and each track are represented as a d-dimensional vector $\in \mathbb{R}^d$. The distance between the model and the single tracks can be calculated for example with the *Euclidean distance* which is defined in Formula 2.7. Techniques like SVMs allow to optimize the performance of similarity measurement. McFee, Barrington, and Lanckriet, 2012, pp. 2207–2210

2.5.2. Contextual Music Recommendation

As listening to music can influence the emotion of a user, the context can play a big role in recommending music. R. Cai et al., 2007, p. 553

Although in CARSs there are many different possibilities to categorize the context, when dealing with music it makes sense to categorize factors which are in relation to the environment and to the user. A. Schedl et al., 2015, p. 460

Examples for environmental context are weather and time, or situation based factors like a landscape, a road type or traffic conditions when driving a car. Examples of user-related context are mood, activities, or social context. Some of these factors have more and some of these have less influence on the recommendation. The main disadvantage in CARSs is the fact that ratings in different contextual situations are necessary to get the correlation between the context and the item. Furthermore, it is vague if different factors have side effects to each other. A methodology for CARSs could comprises four steps: *context factors relevance assessment*, where the importance of each factor is analyzed, and the relevance defined. The *in-context acquisition of ratings*, where users rate tracks under a defined context. *Context-aware rating prediction*, where missing ratings of items under several contextual situations are predicted and finally the *context-aware recommendation generation*, where the user gets a recommendation under a specific contextual situation. Baltrunas et al., 2011, pp. 89–90

Contextual music recommendation can also be seen as an improvement of existing CBF. To achieve improvements through contextual information, it is necessary to gain the information in different ways, such as emotion detection/recognition, emotion description, and low-level feature-based classification like energy or the audio spectrum. *Emotion representation* assumes that every emotion can be mapped to a set of numbers. This allows to update the user-model depending on the contextual situation. Han et al., 2010, p. 434

Thayer (1990), for example, defined a model of emotions with the two dimensions *arousal* and *valence* which allows to divide the model in four quadrants including eleven emotions. Felfernig, Stettinger, et al. (2018, pp. 157–159) present different models of emotions.

A third way of recommending music with contextual information is to label music with context attributes. This approach is cost-intensive since tracks have to be labeled. Furthermore, the results show no significant improvements. Ankolekar and Sandholm, 2011

2.5.3. Hybrid Music Recommendation

As shown in the previous sections, each recommendation technique has its pros and cons. CF outperforms CBF but has the disadvantage of the cold-start problem. CARS need height effort to get the underlying data. In this sense, the next step in RS is to develop hybrid systems combining the pros of two or more RSs and reduce the weaknesses of them. A. Schedl et al., 2015, p. 465

Music recommender systems are successful when they consider different aspects such as music content, context and user-context. The research in music information retrieval (MIR) is widely explored and enables applications such as semantic music search engines, music recommender systems, or automated playlist generation systems. MIR allows to identify correlations between the perception of music and contextual information beyond the audio signal of tracks. Due to the subjective perception of music, it also depends on user-specific factors. User-aware music recommender systems consider user context and user properties. They differ in the changing characteristics. User properties like age or genre preferences slowly change whereas user contexts such as current activity or mood change dynamically. M. Schedl, 2013, pp. 3–4

The music perception can be defined as the dependency of four categories: *User context* can include the mood, social context, or physiological aspects of the user. Music preferences, musical experience, or demographics are aspects of *user properties*. *Music content* in the form of rhythm, timbre, melody, harmony, and loudness. And finally the category *music context* contains song lyrics, artist's background, semantic labels or music video clips. M. Schedl, Flexer, and Urbano, 2013, p. 526

As discussed in Burke (2002, pp. 339–344), there are seven ways of combining recommendations in a hybrid fashion that has been developed so far: The *weighted* method combines the scores of several RSs to a single recommendation. *switching* is defined as the dependency of the selected recommendation on the current situation. *Mixed* hybrid recommendations shows the recommendations of each RS at the same time. *Feature combination* combines the approaches of different RSs taking several data sources as input. In the *Cascade* mode one RS refines the recommendation of another RS. This is not to be confused with *feature augmentation*, where an input feature of a RS is the recommendation of another RS. Finally, the *meta-level* method uses one recommender to define a model which is the basis for another RS.

2.5.4. Automatic Playlist Generation

As tracks are consumed in a row, an improvement of recommending music is the implementation of a playlist generation. Typically, users listen to music in a sequence order and the first track is consumed immediately after the recommendation is finished. As music can influence the current emotion of a user, the change of the mood should be considered in the selection of the following tracks. An approach to consider the different music moods is trying to find a smooth transition between tracks. This can be achieved by taking the current track and compare the set of candidates to pick the one with the smallest Euclidean distance. The main disadvantage is the contemplation of only two songs. Another approach is *frequent pattern* like association rule and sequential pattern where the association between item sets are considered. Either it is possible to consider playlists as users. The ranking represents the track order. Content-based approaches use additional information such as lyrics, genre, user tags, or musical features to find the nearest neighbors. It is hard to evaluate such playlists, but possible strategies are human opinion surveys, comparison to hand-crafted playlists or semantic cohesion. Bonnin and Jannach, 2013

2.6. Evaluating Recommender Systems

In this section, techniques to evaluate RSs are described. It presents different ways how evaluations can be carried out, and it shows parameters which can be investigated to measure evaluations.

2.6.1. Experimental Settings

There are several environments to evaluate a RS. From easy to manage offline testing, to an executed user study under a controlled setting, to online testing with a big set of participants different variants are possible. All three follow the same basic approach: The first step is to form a *hypothesis*. For example, algorithm A gives better recommendations than algorithm B. So only the prediction accuracy should be tested. The next step is to define *controlling variables* to check that the preconditions of evaluating two different algorithms are the same. For example: Are the data set and the influencing context the same in all comparing tests? To persist the result against other evaluations, the experiment needs *generalization power*. This means that the preconditions must be defined in such a way that the algorithm produces the same results in the testing environment and in real situations. It is not valuable to create a data set to gain the perfect result for the preferred algorithm in the test but recommend incorrect items in the real application. Gunawardana and Shani, 2015, pp. 267–268

Offline Experiments

For offline experiments there is a need of a precollected data set which contains a set of items rated previously by users. With this dataset, offline experiments can be compare the result of the algorithm with the ratings of the data set. This has the benefit that there is no need of test users during the tests and it allows a cost-effective comparison of different algorithms. The disadvantage of offline experiments is the restricted amount of testable hypotheses. Furthermore, the fact that ratings differ caused by the way of presenting the recommendations is not taken into consideration. So offline experiments are used to pick a small set of possible algorithm candidates to make further tests with user studies or an evaluation with the actual conditions. Gunawardana and Shani, 2015, pp. 268–269

User Studies

Due to the fact that many applications including RSs are based on interactions with users, offline tests are limited. User studies offers the possibility to evaluate the system including the interaction with users, which allows to not only gain information about the accuracy of the recommended items, but also information about the performance or the user experience can be collected. To conduct a user study, a set of test users is asked to perform several previously defined tasks. During the execution, participants were observed to collect different quantitative measurements. Before, during, and after each task, the users were asked to answer several qualitative questions not directly observable during the tasks. A huge amount of information can be collected during the evaluation. But there are also disadvantages, like the high effort. The evaluation takes a lot of time and the analysis of the collected data is demanding. This typically leads to a small number of testing subjects. Furthermore, pilot user studies were made to avoid failing experiments, as the failure of the application under specific circumstances. The test subjects should represent the users of the application as close as possible. Nevertheless, there is the possibility of a bias of the results caused by the knowledge of the subjects taking part in an experiment. It is possible to let the users test all variations of the system called *within subjects* or let each user test only one variation to compare *between subjects*. When preparing the questions, there is to contemplate that the kind of the question can influence the answer. It is important to ask neutral questions avoiding the suggestion of "correct" answers. It is also important to guarantee that the answers are treated as confidential and anonymous, and convince the test group that there are no wrong answers. Gunawardana and Shani, 2015, pp. 271–274

Online Evaluation

To test different variants of RSs in real situations, online evaluation offers the possibility to test different algorithms by providing them to different users. All users use the same application, only the underlying system is divers, and so users are not able to differentiate. To test the interface, it is the other way around, the underlying algorithm is the same in every version, only the interface changes. This is necessary to get a comparable evaluation result. For these reasons it is a very popular evaluation approach in productive systems. The circumstance that the evaluation participants are real users, makes it mandatory to test such algorithms in offline experiments and even in user studies, because a bad behaviour of the system can lead to lose customers. Nevertheless, online test evaluation provides results closest to real world situations with the highest amount of test subjects and allow to test over a longer period to evaluate the changes in time. Due to the excessive effort, which is necessary to bring a variant of the system to the stage of online evaluation, only few variants were typically evaluated. Gunawardana and Shani, 2015, pp. 274–275

2.6.2. Recommender System Properties

When testing different RSs, it is necessary to define which properties are important, and to rank them. There is no general solution, it depends on various factors such as the kind of recommended items or the goal behind the recommender. Some properties can have a negative impact on others, like accuracy on diversity where a higher focus on accuracy can result in more similar results. To find the correct ratio of such properties, online tests can help to find a good balance between the properties. In such online tests, there typically is a basic recommendation algorithm including tunable parameters that can effect certain properties allowing to find an appropriate configuration. Concerning a property, it is important that on evaluation the participant does not recognize the intentions especially in the questions. Gunawardana and Shani, 2015, p. 280

User Preferences

The simplest way to evaluate different RSs is to let the subject choose one of the systems. Such tests do not investigate a specific evaluation property. The advantage is that users would rather make decisions between two opportunities than rating a system. The disadvantage is that the reason for the decision is not observable. The user's opinion as well is uncertain about the inferior system. In this way it is possible that one system beats another, but the acceptance for the loosing system is higher. The possibility that not every user's decision is valuable for the evaluation has to be taken into account. The missing of information why one system is better than the other makes it impossible to improve the system in such a way. The solution is to break the components of the system into smaller pieces, to get more detailed information and better understand the system. Gunawardana and Shani, 2015, pp. 280–281

Prediction Accuracy

The most discussed property in RSs is prediction accuracy. It is also the property which is easiest to compare and to bring it into a countable form. The common opinion is that a better prediction accuracy leads to a better user acceptance. It is, furthermore, independent of the user interface and is therefore testable in offline experiments. It can be measured in different ways like *accuracy of ratings prediction*, shown in Section 2.4.3, the *accuracy of usage prediction*, which comes into effect when RSs are not trying to predict the value of the rating, but tries to make recommendation getting chosen by the user, and finally the *accuracy of rankings of items*, concerning the order of the recommendation. Gunawardana and Shani, 2015, pp. 281–285

Coverage

Coverage can be *item space coverage* defining the recommendable items of a RS or it can be *user space coverage* describing the number of users the RS can provide accurate recommendations. Especially CF systems have problems with coverage because items can be unrated, or users have not rated items.

Due to the increasing amount of ratings, the coverage of items and users rise over time. Related to the coverage is the *cold-start problem*, where new user or items without any information cannot be included in the recommendation process. Gunawardana and Shani, 2015, pp. 292–293

Confidence

In the field of RSs, confidence is the probability that a rating for an item is correct. With knowing a recommendation's confidence, the user can collect further information for recommended items with low confidence. This can lead to a higher user satisfaction. Gunawardana and Shani, 2015, p. 294

Trust

When many users belief in the results of a RS, it has a high value of trust. A user gets confident to a system by getting repeatedly good recommendations over time. This can be reached besides good recommendations through recommending items; a user has already rated high. A user with a high value of trust in a RS tends to accept even not so accurate recommendations. Gunawardana and Shani, 2015, pp. 295–296

Novelty

Recommended items, which are new to a user, are novel recommendations. An increase of novelty can be achieved through filtering out known items by the user, for example the ones which where rated already. Nevertheless, it can not be guaranted that items are new for users since it can be consumed on other platforms before. Only user studies provide the possibility to check if an item is new to a user. Note that novelty is independent from accuracy, following that it always has to be evaluated in combination with the relevance of items. Gunawardana and Shani, 2015, p. 296

Serendipity

Serendipity is the combination of a surprising and successful recommendation. A recommendation is surprising if it is unlikely that the user will find this item on his/her own. For example, when the user is unaware of a track's artist, or the track is in a genre, the user is usually not listening to. As surprising itself is not hard to achieve, it has to be combined with accuracy to reach serendipity. It can be described as the amount of new and useful information. Defining a recommended item as serendipitous is not as easy because the user tends to follow unknown recommended items. To be sure that the item is appropriate, there has to be an observation over a certain period of time. Gunawardana and Shani, 2015, pp. 297–298

Diversity

Diversity describes how different recommended items are. It can be defined as the opposite of similarity. There are several domains for RSs where it is important to provide divers recommendations. For example, on RSs recommending holiday destinations it is not helpful to recommend several cities in the same country. Higher diversity can decrease the value of accuracy, so a balance between both has to be found. Diversity can be measured with item-item similarity like in Section 2.4.1. Gunawardana and Shani, 2015, p. 299

Utility

The utility of a RS has to be defined in an early state of the development. Examples are the increase of the revenue in an e-commerce system, or an extended user experience. Utility can either be seen as the value the user or the system gets from a recommendation. Gunawardana and Shani, 2015, p. 300

Risk

Some RSs work in a domain where it is important to know how risky a recommendation is. Like in the recommendation of stocks. In addition to the possible profit, it is necessary to know the risk of failure. Gunawardana and Shani, 2015, p. 301

Robustness

Robustness can be seen as the system's defense against all variants of manipulations. Manipulation of data can be done by fake users making fake ratings, or by attacking the system. Both can have affect on the stability and the data security. Developing a 100 percent robust system is unrealistic and the effort in preventing attacks is often a factor of cost. Gunawardana and Shani, 2015, pp. 301–302

Privacy

A RS is respecting the users privacy as long as it is impossible for a third party to get information about the ratings a user has made. People give RSs preferences about items to get good recommendations but are unwilling to share this information with other people. In certain situations, even recommending items that other users have previously consumed may violate the user's privacy. Implementing arrangements protecting privacy in RSs can affect the accuracy and so evaluations are necessary comparing the system with this implementation and without. Gunawardana and Shani, 2015, p. 302

Adaptivity

The adaptivity of a RS describes how rapidly a system can react on changing conditions. For example, changing preferences of a user or trends. It can also be seen as the ability of a system to consider newly made ratings of a user in recommendations. For a user it is very annoying adding ratings without any changes of the recommendations. Gunawardana and Shani, 2015, p. 303

Scalability

Nowadays, due to the huge amount of data it is getting more and more important to get recommendation immediately after the request. Developer of RSs are focused on the accuracy or coverage and reach good response times on small test sets. When working with real data such systems often do not get the performance as they should, or only with increasing the computation power or memory. Scalability can, for example, be improved by reducing the complexity of a model. A measurable value for scalability can be the amount of recommendations per second or the response time getting a recommendation. Gunawardana and Shani, 2015, pp. 303–304

2.7. Group Recommender Systems

RSs discussed so far are for individual users. But there are also many scenarios with the need of suggesting items to groups. Felfernig, Stettinger, et al., 2018 For group recommendations, existing techniques for recommending items have to be adapted and extended. For this purpose, the preferences of all group members must be aggregated. This section explains different aggregation strategies, recommendation techniques adopted for groups, evaluations factors which additionally have to be considered in GRSs, and, finally, this section is about which social factors affect the accuracy of recommendations in groups.

2.7.1. Preference Aggregation Strategies

To get recommendations for a group, the preferences of the individual members of the group have to be aggregated. There are three different schemas nearly all group recommendation approaches use. There are the *aggregation of the ratings of the users*, the *merging of the recommended items for* *the individual users* and the *construction of a group preference model*. Jameson and Smyth, 2007, p. 605

User Rating Aggregation

For this case the assumption is that each user has rated the items, or at least, the rating can be predicted. The rating for an item is computed by aggregating the ratings of all group members, and the items with the highest ratings are recommended. There are different strategies to calculate the resulting rating which are described later in this section. Jameson and Smyth, 2007, pp. 606–607

Merging Individual User Recommendation Items

In case of recommending a set of items, it is possible to calculate items for each user and it can merge the results to a group recommendation. It is a very simple adoption of recommender systems for individuals. The problem in such approach is that each recommended item is accurate for a single group member. This leads to the possibility that the items are perfect for a single group member, but unacceptable for all of the others. Jameson and Smyth, 2007, pp. 605–606

Construction of Group Preference Models

This approach is not focused on the prediction of the user ratings, instead it generates a preference model for the group based on the individual preferences of the group members. With the help of the group preference model it is possible to predict ratings of items and recommend the items with the highest values. The advantage of this strategy is that different approaches are possible to generate the group model, like distance between preference models. Furthermore, it increases the privacy of users. Due to the recommendations based on a group preference model it is not possible to draw conclusions about the preferences of the single group member. Jameson and Smyth, 2007, pp. 607–608

2.7.2. Aggregation Goals

There are different opportunities to compute the aggregation of the preferences with the previously described general strategies. Each of them intends to achieve a specific goal of user satisfaction. Which kind of aggregation is selected often depends on the given situation. The field of aggregation strategies has reached attention in group recommendation. Some approaches are discussed as follows. Jameson and Smyth, 2007, p. 609

Maximizing Average Satisfaction

Maximum Average Satisfaction is achieved by calculating the predicted satisfaction of the single users and recommend the items with the highest average. In some cases, it is necessary to transform ratings to represent satisfaction accurately. Jameson and Smyth, 2007, p. 609

Aggregation strategies following this approach in the broadest sense are *Additive Utilitarian*, *Average*, *Borda Count*, *Most Pleasure* and *Multiplicative*. Felfernig, Stettinger, et al., 2018, p. 32

Minimizing Misery

The disadvantage of maximizing average satisfaction is that some users can be left very dissatisfied. And, if the members of the group are consuming the recommended item together, frustrated group members can decrease the satisfaction of the whole group. Another possibility is that such users refuse to consume the recommended item, bringing the whole group in trouble. Minimizing misery is a strategy to recommend items with the *least misery*-function. A possible way to reach this strategy would be to recommend items only over a defined threshold. Jameson and Smyth, 2007, p. 609

A typical scenario for this strategy are shopping centers, avoiding customers getting dissatisfied caused by playing music the customers don't like. Besides the fact that such recommendations often gain low average satisfaction, furthermore, they are easy to manipulate by rating bad all disliked items. Felfernig, Stettinger, et al., 2018, p. 33

Concrete strategies following this approach are *Average without Misery* and *Least Misery*. Felfernig, Stettinger, et al., 2018, p. 32

Fairness

The goal in reaching fairness is that no one is being preferred by the expense of others. Additionally, fairness is combined with another goal. For example, it makes no sense to achieve fairness by recommending items which all group members equally dislike. Jameson and Smyth, 2007, p. 610

Fairness can be measured for single items or for a group of recommendations. Felfernig, Stettinger, et al., 2018, pp. 68–69

An example of an aggregation strategy reaching fairness is *Fairness*. Felfernig, Stettinger, et al., 2018, p. 32

Treating Group Members Differently

In some cases, it is desirable to treat separate group members differently. For example, if someone has a birthday party, this group member could be preferred on the condition not to annoy others with the recommendations. Jameson and Smyth, 2007, p. 610

Another situation could be that one member is more respected than others. And in this way, a recommendation preferring the preferences of this group leader could lead to a higher overall satisfaction. Felfernig, Stettinger, et al., 2018, p. 33

Exemplarily for this strategy is *Most Respected Person*. Felfernig, Stettinger, et al., 2018, p. 32

2.7.3. Recommendation Techniques for Groups

As already mentioned, there have to be adaptations on single user RSs to recommend items to groups. In the following sections, several approaches are explained.

Collaborative Filtering for Groups

As in Section 2.1 explained, CF is based on predicting the preference of a user by finding the *nearest neighbors*. Both *aggregate models* and *aggregate predictions* are possible strategies of applying CF for groups. Felfernig, Stettinger, et al., 2018, p. 34

When using the *aggregate models'* strategies, the rating of the group members are aggregated to a group profile (gp). The *gp* contains item-specific ratings. Unknown ratings are calculated with collaborative filtering finding similar groups. The similarity between groups can be calculated with the *Pearson correlation*. Formula 2.20 shows an adaption for groups. $\overline{r_{gx}}$ represents the average rating of group gx, TD_c is the set of items rated by both groups and r_{gx,t_i} is the known rating for item t_i from group gx. Felfernig, Stettinger, et al., 2018, pp. 35–37

$$similarity(gp, gx) = \frac{\sum_{t_i \in TD_c}(r_{gp,t_i} - \overline{r_{gp}}) \times (r_{gx,t_i} - \overline{r_{gx}})}{\sqrt{\sum_{t_i \in TD_c}(r_{gp,t_i} - \overline{r_{gp}})^2} \times \sqrt{\sum_{t_i \in TD_c}(r_{gx,t_i} - \overline{r_{gx}})^2}} \tag{2.20}$$

On the basis of the similarity of the *nearest neighbors NN* the rating can be predicted. This is shown in Formula 2.21 by Felfernig, Stettinger, et al. (2018, p. 37)

$$prediction(gp, t) = \hat{r}(gp, t) = \overline{r_{gp}} + \frac{\sum_{gj \in NN} similarity(gp, gj) \times (r_{gj,t} - \overline{r_{gj}})}{\sum_{gj \in NN} similarity(gp, gj)} \tag{2.21}$$

Aggregating predictions starts with predicting items or ratings of individuals. This predictions were aggregated to recommendations for groups. The predicted items, calculated by individual recommenders, are aggregated to a collection of proposed items without a ranking, whereas rating predictions, calculated with approaches like matrix factorization, are aggregated to get a ranking of candidate items. Felfernig, Stettinger, et al., 2018, pp. 34–35

Content-Based Filtering for Groups

Section 2.2 explains that CBF recommend unknown items like items preferred by the user. Similar to Section 2.7.3, *Content-Based Filtering for groups* can also follow the strategies of *aggregated predictions* and *aggregated models*. Felfernig, Stettinger, et al., 2018, p. 37

The individual results of the CBF applied on the group members are used by *Aggregated Predictions* to aggregate them to a group recommendation. The aggregation results are represented as similarity values between items and user profiles. The result is a $user \times item$ similarity matrix. Based on this matrix aggregation functions can determine the recommended items for the group. Felfernig, Stettinger, et al., 2018, pp. 37–39

The *Aggregated Models* strategy aggregates user profiles into a gp and determine the recommendations based on the similarity between the items and the gp. Felfernig, Stettinger, et al., 2018, p. 39

Hybrid Recommendations for Groups

As mentioned in Section 2.5.3, hybrid RSs try to combine the advantages of different systems and to reduce the negative aspects. This approach can also be adopted in RSs for groups. Following, there are two approaches to combine GRSs explained. The first is the *weighted* approach, where the scores or predicted ratings of items calculated by two or more RSs are combined to a single result. The second approach is *mixed*, where the recommended items of different RSs are combined into one recommendation of items. This can be done by the fairness approach, for example by taking the best fitting item of each individual recommendation followed by the second ones and so on. Felfernig, Stettinger, et al., 2018, pp. 51–52

Matrix Factorization for Groups

As matrix factorization is nowadays very popular in CF, there are several approaches adopting it for GRSs. Matrix factorization for single users is described in Section 2.4.1.

Two possible approaches are *after factorization* and *before factorization*. *After factorization* aggregates the user specific factors performed by matrix factorization, whereas *before factorization* performs matrix factorization with aggregated user-specific item ratings. *After factorization* is simple and efficient and provides a solid baseline, but especially on large datasets with large groups *before factorization* performs significantly better on predicting results. Ortega et al., 2016, pp. 314–317

2.7.4. Evaluating Group Recommender Systems

In general evaluation techniques for GRSs are not different to the ones for single user recommendation shown in Section 2.6. But there are properties which should be more in focus for recommending items to groups, or they can be differently interpreted for groups. Felfernig, Stettinger, et al., 2018, pp. 59–71

Coverage and Serendipity

Coverage in term of user RSs can be item or user space coverage, showing how many items have been rated from users. *Group coverage* (GC) can be defined as the number of groups with at least one group recommendation compared to the number of groups overall. Formula 2.22 shows a possible calculation. *Catalog coverage* (CC) shows the number of recommended items in comparison to the whole catalog. A suggestion for calculation is shown in Formula 2.23. Felfernig, Stettinger, et al., 2018, pp. 59–60

$$GC = \frac{|groupswithprediction|}{|groups|} \tag{2.22}$$

$$CC = \frac{|recommendeditems|}{|groups|} \tag{2.23}$$

The value of *Serendipity* in RSs for single users is described in Section 2.6.2. In GRSs, it can be seen as advantage of a GRS against a primitive prediction model *PM(g)*. This can be a system simply recommending popular items in an easy way. The serendipity value of a GRS is the average value over all group specific values *SER(g)* calculated with Formula 2.24. Ge, Delgado-Battenfeld, and Jannach, 2010, pp. 259–260

$$SER(g) = \frac{|RS(g) - PM(g)|}{|RS(g)|} \tag{2.24}$$

Consensus and Fairness

Consensus in GRSs is the similarity between a user and a group preference. This can be measured by the item preferences in collaborative filtering systems or by model comparison in model-based RSs. Formula 2.25 shows the calculation of the consensus. Felfernig, Stettinger, et al., 2018, pp. 67–68

$$consensus(g,t) = 1 - \frac{\sum_{(u_i,u_j)\in g(i\neq j)} |r(u_i,t) - r(u_j,t)|}{|g| \times (|g|-1)/2 \times r_{max}} \tag{2.25}$$

Groups to recommend items, which can either be homogeneous or stakeholders, have different interests. This leads to a different view of *fairness*. Also, the number of recommended items plays a role in *fairness*. In collaborative filtering, *fairness* can be measured by the number of users who rate the recommended item with a value higher than a defined threshold. Recommending a set of items, it can be measured in the number of items a user likes in the recommendation. Felfernig, Stettinger, et al., 2018, pp. 68–69

Fairness can also be reached over time. Users which were discriminated in previous recommendations can get a greater weight for future recommendations. This is called *long-term fairness*. O'connor et al., 2001, p. 206

2.7.5. Explanations for Groups

As for single users, explanations of the results for groups can improve RSs. In this chapter, the motivation for explanations as well as the possibilities to visualize them are described.

Motivation

There are several reasons a designer of a RS wants to explain recommendations. Especially in groups the potential for a conflict could be decreased by explaining the fairness of the recommendation. Felfernig, Stettinger, et al., 2018, pp. 107–109

Additional information can help to understand recommendations easier. Developers of RSs try to gain more confident in the system, increase the satisfaction or achieve a higher benefit with additional explanations of recommendations. Felfernig, Stettinger, et al., 2018, pp. 105–106

Visualization of Explanations

The simplest form of visualization is *verbal*. RSs for single users often describe recommendations as *"user x bought this product and also product y"* or *"because you like category x we recommend product y"*. On the one hand, GRSs can explain this group based like *"group x like this item also and additionally like item y"* or, on the other hand, member based like *"we recommend item x because group members y and z like this"*. Felfernig, Stettinger, et al., 2018, pp. 106–109

Another possibility is to visualize explanations in *diagrams*. This can be a bar chart showing the predicted values of different items or multidimensional graphs showing the distance to the nearest neighbors. Felfernig, Stettinger, et al., 2018, pp. 111–112

But there are many other possibilities. An example is the tag-cloud representation by Gedikli, Jannach, and Ge (2014). It is often domain specific or in the context of the underlying recommendation paradigms. Felfernig, Stettinger, et al., 2018, pp. 122–123

Hybrid Explanations for Hybrid Recommender Systems

Especially in the context of hybrid RSs the explanation is difficult to illustrate. The result of a hybrid RS can be differently calculated depending on the information available for the recommendation. This makes it necessary to provide various visualizations. Venn diagrams for example support this and perform outstanding against other visualizations like Concentric Circles. Kouki et al., 2017

2.7.6. Personality, Emotions and Group Dynamics

There are factors in recommending items which are hard to measure but affect the accuracy of a recommendation. Personality, emotions, and group dynamics can positively or negatively influence the opinion about a recommended item or even the trust in a RS. Felfernig, Stettinger, et al., 2018, p. 157

Personality

The Thomas-Kilmann model is designed to show the interactions between members of a group. This makes it possible to define conflict resolution styles of humans. The model has two axes of aspects: the assertiveness and the cooperativeness resulting in five different personality types. The types are *competing* (uncooperative and assertive), *collaborative* (cooperative and assertive), *compromising* (moderate in both aspects), *avoiding* (not assertive and uncooperative), and finally, *accommodating* members (cooperative and not assertive). Knowing the personality types of the group members and including this information when recommending items can achieve higher overall satisfaction. Kilmann and Thomas, 1977

In GRSs the values of *assertive* and *cooperative* can be used to weight the values of the single group members. Felfernig, Stettinger, et al., 2018, pp. 161–162

Emotions

Emotions are usually a result of stimulation. Examples for emotions, which are often used in base models, are anger, disgust, fear, happiness, sadness, and surprise. Valence and arousal are measurable factors which can be used to estimate the current emotional state. This could be anger expressed with a high arousal and low valence. The emotional state can be used as contextual dimension in CARSs to improve the predictive performance. Felfernig, Stettinger, et al., 2018, p. 159

Group Dynamics

Group decisions can sometimes be not rational and/or they are possibly not assignable to the individual group members. This is caused by group dynamics. GRSs have to take this into account for accurate item recommendation. There are two aspects, *emotional contagion* and *conformity* which are mainly responsible for influences in the group decision process. Felfernig, Stettinger, et al., 2018, p. 159

Emotional contagion is the mood's influence of a single group member on the whole group. This can be both in a negative and in a positive aspect and not every group member is equally susceptible to emotional influences of other group members. Due to higher interaction between group members in consuming music than in watching TV, listening to music has more influence on the affective state. Masthoff and Gatt, 2006, pp. 299–300

Conformity is the adjustment of a group member to others. Users often support the decision of the majority of the group or of a member with a strong personality. Humans tend to conform with the opinions of others. The reason for conformity can either be the *informational influence*, by trusting other people's opinion, or the *normative influence*, that group members do not want to stand out, or because of group pressure. Masthoff and Gatt, 2006, pp. 299–303

3. Recommending Music to Groups

This chapter presents techniques to recommend music to groups. In the first Section 3.1, existing approaches on the topic of GRS for music are introduced. Followed by the parts of describing the recommendation process which begins with the request of a new recommendation and ends with a group recommendation. The Section 3.2 shows techniques to gain information about the users' music preferences. Section 3.3 describes the approach to calculate a list of tracks for each user which contains songs based on the users' preferences. The songs of these lists are the candidates for the group recommendation. In the area of RSs, candidates are items from which the elements for the recommendation are selected. The Section 3.4 shows aggregation functions to rate the recommendation candidates. Section 3.5 shows an approach to order tracks. By listening the tracks in the recommended order, the pleasure is consistently increasing. The Section 3.6 describes the problem of users feeling discriminated over time and how to solve this problems. The last Section 2.6.2 reviews the recommendation algorithm of the app *M'Usic* based on the properties of RSs.

At the end of all sections, except the first and the last one, you can find a description of how the approach is implemented in the app *M'Usic*. Initially one algorithm had been developed which was the basis for several different variants. In Chapter 5, an evaluation of these variants is shown. Based on the results of the evaluation, a final algorithm is implemented and described in this chapter.

3.1. Existing Approaches

In this section existing approaches are presented. The solutions are ordered by the year they were published.

3.1.1. MusicFx

MusicFx is a group arbitration system that decides which music is played in a fitness center based on the current visitor's preferences. The calculated user preferences are based on a database where user ratings of categories are stored. The group preference is calculated by a summation formula of the individual preferences. McCarthy and Anagnost, 1998

3.1.2. Flytrap

Flytrap is a group music environment that knows the taste of its users and can detect, with the help of radio frequency ID badges, the present users. The system tries to play music that satisfies all members who are currently in the room. The preferences of the users are gathered from the tracks the users have been listening. With the knowledge of the interrelation of genres, it decides which tracks the users want to listen to. Crossen, Budzik, and Hammond, 2002

3.1.3. Jukola

Jukola is an interactive MP3 jukebox offering possible songs to play. It contains a voting system, allowing users to select their preferred songs out of the offered ones. The song with the highest votes is played next. O'Hara et al., 2004

3.1.4. Adaptive Radio

Adaptive Radio is a system in a shared environment where music is played. It follows the principal that it is easier to find songs which users do not like than songs which the users like. This allows the system to generate a list of songs. For each song there is at least one group member who does not like it. The technique is called *negative preferences*. Additionally, CF allows to find similar songs to the list and extend it. All other songs are possible candidates to be recommended. Chao, Balthrop, and Forrest, 2005

3.1.5. In-Vehicle Multimedia Recommender

In-Vehicle Multimedia Recommender is a RS for passengers of a vehicle. This can be a car, a bus, or even a plane. The system merges individual user profiles with the help of *feature selection* and *weight assignment* to a *common user profile*. This is the basis for the recommended songs. *feature selection* is a technology to reduce the total distance by SVM, *weight assignment* is the combination of *weight normalization* where the weights of the user profiles are normalized, and *weight calculation* where the average weight of all normalized user weights for each feature are calculated. Yu, Zhou, and Zhang, 2005

3.1.6. PartyVote

PartyVote is a music jukebox with a democratic aspect. Every user can give votes for songs and at least one voted song is played per user. Similar songs are more likely to be played. This leads to a potentially playable song region. Songs outside of this region will not be played. The distance between songs are calculated with the Euclidean distance. Sprague, Wu, and Tory, 2008

3.1.7. jMusicGroupRecommender

jMusicGroupRecommender is a GRS following the approaches merging of individual recommendations, individual rating aggregation and the construction of group preference models. Additionally, a data model based on the preferences and the correlation among songs helps to estimate ratings of unrated tracks. Christensen and Schiaffino, 2011

3.1.8. GroupStreamer

GroupStreamer[1] is an app provided on Google Play Store[2]. The user preferences are based on their music library with the use of Last.fm[3] to get key words corresponding to these songs. It models the user's preferences using a Gaussian mixture model. To reduce the number of distinct tags, a latent semantic analysis is used to get a smaller concept space. With the use of aggregation strategies, the recommended tracks are calculated. Maystre, 2012

3.2. Information Retrieval

This section examines the possibilities of a RS to gain information about the users' preferences, the context information of songs and the relationship between songs, between users, and between songs and users.

3.2.1. Information Retrieval Approaches

In the following, several approaches to information retrieval are discussed.

[1]https://play.google.com/store/apps/details?id=ch.epfl.unison
[2]https://play.google.com/store
[3]https://www.last.fm/

Explicit Information Retrieval

If there is no data about the users' preferences available, GRSs for music have to ask the users about their preferences. This can be done in different ways:

- *Initially ask for favorite genres, interpreters or songs*. This is a possibility to query the basic music taste of the user with little effort. The disadvantage of this method is that the system cannot get deeper information about the user's preferences. The system has to implement further mechanisms to gain more knowledge of the user's taste. For example, to improve the results of following recommendations by asking the user about his/her opinion about the recommended songs.
- *Ask for ratings of songs*. A higher number of rated songs leads to a better knowledge of the user's preferences. This is a very time-consuming process for users, and so they are often not willing to do that. There have to be incentives to get a satisfactory amount of ratings. This can be better personal recommendations, under the circumstance that the GRS is integrated in a music player which also includes a RS making personal recommendations for each user. The less ratings available, the worse the recommendations are. The worst case is a user without ratings. This case is discussed in Section 3.2.1.

Explicit user action to get information about music preferences has no high acceptance, especially because of the fact that a high value of knowledge needs more user action. Another disadvantage is that preferences can change over time and it is hard to determine if all ratings are still correct. Checks lead to an even higher user effort. Because of the music's diversity, it is also possible that users overlook genres they would like to rate or to select as favorite.

Implicit Information Retrieval

If the GRS for music is integrated in a music player or has an interface to a music service, it is possible to improve recommendations by provided information. The following items are examples of information a system can gather.

- *played music*. A system can grab information about all songs a user listened to, and additionally, the date and the time of the day it was played. At least the top tracks and interpreters are provided by interfaces.
- *playlists*. Most music services provide the possibility to create playlists. It can be assumed that the users like the songs in their own playlists.
- *favorite interpreters and genres*. Some music services allow users to define or follow their favorite interpreters or genres. The advantage for users is to get recommended songs which are similar to this interpreters or genres. Some services, additionally, provide information about news like new songs or albums, or even tour dates of users' favorite interpreters as an incentive to like or follow artists.
- *region based top songs and interpreters*. Some interfaces provide information, filtered by regions or overall, about how popular songs are and which are the most listened to songs respectively artists.
- *recommended playlists*. Spotify[4], for example, suggests public playlists for a specific time and location to users.
- *track information*. Music services have big music collections including contextual information as a basis. A GRS for music can use this to get track information of an enormous amount of songs.
- *similar songs and artists*. Some music services like Last.fm[5] provide the possibility to get similar songs or artists to a given one.
- *songs similar to track features*. Spotify provides the possibility to get tracks with features similar to the requested ones. This allows to receive songs with a small distance to a track model.

Information Retrieval over Time

Information Retrieval can be improved by continuous observation of user preferences and getting new songs. The more information a system gets, the better the recommendations can be.

User preferences change over time which leads to wrong ratings or assumptions. A GRS for music has to recognize changes and react. Unfortunately,

[4]https://spotify.com/
[5]https://www.last.fm/

the system can not conclude that a user does not like a rarely played song any more which was listened frequently in the past. A system has to decide how to deal with these changes.

Music recommendation is a topic where both, new songs and old songs, have to be possible suggestions. As the charts changing within a few weeks, music recommenders are outdated and mainly not accurate if the underlying music database is a few months old.

Cold Start Problem

The *Cold Start Problem* describes the obstacle that a RS does not have any collaborative information about a new song, or any collaborative and preferences information about a new user. There are several ways to deal with this problem. As CF recommender systems are depending on collaborative information, a way would be to combine approaches to a hybrid RS. Another way would be to explicitly ask new users about their preferences.

In groups, there are additional ways to deal with new users:

- *mainstream user*. If a new user in a RS for individuals is treated like a mainstream user, this could lead to completely wrong suggestions and dissatisfaction of the user. In a group, a single member's preferences are balanced by the preferences of the other group members. Assuming that a new user is listening to the region based most popular songs can improve the results of a music recommendation to such groups.
- *average group member*. A new user can be seen as an average group member. In this case, the preference model of the new user is the calculated preference model of the group. This could be a successful opportunity in cases where the system provides personal suggestions and the user can modify them. Afterwards, the modified list is used as the preference list of this user.
- *ignore the user*. A simple approach, which does not sophisticate the results, is to ignore the user. Often it is better to ignore the preferences of a user than assuming wrong preferences. Under the circumstance that a user is informed that due to a lack of information his/her

> musical taste is not included in the group preferences and how he/she can supplement these, the user would accept this.

All approaches have the same disadvantage. Only if the preferences of the large majority are known, new individual users can be handled with them.

3.2.2. Information Retrieval of M'Usic

The app *M'Usic* retrieves information with the help of the Spotify for Developers[6] application programming interface (API). It is collecting the information about the top tracks of the users short, medium and long term. Furthermore, it requests special playlists based on a specified time and region from the API. For the cold start, it also uses the *featured playlist* interface, letting Spotify[7] recommend the preferences of the new user. To get detailed information of each song, *M'Usic* requests the track information of all songs. In further steps it also requests similar songs to a given preference model.

Advantages

The advantage of the chosen information retrieval approach is that no explicit information retrieval is necessary. As Spotify is widely used, a big user basis and their preferences are available. This allows millions of users to use the app out of the box without any questions to answer about their preferences, and then can initially access to the information from the first day on using Spotify. Either, the amount of tracks and their information is nearly impossible to get through other ways. The Spotify Company has invested a lot of effort in recommending music to single users and additionally, to give a wide range of popular playlists, even to new users. This allows the app to use these suggestions as the basis for the cold start problem. Such playlists are recommended in the context of date and time. This allows to get songs typically played at this time. Since the moment a song is listened to, Spotify

[6]https://developer.spotify.com/
[7]https://spotify.com/

knows which song was played at what time. Each track is analyzed by an algorithm, for filtering the track information out of a song. This allows to create a preference model and calculate the distance between the model and tracks.

Disadvantages

The disadvantage is that Spotify for Developers does not provide the possibility to get a user's played tracks and the time the tracks were listened. As compensation the app uses the top tracks of the users to calculate the preference model. Anyway, the top tracks are available for three different time ranges. Another disadvantage of the API is that there is no possibility to access all songs which are available on Spotify. Track information can only be received when the id is known. The ids are received by other interfaces of the API, like the top tracks or recommended playlists. This prevents the possibility to find tracks near a group preference model with the help of SVD. Fortunately, the API provides the possibility to get recommended tracks similar to a track model.

3.3. Personal Suggestion List

A personal suggestion list is a set of recommended items for a single group member. Such lists can be calculated for each user in a group to take the result as the basis of the group recommendation. Users can manipulate their lists to get a set of items matching the taste of them. Different approaches of creating and handling personal suggestion lists are discussed in Section 3.3.1. Afterwards, the developed technique of the app *M'Usic* is presented.

3.3.1. Personal Suggestion List Approaches

A common way to calculate a music recommendation to groups is to start with the generation of a personal suggestion list. This is a collection of song candidates for the recommendation, fitting to the preferences of a group member. The first step is to figure the music taste of the group members. With these preferences the RS tries to form a model. Based on this, the RS find a set of possible tracks. Finally, the RS selects the best fitting songs of this set to create a personal suggestion list.

User Preferences

User preferences can be extracted out of different kind of data. This can be a set of ratings of tracks made by the user on the one hand, or on the other hand, the list of played songs. In the first case, techniques explained in Section 2.1 can be used to calculate user preferences. In the second case, the data needs to be preprocessed. The most played tracks have to be figured out, and additional attributes can be extracted, for example, the time of the day the tracks were played, or even the day of the week. Furthermore, the set of played songs can be prefiltered in time ranges, for example, by counting only tracks listened in the last year. Another useful information is the range of genres the user has listened to and their properties in common. And, of course, all contextual information which can be extracted out of the underlying data.

Model Creation

In the step of *Model Creation*, the RS tries to form a model out of the user preferences gained before. As described in Section 2.4.2, this can be a single calculation or a continuous process. The process of model creation can be affected by the contextual situation considered in CARSs described in sections 2.3 and 2.5.2. For example, a preference model could be different on Monday morning and on Saturday evening.

In GRSs this is the first step in recommending items, in which other members of a group can influence the result of a user. The tastes of all members can be aggregated to affect the preference model of the users. An example would be that the furthermost values of the model compared to the aggregated model are weakened.

Collecting Possible Tracks

Possible tracks are candidates for the personal suggestion list. One type of candidates are the positively rated songs of a user respectively the played songs of a user. These are basically good candidates. Successful RSs have to set a focus on the properties *Novelty* and *Serendipity*. So only the top tracks would not satisfy a user, and furthermore, there are highly likely songs fitting better to the taste of all group members.

Additional songs of the candidate list are the ones fitting to the preference model calculated before. With the help of SVD described in Section 2.4.1, relevant tracks can be effectively selected out of big song databases.

To receive unknown songs for users, an approach is to find songs which correlates to the top genres of a group, based on the preference model of the users.

Song Selection

In the step of *Song Selection* the RS has to select the most relevant songs out of the candidate set calculated before. The amount of the songs should be large enough to represent the taste of the user, and additionally, as large as the requested amount of songs of the recommendation. To calculate the best fitting songs, *Similarity Measures* described in Section 2.4.1 are used.

There are different ways to order the candidates:

- *Order by the user preference model*. The whole list of possible tracks is ordered by the distance between the user preference model and the individual songs.
- *Order by the group preference model*. Another way is to order the tracks by the distance between the aggregated group preference model and the individual song. This would bring songs to the top which are fitting best to the taste of the group.
- *Fix amount of songs of the top tracks and filtered tracks*. The order of both lists is made with one of the possibilities above, and the resulting selection is a combination of the lists in a fixed ratio. This has the advantage that top tracks of a user will definitely stay in the suggestion

list, and additionally, there are songs to improve the RS in *Novelty* and *Serendipity*.

3.3.2. Personal Suggestion List of M'Usic

The basis for the user preferences are the top tracks of the user. There were variations of the algorithm using the most listened songs of the user in the short history, medium term or overall. The evaluation results in 5.1 and 5.12 show that long term top tracks and especially short-term top tracks result in a good recommendation. Furthermore, the diagrams in 5.7 and 5.18 show that a recommendation based on the top tracks of overall result in higher satisfaction. As a result of the evaluation, the final algorithm is based on the combination of the top tracks of the last four weeks and the overall top tracks.

In the initial algorithm there was the intention to bring the context of time into the recommendation. As there were no data available showing how songs differ due to the time of the day or the day of the week, another way had to be found. As the preference model is defined as the *Audio Feature Object* in Spotify for Developers[8], the idea was to define some of the values as time depending. These were the *loudness*, the *tempo*, the *valence*, the *danceability* and the *energy*. To get target values for these attributes, the calculation was based on the suggested playlists for a specific time provided by the API of Spotify for Developers. Nevertheless, the results in 5.3, 5.4, 5.14, and 5.15 show that the approach was not perceptible better than the time independent algorithm. The reason can be a wrong assumption of the time depending attributes or caused by unsuitable calculated values for these attributes.

The initial algorithm, additionally, adds a set of songs to the candidate list based on the preference model in combination with the five most common genres of all group members, and a set of songs by artists which are similar to the top artists of the user and to the preference model. As it is not possible to get songs only based on the preference model from the API, the app has to get songs based on the model in combination with five genres and five

[8]https://developer.spotify.com/

artists. As variant, the algorithm was modified that either only genres or only top artists are considered. But neither the recommended songs based on the genres nor the recommended songs based on the artist were more successful than the initial algorithm which is shown in the Figures 5.1 and 5.12.

The initial algorithm orders the tracks by the *Euclidean distance* between the individual songs and the preference model. Additionally, there is a fixed ratio between top tracks and tracks found by the model of 3:7.

A variation is to prefer songs with a high value of valence. The assumption is that songs listened to in groups should transmit a positive feeling, because in groups everyone wants to be happy. The evaluation shows in 5.1 and in 5.12 a noticeable increase of satisfaction. In 5.7 and 5.18 it is obvious that inappropriate songs decrease significantly. This variant is implemented in the final algorithm.

Another approach is to prefer songs which can be associated with the top five genres of the group. Unfortunately, as 5.1 and 5.12 show, the correctness of the recommendation can not be increased. Only light decreasing of dissatisfaction can be observed in 5.7 and 5.18. The results show no clear improvements, so the variation is not included in the final algorithm.

As a result of the big music database of Spotify, same songs are multiple times in the database. The reason is that songs can be released in a single and in an album. Sometimes there are variations of the same song available, like an album version and a radio edit. The algorithm filters double entries out of the candidate's suggestion list, leaving only the best fitting version in the collection of possible tracks.

3.4. Aggregation Strategies

In the field of GRS, *Aggregation Strategies* are methods to combine the individual user preferences to a group preference model. In the following, different approaches are discussed. Afterwards, the strategy, which was developed for the app *M'Usic*, is presented.

3.4.1. Aggregation Strategy Approaches

In this section, strategies are discussed to aggregate the users' individual song lists into a set of candidates for the resulting group recommendation including a rank or a value to order the tracks.

The system has to form a group preference model out of the personal suggestion lists. All songs have to be compared with this model. The closest to the model are the recommended tracks.

Manipulation of the Suggested List by the User

The GRS can give the users the possibility to manipulate the individual suggested list of songs. This approach has multiple advantages. A list of recommended tacks is presented to the user. The system would add this list to the candidate list of the final recommendation on behalf of the user. This gives an insight into the type of music the system recommends to the user as a member of the group. The user has the possibility to remove unliked tracks. Additionally, songs can be added, for example, when the user especially likes them or the user would like to suggest them to the group to listen to. The disadvantage of giving the user this possibility is that a group member can try to manipulate the personal suggestion list maliciously to dissatisfy other users. A RS has to prevent that single users can have too much influence in the resulting recommendation by manipulating the individual set of suggestions.

Model Creation

After making available all suggestion lists, the RS can start to form a preference model to compare all songs with. There are several strategies to calculate such a model. In the following some are described.

- *Aggregation model of all initial user models*. The system can create the group preference model based on the preference models of all group members calculated previously.

- *Aggregation model of updated user models*. In this case, the system initially creates new individual user preference models based on the manipulated song list. The group preference model is calculated by the aggregation of these models.
- *Group preference model based on all candidates*. A group preference model can be calculated based on all sets of track candidates provided by the users.
- *Group preference model based on initial candidates*. It is also possible to calculate the preference model on the initial suggestion list provided to the users.

The resulting model should represent the music taste of the whole group. Based on this, the recommended tracks are calculated.

Distance Calculation

The distance calculation is the step to choose the resulting recommended songs out of the candidates provided by the users. This can be done by comparing the list of possible tracks with the preference model of the group. Another way is to compare the song candidates with the user models of each group member and sum up the results to a single comparable result value.

Due to the possibility that songs can be multiple times in the set by containing in the suggestion list of different group members, the RS has to define a strategy to deal with these tracks.

3.4.2. Aggregation Strategies of M'Usic

The app *M'Usic* lets users manipulate their personal suggestion list. There are two possible ways of confirming the list. Either by confirming explicit the manipulation, or alternatively by doing nothing. For unanswered personal suggestion lists, a service checks the time of the event. If this is imminent, these lists are automatically confirmed. When all lists are either confirmed or declined, the final step of the recommendation starts.

The initial algorithm calculates the group preference model based on all song candidates provided by the group members in combination with the time-based values calculated in the previous step. Each track is compared to the group preference model resulting in different distances. If there are songs that are included more than once in the candidate set, each time, except the first one, the distance value is decreased by ten percent. The list is ordered by the resulting distance value and the ones with the smallest distance will be selected.

The variation of the algorithm ignoring the time context already discussed in Section 3.3.2, calculates the group preference model based on the selected songs by the group members only. As mentioned above, the factor of time could not reach the expected results in comparison to the variation which is time independent.

At this point, the variation where songs from the five most common genres are preferred has an impact. Each song, which is classifiable to one of the genres, is preferred by reducing the distance value by ten percent. But this approach is only slightly more successful than the basic algorithm. Whereas the variation has similar results than the initial algorithm for the personal recommendation as seen in 5.1, the results of the final recommendation show widespread user ratings in 5.12 and is therefore not qualified to be implemented in the final algorithm.

Also, the factor of *valence* was considered during this step. Valence describes the positiveness of a song. Tracks with a higher value of valence are more happy, cheerful, or euphoric. The variation reduces the distance between the group preference model and the individual songs by the factor of valence. A high valence shrinks the distance more than a low value. As mentioned in Section 3.3.2, the variation has a high positive impact on the results of all evaluated attributes.

A variation of the algorithm is the user-based aggregation. In this approach, the sum of the distances between a song and the individual user models of all group members is calculated. The variation was more like a test to confirm the approach to calculate a group preference model. Both algorithms reach the same results on evaluating the personal suggestion list shown in 5.1, which is expected because there is no difference in the algorithm up to this step. Unfortunately, the initial approach reaches better overall results in the final recommendation shown in 5.12. An interesting aspect of the user-based aggregation version is that it reaches better results in the prevention of dissatisfaction. This is shown in 5.18. This is probably because in the comparison between the users and songs, much disliked songs have such a big distance that the sum is also that high that it is not part in the final recommendation. The approach is effective on reaching *least misery*, which is discussed in Section 2.7.2.

3.5. Sequence Order of Recommended Items

As in most of the cases only one recommended item is consumed, the best results are presented first. When recommending music this is different. In the following, different approaches are discussed how to order the presented items. Afterwards the way of calculating the sequence order of the recommended items in the app *M'Usic* is presented.

3.5.1. Sequence Order Approaches

When consuming only one item, the best recommendations are presented first. Recommending music is a special case. People prefer to listen to songs as a playlist rather than separately. Either the tracks are quite similar, so the order of them is not that important, or the tracks are from different genres. In this case tracks should be ordered by their similarity. For example, at a party, members get dissatisfied when fast and calm tracks are played alternatingly. Another point that has to be taken into account is the fact that the best fitting songs should not be played at the beginning. A user tends to remember things in the immediate past more than things before. Another approach is to select a following song as the best fitting track to the most dissatisfied group member. So, the system has to take into account different points, where some can be converse. For example, if a member listens to completely other music than the others, over time the user gets dissatisfied. When the next song is appropriate to the taste of this user, there is a hard break in the flow, which is divergent to the approach that similar songs should be played one after another. The resulting order should be a continuous set without disregarding the other points.

3.5.2. Sequence Order of M'Usic

The underlying approach of the algorithm is to measure the similarity of songs by the amount of the same assignable genres. This is done by creating connected weighted graphs, where the vertexes are the tracks and the edges are the similar genres between two songs. The weight of the edges is defined by the number of similar genres. The result is always a set of contiguous weighted graphs. The number of graphs can be between one if all songs are connected, and the number of songs if no genres match. The algorithm is using the *Kruskal (1956)'s algorithm* to calculate minimum-spanning-trees. Within these spanning-trees the algorithm computes the longest connecting chain of songs by finding the two deepest leaves and add this list to a result list. The algorithm repeats the finding of the longest connection chain in the remaining now split trees and for all minimum-spanning-trees. The result is a set of lists. The algorithm tries to combine lists by finding endpoints of two lists having at least one similar genre, starting with the highest number of similar genres. At this point, another approach is used to form a final playlist: to increase the positive atmosphere in the group over time, the measure of musical positiveness should increase track by track. The algorithm orders the chain by the increasing positiveness and calculates the mean value of this. The resulting playlist is the combination of all chains ordered by the average value of positiveness ascending. The goal of this method is to get long coherent sub-lists in the playlist and overall increasing happiness over time. As shown in Figure 5.21 75 percent of the survey participants, who mostly listen to the playlists in the given order, prefer the order of the recommended playlist to the shuffle mode.

3.6. Long-term Fairness

Long-term Fairness is a possibility in GRSs to avoid dissatisfaction. This section includes a brief discussion about approaches in this field followed by the solution, the app *M'Usic* has implemented.

3.6.1. Long-term Fairness Approaches

Long-term Fairness is a desired goal to equally satisfy every member of a group over time. The approach has to be considered from the second request of a playlist of the same group. *Fairness* is described in Section 2.7.4. Not every member of a group can be equally satisfied with a single recommendation. A system, which considers a long-term fairness approach, has to define a calculation of fairness. This calculation should include a value of fairness before the recommendation and a satisfaction value of the current recommendation of every group member. This allows to adjust a value of fairness for each member after every recommendation. This value is initially the same for everyone in the group. On calculating a new recommendation, the group members can be treated differently based on the value of fairness. Over time, this results in a balanced satisfaction of all group members. As the group members can change over time, the system has to define a way to deal with this.

3.6.2. Long-term Fairness in M'Usic

In *M'Usic* on creating a new group, every member gets the same initial value. After a recommendation, every user preference model is compared to every track, and an aggregated value of the differences to the tracks is calculated. The adjusting value is the percent-based variation of the individual aggregation values compared to the average value of all aggregation values. The fairness value is decreased by this percentage. When the next recommendation comes to the part of the aggregation of the personal suggestion lists, the distance between the track and the group preference model is multiplied by the fairness value of the users' which have included this track in their personal suggestion list. A smaller fairness value comes to a smaller distance value and this increases the possibility that this track is part of the final recommendation. As the evaluation of fairness would need an investigation over a period, it has not been taken into account when evaluating the algorithm.

3.7. Recommender System Properties in M'Usic

In this section, the algorithm of *M'Usic* is considered by some of the properties of RSs which were described in Section 2.6.2.

3.7.1. Prediction Accuracy

The evaluation includes the investigation of how the accuracy differs when recommending a different number of songs. The algorithm was tested with 30, 50 and 80 tracks. The outcome of the survey is quite similar, but surprisingly, the best results are reaching playlists with 30 and 80 songs as shown in 5.12 and 5.19. A reason for this phenomenon could be that a shorter playlist includes the favorite songs of the group members and therefore the list is rated higher. If the list contains more elements, the number of songs not liked by users can increase, which decrease the ratings. Large playlists include a higher number of tracks, which are not loved by single members of the group but appreciated by the group as a whole. This could lead to a higher rating. The explanation is only an assumption which has to be further evaluated.

3.7.2. Coverage

As shown in Section 2.6.2, *Coverage* can be seen as *item space coverage* and *user space coverage*. The algorithm of *M'Usic* is able to recommend all items. Therefore, the *Item space coverage* is the number of tracks available on Spotify[9]. As the algorithm has a fallback to recommend songs a user has even not listened to music in *Spotify* before, the *user space coverage* is one hundred

[9]https://spotify.com/

percent with the limitation that the recommendations are getting better for users who are listening to music in *Spotify* frequently.

3.7.3. Trust

Trust is a factor which can be reached over time. Therefore, it is hard to state if users trust in the algorithm of *M'Usic* or not. An indication for this property can be the question *Would you use this algorithm again?* in the evaluation. The results are shown in Figure 5.19. The initial algorithm is not rated high, but variations which are implemented in the final algorithm reach significantly better results. This is an indicator for a high value of trust.

3.7.4. Novelty

As described in Section 2.6.2, novelty is a factor that describes how many items of a recommendation are new to a user. The property is evaluated based on the individual recommendation caused by a typically high level of novelty in a group recommendation when users with different tastes of music are in the same group. Additionally, at this stage, the RS can suggest songs the user would not have thought about, but like. The results in 5.10 show that the proportion of known songs in all variations of the algorithm are about 50 percent or slightly higher. Correlated to the results in 5.11, the candidates do not want to have more unknown songs.

3.7.5. Serendipity

Serendipity in RSs for music is when a user surprisingly spots a new song he/she likes. This is already discussed in Section 2.6.2. To evaluate the value of *Serendipity*, the question *Did you get an unknown song from an unknown interpret which you like in your personal recommendation?* was asked. The result displayed in Figure 5.8 shows that all variations of the final algorithm up to the *long term* have surprisingly liking songs in more than the half of the individual recommendations. In correlation with the results of the question *Did you get an unknown song which you really like in your personal recommendation?*, which are shown in Figure 5.9, it can happen that most of the unknown songs are from unknown interprets.

3.7.6. Diversity

The factor of *Diversity* is described in Section 2.6.2 and is part of the evaluation as well. The results in Sections 5.5 and 5.16 show that the value of diversity increases from the individual recommendation to the final group recommendation. Furthermore, the participants of the evaluation were asked if they want more diversity in their recommendations. The results, displayed in Figures 5.6 and 5.17, show clearly that there is at least sufficient diversity for the participants.

3.7.7. Utility

The *Utility*, defined in Section 2.6.2, of the app *M'Usic*'s algorithm is to increase user satisfaction while listening to the recommended music. Furthermore, it should decrease the effort in creating playlists. The utility for the app *M'Usic's*, is the ability to make further research based on the collected data. Further, Spotify[10] can have a benefit caused by the obligatory *Spotify* account to use the app.

3.7.8. Robustness

The app *M'Usic* has integrated an manipulation avoidance, caused by the architecture of the system. The basic information about the preferences of the users are stored in Spotify and the security of the data is therefore not the purpose of the app. The manipulation by faking the preferences of the own user on *Spotify* is not that effective in a group recommendation and has no impact on recommendations the user is not part of. Furthermore,

[10]https://spotify.com/

this results in bad recommendations on *Spotify* for the user itself. As all communications between the internal parts and external services of the app are encrypted, the maximum effort in security was made compared to the risk.

3.7.9. Adaptivity

The algorithm reacts on changes of the user preferences over time. Each listening to a song is observed by Spotify and is therefore included in the data the algorithm uses to create recommendations. As the algorithm is using the top tracks of a user, listening to a few songs will not change the preference model. Nevertheless, the top tracks of the last four weeks are part of the adduced data, so the changes in the preference would quickly change the preference model of the user. Other factors of *Adaptivity*, which were discussed in Section 2.6.2, have not to be considered, because recommendations are based on the amount of listened songs and not on ratings.

3.7.10. Scalability

Scalability as described in Section 2.6.2 is in the app *M'Usic* not as difficult to reach than in other RSs, due to the fact that the high computing power needing calculations is made by Spotify for Developers[11]. The response time of the API is quite good. Additionally, the use case of the app is to request for recommendations for the near future, when the group wants to listen to music. Therefore, immediate recommendations are not as important as in recommendations for individual users.

[11]https://developer.spotify.com/

3.8. Algorithms of M'Usic

In this section, the basic algorithm and its variations are described formally in Table 3.1. q_{up} is a time-independent vector of the user profile up, whereas p_{up} is a time-dependent vector of up. The user profile vector r_{up} and the group profile vector r_{gp} can both be time-dependent and time-independent. Section 3.3.2 lists which attributes are time-dependent and which are time-independent. The function *top* delivers a set of most listened songs. The first parameter indicates the number of songs, the second the time span in which the songs were listened to. s returns the most listened songs of the last two weeks, m the songs of the last months, and l the most listened songs without a time limit. The function *time* returns a required number of songs that perfectly suit for the given date and time. The distance function d calculates the distance value between the user profile or the group profile and a song. s_v and c_v are the values of the valence of the songs s and c. A song s is part of a set of candidates for the user profile. The candidate list for the users consists of the top tracks of the user and songs which are similar to the user profile. Preference is given to songs that match the most common artists and genres of the entire group. The songs sa and sg are part of special candidate lists, which either prefer only common artists or only common genres. The function *genre* returns the value 0.9, if the given song is assignable to the five most common genres of the group, otherwise 1. To get a user's suggestion list, the function *topUP* returns the songs of the candidate list with the lowest distance value according to the user profile. The songs c from the candidate list for the group recommendation are the songs of the group member's suggestion list. The recommended songs for the group are those from the candidate list which have the smallest distance value to the group profile.

Table 3.1.: Formulas of the variations

Variation	**Formula**
Basic Algorithm	$q_{up} = \frac{1}{30} \sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9} (up_k - s_k)^2}$ $q_{gp} = \frac{1}{30} \sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$
Basic50	$q_{up} = \frac{1}{50} \sum_{i \in top(50,m)} q_i$ $p_{up} = \frac{1}{50} \sum_{i \in time(50)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9} (up_k - s_k)^2}$ $q_{gp} = \frac{1}{50} \sum_{i \in topUP(50)} q_i$ $p_{gp} = \frac{1}{50} \sum_{i \in time(50)} p_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$
Basic80	$q_{up} = \frac{1}{80} \sum_{i \in top(80,m)} q_i$ $p_{up} = \frac{1}{80} \sum_{i \in time(80)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9} (up_k - s_k)^2}$ $q_{gp} = \frac{1}{50} \sum_{i \in topUP(80)} q_i$ $p_{gp} = \frac{1}{50} \sum_{i \in time(80)} p_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$
ShortTerm	$q_{up} = \frac{1}{30} \sum_{i \in top(30,s)} q_i$ $p_{up} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9} (up_k - s_k)^2}$ $q_{gp} = \frac{1}{30} \sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$

Continuation of Table 3.1	
Variation	**Formula**
LongTerm	$q_{up} = \frac{1}{30}\sum_{i \in top(30,l)} q_i$ $p_{up} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9}(up_k - s_k)^2}$ $q_{gp} = \frac{1}{30}\sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9}(up_k - c_k)^2}$
UserBasedAggregation	$q_{up} = \frac{1}{30}\sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9}(up_k - s_k)^2}$ $d(gp,c) = \sum_{u \in groupmembers} d(gp_u, c)$
TimeIndependent	$r_{up} = \frac{1}{30}\sum_{i \in top(30,m)} r_i$ $d(up,s) = \sqrt{\sum_{k=1}^{9}(up_k - s_k)^2}$ $r_{gp} = \frac{1}{30}\sum_{i \in topUP(30)} r_i$ $d(gp,c) = \sqrt{\sum_{k=1}^{9}(up_k - c_k)^2}$
Valence	$q_{up} = \frac{1}{30}\sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(up,s) = \frac{1}{s_v} * \sqrt{\sum_{k=1}^{9}(up_k - s_k)^2}$ $q_{gp} = \frac{1}{30}\sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(gp,c) = \frac{1}{c_v} * \sqrt{\sum_{k=1}^{9}(up_k - c_k)^2}$
Genre	$q_{up} = \frac{1}{30}\sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(up,s) = genre(s) * \sqrt{\sum_{k=1}^{9}(up_k - s_k)^2}$ $q_{gp} = \frac{1}{30}\sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30}\sum_{i \in time(30)} p_i$ $d(gp,c) = genre(s) * \sqrt{\sum_{k=1}^{9}(up_k - c_k)^2}$

Continuation of Table 3.1	
Variation	**Formula**
RecommendGenre	$q_{up} = \frac{1}{30} \sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(up, sg) = \sqrt{\sum_{k=1}^{9} (up_k - sg_k)^2}$ $q_{gp} = \frac{1}{30} \sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(gp, c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$
RecommendArtist	$q_{up} = \frac{1}{30} \sum_{i \in top(30,m)} q_i$ $p_{up} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(up, sa) = \sqrt{\sum_{k=1}^{9} (up_k - sa_k)^2}$ $q_{gp} = \frac{1}{30} \sum_{i \in topUP(30)} q_i$ $p_{gp} = \frac{1}{30} \sum_{i \in time(30)} p_i$ $d(gp, c) = \sqrt{\sum_{k=1}^{9} (up_k - c_k)^2}$

4. The App “M’Usic”

This chapter presents *M’Usic*, what stands for an app to support groups finding songs that match the taste in music of all members. All accounts are connected to *Spotify*[1], so the app knows the taste in music of all users. This allows *M’usic* to find appropriate playlists for groups without explicit information retrieval. This can be useful when groups make parties, road-trips, or when group members are studying together. *M’Usic* is based on the implementation of the algorithm developed in Chapter 3. The chapter is divided into the Section 4.1, where the use-cases of the app are shown, the Section 4.2, which shows the technical background of the app and the used technologies, the Section 4.3, describing the information which is used to calculate the recommendations followed by the last Section 4.4, where the tools to increase the quality of the app are shown.

4.1. Functionality

In this section, the use-cases are shown, and we explain how the app works. *M’Usic* is generally divided into the account-management, the group-management, and the playlist-management. In the playlist part, existing playlists and playlists, which are currently in the creation process, are shown. Further, new recommendations can be requested. An overview of creating a new group playlist is explained in the following. Initially, the user has to register on the app and connect his profile to *Spotify*. In the case of an existing account, the user can use the app by logging in. The next step is to create a group. It is possible to manage the users of a group afterward by the creator of a group. On the *group detail* page, a user can request a new

[1]https://spotify.com/

recommendation for the group. After the calculation of the user-specific songs for the group recommendation, the user gets a notification and can add or remove items and confirm or refuse the item collection. After each group member has confirmed or rejected the collection, the aggregation calculation is triggered. Each user is notified and receives the tracks of the playlist presented when the calculation is finished. The user can transfer the playlist to *Spotify* to listen to the songs.

The following use cases are beside the textual description shown by screenshots of the app. Most figures are only shown in the iOS version only. There are some navigation items that typically differ between *iOS* and *Android*. The app takes care of these peculiarities which leads in some cases to different screens. In such cases, screenshots in the following sections show both, the *iOS* and the *Android* views.

4.1.1. Account Management

The account management use-cases are related to tasks concerning on the account of the users. This includes the registration process, the login and logout, and the forgotten password process flow.

Registration

When starting the app the first time, a login screen (4.1) is shown. If the user is not registered, a registration is yet necessary. The registration starts by clicking the link *Sign up now*. The registration view (4.2) opens. The first step is to insert the users email address and press *Send verification code* to receive the verification code necessary for the check, if the email address is one of the users'. The verification code has to be verified by clicking *Verify code*. After filling all input fields correctly, the user can create his account and is forwarded to a *Spotify* authentication page (4.3), to connect the user profile with *Spotify*. There, the user has to click on *sign up for Spotify*. On the forwarded page, the user has to fill in the login information and press the *login* button. The app confirms the connection to *Spotify* by showing a registration finished page (4.4).

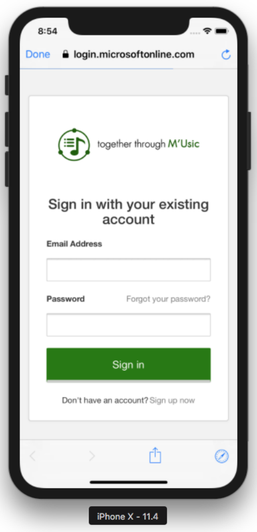


Figure 4.1.: M'Usic SignIn Screen on iOS

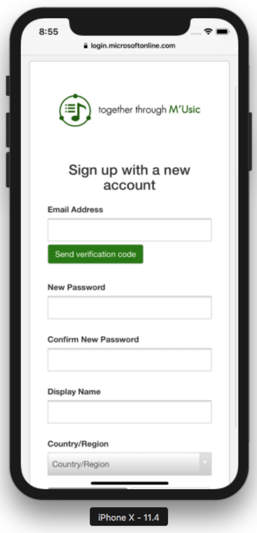


Figure 4.2.: M'Usic Registration Screen on iOS

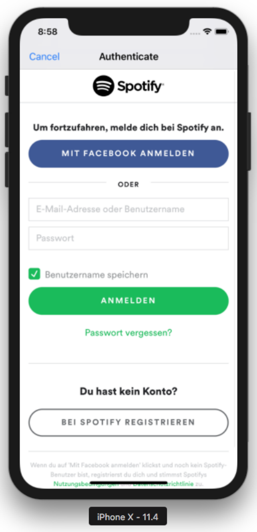
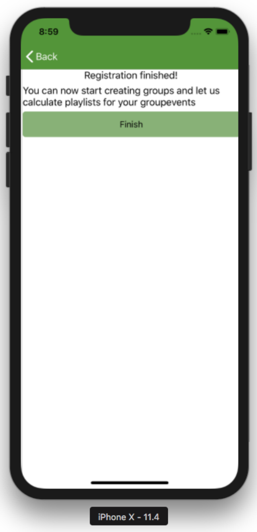


Figure 4.3.: M'Usic Spotify Registration Screen on iOS



Registration finished!
You can now start creating groups and let us calculate playlists for your groupevents

Figure 4.4.: M'Usic Successfull Registration Screen on iOS

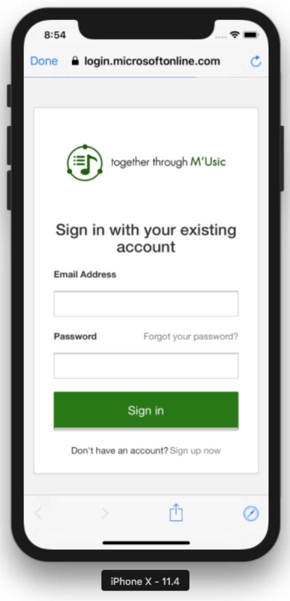


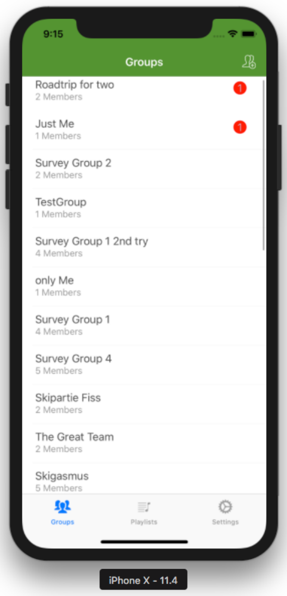
Figure 4.5.: M’Usic LogIn Screen on iOS

Login

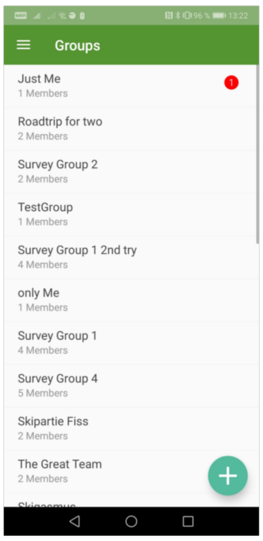
If a user is already registered and starts the app on a smartphone for the first time, or even after a logout, the app shows a login page (4.5). In this case, the user can insert his credentials and press the *Sign in* button. In the case of a correct input, the *groups* page is shown (4.6) listing all the groups in which the user is a member.

Logout

To logout, the user has to open the *settings* page (4.7). On pressing the *Logout* button, the app logs out the user and deletes the stored user information for automatic sign in.

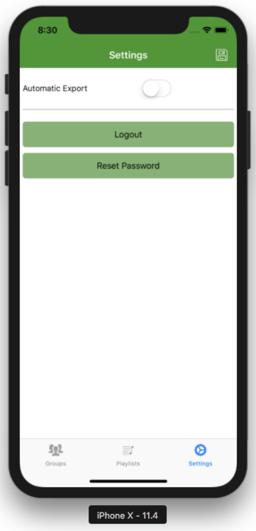


Roadtrip for two
2 Members
Just Me
1 Members
Survey Group 2
2 Members
TestGroup
1 Members
Survey Group 1 2nd try
4 Members
only Me
1 Members
Survey Group 1
4 Members
Survey Group 4
5 Members
Skipartie Fiss
2 Members
The Great Team
2 Members
Skigasmus
5 Members



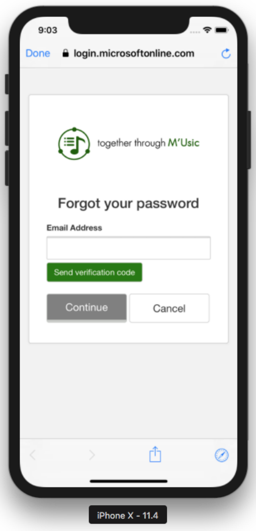
Just Me
1 Members
Roadtrip for two
2 Members
Survey Group 2
2 Members
TestGroup
1 Members
Survey Group 1 2nd try
4 Members
only Me
1 Members
Survey Group 1
4 Members
Survey Group 4
5 Members
Skipartie Fiss
2 Members
The Great Team
2 Members

Figure 4.6.: M'Usic GroupList Screens



Automatic Export

Figure 4.7.: M'Usic Settings Screen on iOS



Forgot your password
Email Address

Figure 4.8.: M'Usic Forgot or Reset Password Screen on iOS

Forgot/Reset Password

If a user has forgotten the password or he/she needs the change of the password, the user can renew this. If the user has forgotten his password during the login process, it is possible to press the *Forgot your password?* link and get forwarded to the *forgot password* page (4.8). The same page is shown, if the user press on the *settings* page (4.7) *Reset Password*. On the landing page, the user can send a verification code to his email address confirming his identity and enter a new password.

4.1.2. Menu

The menu differs between *iOS* and *Android*. The screens are shown in Figure 4.9. In *iOS* the user can switch between the different sections by selecting the respective item on the bottom section. *Android* user have to open the menu by clicking on the menu bar icon on the upper left and select the respective menu item.

4.1.3. GroupManagement

The *groups* page (4.6) is the first page the user sees after the login. It shows the list of groups the user is member. On this page the user has the possibility to open a single group detail page, create a new group, or switch to another part of the app over the menu. By clicking on one of the list items, the *group detail* page (4.10) of the corresponding group item opens. This page shows the group information and the list of playlists, and additionally, the ones, the calculation process has not finished yet. From this page, the user can visit the members page of the group, create a new request for a group music recommendation or go back to the *groups* page.

Create a Group

A user can create a new group by pressing the *plus* button on the *groups* page (4.6). The *add member to group* page (4.11) opens. The user can add

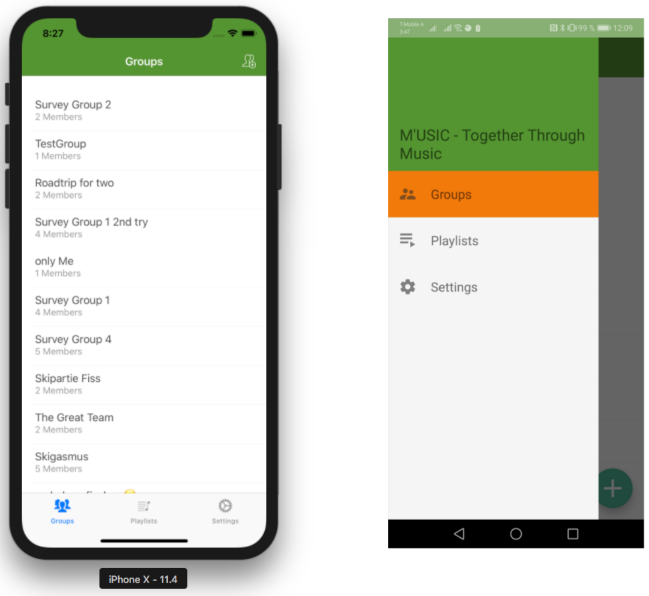
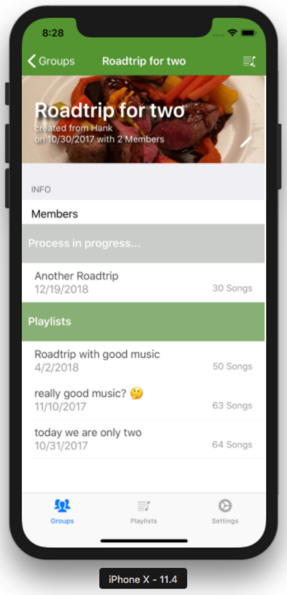


Figure 4.9.: M'Usic Menu Screens

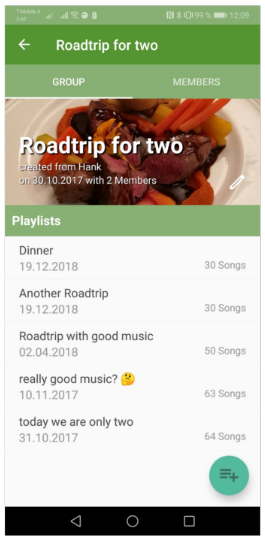
members by selecting known users in the list or add them by entering the email address or the username of registered users. By clicking the *right arrow* button, the user can fill the group form, containing the name of the group and optionally a group image on the *add group details* page (4.12). The process of creating a new group can be finished by pressing the *Create new Group* button. Every member of the group gets a notification that the new group is created.

Manage Group Members

The members of a group can only be managed by the creator of a group. To enter the *manage group members* page (4.14), the founder has to click on the *manage group members* button on the members page (4.13) of a group. The button is only visible for creators of a group. On this page, the user can remove or add members by selecting the items of the user-list or entering

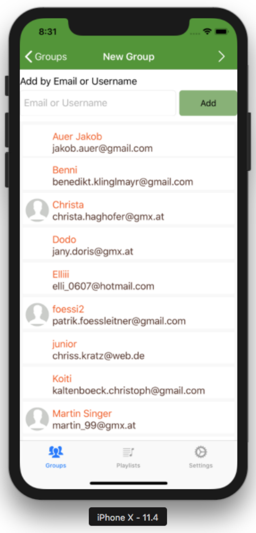


8:28
Groups Roadtrip for two
Roadtrip for two
created from Hank
on 10/30/2017 with 2 Members
INFO
Members
Process in progress...
Another Roadtrip 12/19/2018 30 Songs
Playlists
Roadtrip with good music 4/2/2018 50 Songs
really good music? 🤔 11/10/2017 63 Songs
today we are only two 10/31/2017 64 Songs
Groups Playlists Settings
iPhone X - 11.4



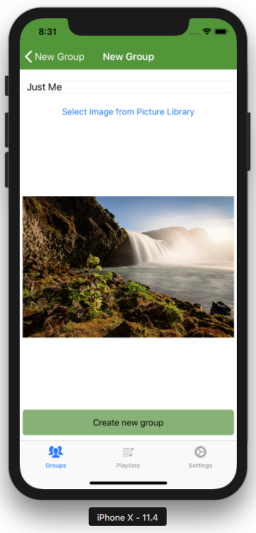
Roadtrip for two
GROUP MEMBERS
Roadtrip for two
created from Hank
on 30.10.2017 with 2 Members
Playlists
Dinner 19.12.2018 30 Songs
Another Roadtrip 19.12.2018 30 Songs
Roadtrip with good music 02.04.2018 50 Songs
really good music? 🤔 10.11.2017 63 Songs
today we are only two 31.10.2017 64 Songs

Figure 4.10.: M'Usic Group Detail Screens



8:31
Groups New Group
Add by Email or Username
Email or Username Add
Auer Jakob jakob.auer@gmail.com
Benni benedikt.klinglmayr@gmail.com
Christa christa.haghofer@gmx.at
Dodo jany.doris@gmx.at
Elliii elli_0607@hotmail.com
foessi2 patrik.foessleitner@gmail.com
junior chriss.kratz@web.de
Koiti kaltenboeck.christoph@gmail.com
Martin Singer martin_99@gmx.at
Groups Playlists Settings
iPhone X - 11.4

Figure 4.11.: M'Usic New Group Members Screen on iOS



8:31
New Group New Group
Just Me
Select Image from Picture Library
Create new group
Groups Playlists Settings
iPhone X - 11.4

Figure 4.12.: M'Usic New Group Name and Image Screen on iOS

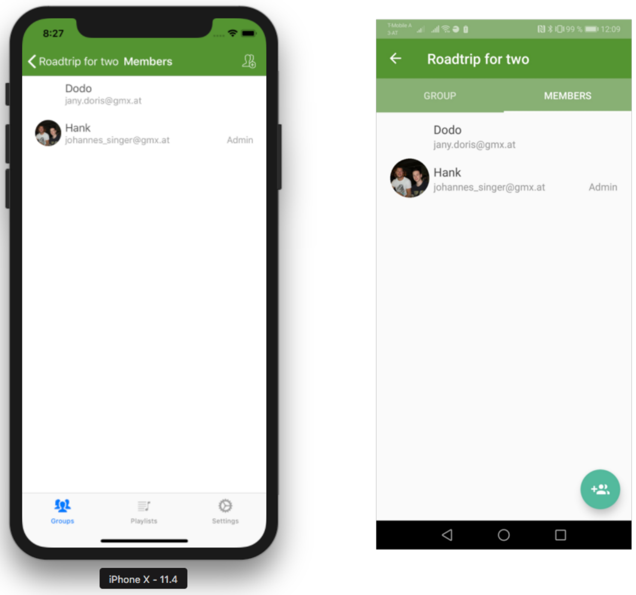


Figure 4.13.: M'Usic Group Members Screens

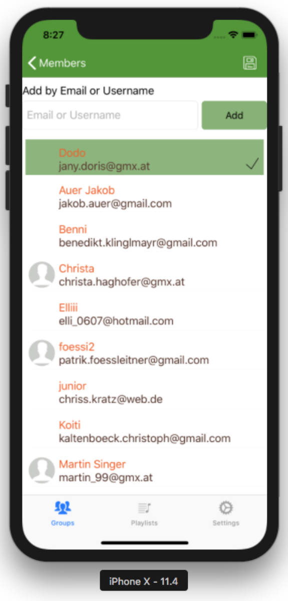
the email address or username in the search field. The changes are adopted by the click on the *save* button.

Delete or Leave Group

The creator can delete a group and every other member can leave a group by pressing long on the group item in the *groups* page (4.15) on Android, or by sliding the item to the left on iOS. A button, titled either *delete group* or *leave group*, appears. By pressing this button, the user either deletes or leaves the group.

4.1.4. Recommendations

There are two possible ways to see recommendations in *M'Usic*. First, on the *group detail* page (4.10) there is a list of recommendations according



Dodo ✓
jany.doris@gmx.at

Auer Jakob
jakob.auer@gmail.com

Benni
benedikt.klinglmayr@gmail.com

Christa
christa.haghofer@gmx.at

Elliii
elli_0607@hotmail.com

foessi2
patrik.foessleitner@gmail.com

junior
chriss.kratz@web.de

Koiti
kaltenboeck.christoph@gmail.com

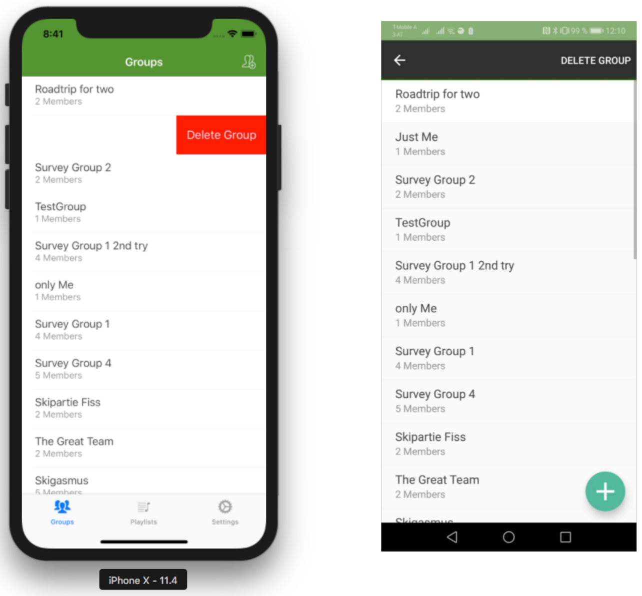
Martin Singer
martin_99@gmx.at

Figure 4.14.: M’Usic Manage Members Screen on iOS

to the specific group. Second, on the *playlists* page (4.16), which shows all recommendations of groups the user is a member. In both pages, the list is split in finished recommendation calculations called *playlist* and requests of playlists not completed at this point.

Request for a Recommendation

To create a new request for a group recommendation, the user has to open the corresponding group detail page. Pressing the *plus* button opens the *New Playlist* page (4.17). The user enters the data of the *Event name*, *Event date* and *Expected starting time*. The click on the *Create new Playlist* starts the calculation of the request.


8:41

Groups

Roadtrip for two
2 Members

Delete Group

Survey Group 2
2 Members

TestGroup
1 Members

Survey Group 1 2nd try
4 Members

only Me
1 Members

Survey Group 1
4 Members

Survey Group 4
5 Members

Skipartie Fiss
2 Members

The Great Team
2 Members

Skigasmus
5 Members

DELETE GROUP

Roadtrip for two
2 Members

Just Me
1 Members

Survey Group 2
2 Members

TestGroup
1 Members

Survey Group 1 2nd try
4 Members

only Me
1 Members

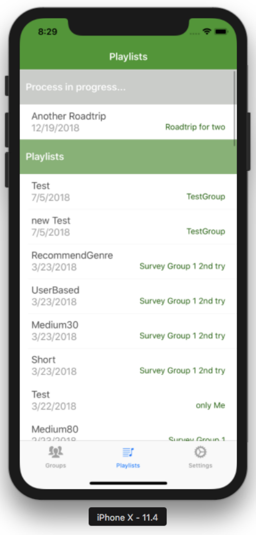
Survey Group 1
4 Members

Survey Group 4
5 Members

Skipartie Fiss
2 Members

The Great Team
2 Members

Figure 4.15.: M'Usic Delete Group Screens


8:29

Playlists

Process in progress...

Another Roadtrip
12/19/2018 Roadtrip for two

Playlists

Test
7/5/2018 TestGroup

new Test
7/5/2018 TestGroup

RecommendGenre
3/23/2018 Survey Group 1 2nd try

UserBased
3/23/2018 Survey Group 1 2nd try

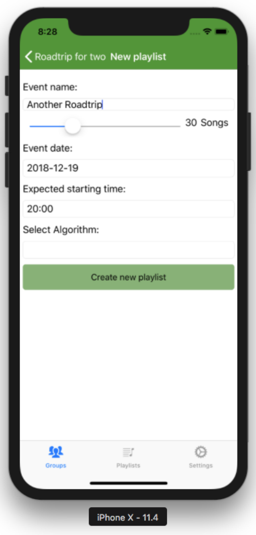
Medium30
3/23/2018 Survey Group 1 2nd try

Short
3/23/2018 Survey Group 1 2nd try

Test
3/22/2018 only Me

Medium80

Figure 4.16.: M'Usic Playlists Screen on iOS


8:28

Roadtrip for two New playlist

Event name:
Another Roadtrip

30 Songs

Event date:
2018-12-19

Expected starting time:
20:00

Select Algorithm:

Create new playlist

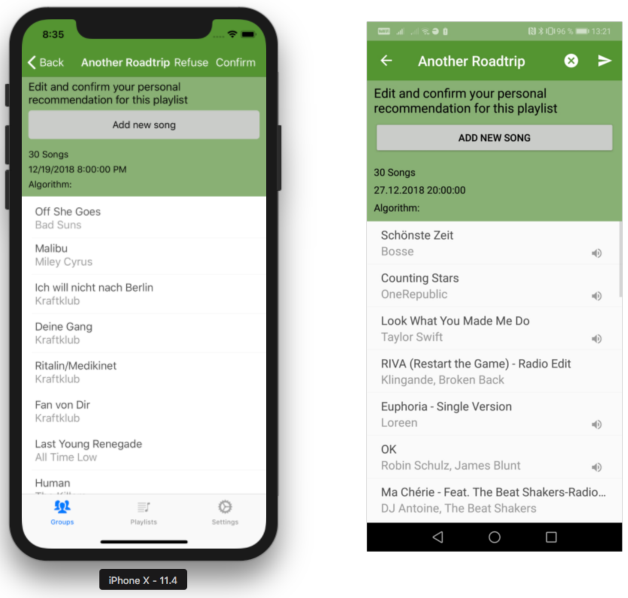
Figure 4.17.: M'Usic New Playlist Screen on iOS

Personal Recommendation

In the first step of a group recommendation, each user gets a notification for receiving his/her personal recommendation. This is a specific item collection containing tracks the app would suggest for the group in the name of the user and is shown on the *personal recommendation* page (4.18). The user has the possibility to manipulate this collection by adding and deleting items. Adding items is done by clicking the *Add new Song* button. This opens the *Add new Song* page (4.19) with a search field on the top of the page. Inserting a part of the name of a track or an artist of a track and pressing the *search* button lists all found songs on *Spotify*. The user can add now one of these songs by selecting the item. Deleting is done in Android by pressing the item for a short time, opening a context menu, and clicking the *delete* button. In iOS, a slide to the left side on the item opens the context menu, and by clicking on the appearing *delete* button, the item is removed shown in Figure 4.20. Most of the items have an icon on the list item showing a speaker. This symbol shows that an audio sample for the song is available. The app starts playing thirty seconds of the track with a click on the item. It stops on a second click on the item. On pressing the *right arrow* button, the list is confirmed. There are also opportunities to refuse the collection or to ignore it. Refusing the collection removes the user from the members of the group, but only for this recommendation and not for all the following. This result is not taking care of the taste of the user in the final playlist, and additionally, the possibility of the user to see the finished playlist. A reason for refusing a track suggestion would be that the user is not taking part in consuming the playlist in the special event. A service checks the personal recommendations three hours before the event starts. If a user has either refused nor confirmed the suggestion list, the service confirms the personal recommendation in the name of the user.

Playlist

At the moment every member has confirmed or refused the personal recommendation, the group aggregation algorithm starts and calculates the resulting recommendation. Every remaining member gets notified and can


8:35

Back Another Roadtrip Refuse Confirm

Edit and confirm your personal recommendation for this playlist

Add new song

30 Songs
12/19/2018 8:00:00 PM
Algorithm:

Off She Goes
Bad Suns

Malibu
Miley Cyrus

Ich will nicht nach Berlin
Kraftklub

Deine Gang
Kraftklub

Ritalin/Medikinet
Kraftklub

Fan von Dir
Kraftklub

Last Young Renegade
All Time Low

Human

Groups Playlists Settings

iPhone X - 11.4

Another Roadtrip

Edit and confirm your personal recommendation for this playlist

ADD NEW SONG

30 Songs
27.12.2018 20:00:00
Algorithm:

Schönste Zeit
Bosse

Counting Stars
OneRepublic

Look What You Made Me Do
Taylor Swift

RIVA (Restart the Game) - Radio Edit
Klingande, Broken Back

Euphoria - Single Version
Loreen

OK
Robin Schulz, James Blunt

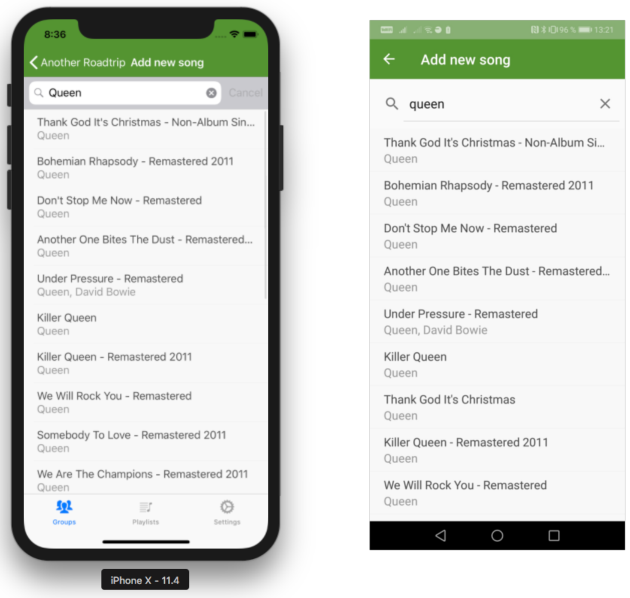
Ma Chérie - Feat. The Beat Shakers-Radio...
DJ Antoine, The Beat Shakers

Figure 4.18.: M'Usic Personal Recommendation Screens

view the playlist on the *playlist detail* page (4.21). If a speaker icon is displayed on an item, the user can play an audio example of the track with a click on them. To listen to the whole playlist in full length, the user can press the *share* button to transfer the list to *Spotify*. The user gets a notification when the transmission is finished. The playlist is then available on *Spotify* in the section Playlists with the name of the event.

4.2. Architecture

In this section, the used technologies and the interaction between the technologies is described. Figure 4.22 shows an overview of the infrastructure and the connection between the components.


8:36

Another Roadtrip Add new song

Queen Cancel

Thank God It's Christmas - Non-Album Sin...
Queen

Bohemian Rhapsody - Remastered 2011
Queen

Don't Stop Me Now - Remastered
Queen

Another One Bites The Dust - Remastered...
Queen

Under Pressure - Remastered
Queen, David Bowie

Killer Queen
Queen

Killer Queen - Remastered 2011
Queen

We Will Rock You - Remastered
Queen

Somebody To Love - Remastered 2011
Queen

We Are The Champions - Remastered 2011
Queen

Groups Playlists Settings

iPhone X - 11.4

Add new song

queen

Thank God It's Christmas - Non-Album Si...
Queen

Bohemian Rhapsody - Remastered 2011
Queen

Don't Stop Me Now - Remastered
Queen

Another One Bites The Dust - Remastered...
Queen

Under Pressure - Remastered
Queen, David Bowie

Killer Queen
Queen

Thank God It's Christmas
Queen

Killer Queen - Remastered 2011
Queen

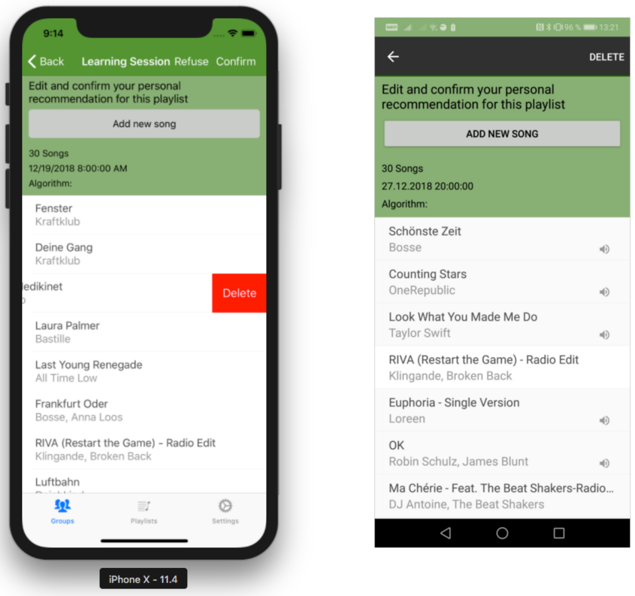
We Will Rock You - Remastered
Queen

Figure 4.19.: M’Usic Add New Song to Personal Recommendation Screens

4.2.1. Xamarin

Xamarin[2] is a tool to develop cross-platform mobile applications. The whole code is written in C# and allows to use its native frameworks. It is developed to implement apps for iOS, Android and Windows Phone. There are different possibilities to develop the user-interface for these apps. Either separately for each platform or by using *Xamarin.Forms*. This allows using the same code for all platforms. Xamarin is developed by Microsoft[3]. As a result, there is a very good integration of connections to tools of Microsoft’s cloud solution Azure[4]. The advantage of Xamarin is the increased productivity developing of one single app instead of one for each platform and the performance benefit compared to HTML with Javascript solutions. Peppers, Taskos, and Bilgin, 2016

[2]https://dotnet.microsoft.com/apps/xamarin
[3]https://www.microsoft.com/
[4]https://azure.microsoft.com/

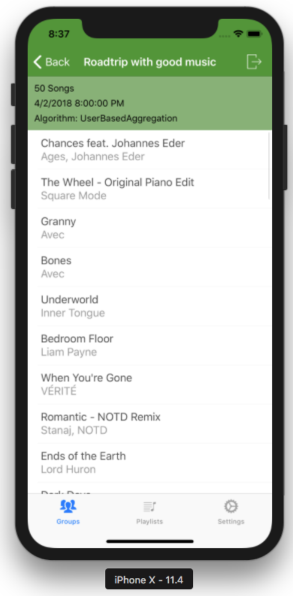


9:14
Back Learning Session Refuse Confirm
Edit and confirm your personal recommendation for this playlist
Add new song
30 Songs
12/19/2018 8:00:00 AM
Algorithm:
Fenster
Kraftklub
Deine Gang
Kraftklub
edikinet
Delete
Laura Palmer
Bastille
Last Young Renegade
All Time Low
Frankfurt Oder
Bosse, Anna Loos
RIVA (Restart the Game) - Radio Edit
Klingande, Broken Back
Luftbahn
Groups Playlists Settings
iPhone X - 11.4

DELETE
Edit and confirm your personal recommendation for this playlist
ADD NEW SONG
30 Songs
27.12.2018 20:00:00
Algorithm:
Schönste Zeit
Bosse
Counting Stars
OneRepublic
Look What You Made Me Do
Taylor Swift
RIVA (Restart the Game) - Radio Edit
Klingande, Broken Back
Euphoria - Single Version
Loreen
OK
Robin Schulz, James Blunt
Ma Chérie - Feat. The Beat Shakers-Radio...
DJ Antoine, The Beat Shakers

Figure 4.20.: M'Usic Delete Song of Personal Recommendation Screens

It is highly recommended to use the design pattern *MVVM* in *Xamarin*. *MVVM* stands for *Model View ViewModel*. The *Model* layer is responsible for the business objects, and therefore it is the backend business logic of the application. Additionally, the *Model* manages all connections to external parts of the app. This can be a web request or an interface to hardware sensors of the mobile phone. The *View* is the layer for the user interface. It manages the interaction with the user and is responsible for all items being displayed. Additionally, it handles the events fired by user actions. The *ViewModel* is the link between the *Model* and the *View*. It sends the data created by the user from the *View* to the *Model* and provides the needed data from the *Model* for the *View*. Additionally, it calls operations of the *Model* triggered by the *View*. Peppers, Taskos, and Bilgin, 2016



8:37

Back Roadtrip with good music

50 Songs
4/2/2018 8:00:00 PM
Algorithm: UserBasedAggregation

Chances feat. Johannes Eder
Ages, Johannes Eder

The Wheel - Original Piano Edit
Square Mode

Granny
Avec

Bones
Avec

Underworld
Inner Tongue

Bedroom Floor
Liam Payne

When You're Gone
VÉRITÉ

Romantic - NOTD Remix
Stanaj, NOTD

Ends of the Earth
Lord Huron

Figure 4.21.: M'Usic Playlist Detail Screen on iOS

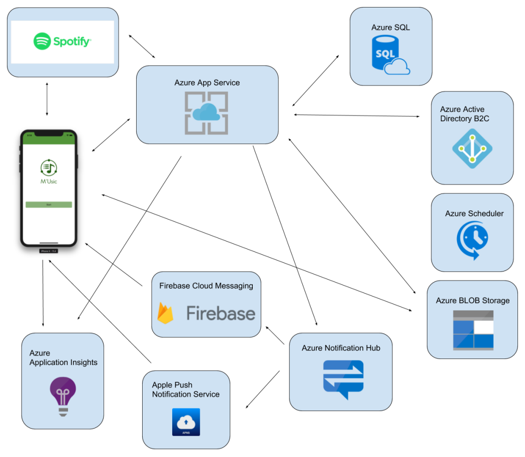


Figure 4.22.: M'Usic Infrastructure Overview

Xamarin.Forms

Xamarin.Forms is a framework for *Xamarin* to develop a single user interface for different mobile platforms. It can be implemented in pure C# code or in the markup language Extensible Application Markup Language (XAML). This cross-platform solution allows to write most of the code once, and additionally, enables the possibility to design custom elements for different platforms. This is necessary to give the user a native app experience, caused by different design guidelines of each platform. Xamarin.Forms supports the development by custom renderers, styling the provided elements differently for each platform. Peppers, Taskos, and Bilgin, 2016

Figure 4.10 shows screenshots of the same page of the app *M'Usic*. The first is the Android app, the second one shows the iOS app. With Xamarin.Forms such look and feel of native apps is easy to achieve.

Android

Although *Xamarin* in combination with *Xamarin.Forms* allows to write most of the code once, some parts have to be written for the Android version of the app *M'Usic* separately:

- Authentication - To open the view for the authentication with *Spotify*, the webpage has to be displayed in a browser. This browser is platform specific for Android. Additionally, the account storage to remember the users' account information is developed separately for Android.
- Images - To load images stored locally on the mobile phone, the image selector and the import had to be developed specific for Android separately.
- Audio - To play and stop song previews in the app, a specifically service had to be implemented in Android.
- Notification - To get notifications, Android uses a service called Firebase Cloud Messaging (FCM). This is the access point to retrieve the notifications. The implemented service manages the registration of the respective Android phone and the appearance and handling of the receiving notification.

- Floating Action Button - It is a commonly used button in Android to trigger an action in a list view. The round button has an icon and is placed on the bottom right of a scrollable list view. This kind of button is not part of the provided elements of Xamarin.Forms and is therefore self-implemented.
- Icons - Every icon in the app has to be provided in different resolutions in special intended folders in the Android app project.

iOS

As in Android, there are a few adaptions on the app *M'Usic* necessary to achieve a look and feel of a native app:

- Authentication - As for Android, a service for the authentication on *Spotify* had to be implemented. In case of a successful login, the authentication information is stored in the iOS specific account store.
- Images - The image selection and import service is implemented for iOS separately.
- Audio - As in Android to play previews of the song, an extra audio service was necessary for iOS.

4.2.2. Azure

Microsoft Azure[5] is a cloud solution and is the optimal back end for apps developed with Xamarin. This is caused by the fact that both products are from Microsoft[6] and so the integration of Azure in Xamarin is intuitive. In the following, the resources of Azure, used for the app *M'Usic*, are introduced.

[5]https://azure.microsoft.com/
[6]https://www.microsoft.com/

App Service

The App Service is the core service and connection point to all other parts of the whole solution. It contains the interfaces to the app, the database, *Spotify* and to all other following listed parts. In the App Service solution, the algorithm for the whole recommendation process is implemented.

SQL-Database

The database contains all information of users, except the security critical authorization data, groups, tracks, and all information regarding the recommendation process. Only the App Service has a connection to the database and is authorized to get, create, update and delete data.

B2C Tenant

The B2C Tenant was designed to guarantee safety and secure management of accounts for a business to customer relationship. It provides the storage of the accounts and manages all use cases connected to the authentication of the user. This can be the registration, login, logout or password reset process. The B2C Tenant uses the Azure Active Directory in the background to store the accounts and provide only interfaces for the App Service and the *M'Usic* App itself.

Scheduler

The Scheduler is a timing job, running every hour. It checks all currently calculated playlists if there are personal recommendations that are still not answered. When the start event of these playlists is closer than three hours, the job confirms all open requests. This triggers the *App Service* to start the group aggregation algorithm.

Notification Hub

The *Notification Hub* is the connection point between app and *App Service* for notifications. The app registers the client on the *Notification Hub* and the *App Service* sends the notifications to the hub. This manages the connection with the FCM and the iOS notification provider which are sending the notifications to the mobile phones.

Blob Storage

The Blob Storage is designed to store files in a cloud storage. The app *M'Usic* stores the group images in the Blob Storage.

Application Insights

Application Insights is a service of Azure to monitor diverse metrics regarding to performance and failure. It allows to recognize problems of the app or the App Service in an early stage, even when the app is rolled out to end users.

4.2.3. Spotify

Shown in Figure 4.23, *Spotify*[7] is by 2017 the music streaming service with the most paying subscribers worldwide.

This, and the fact that *Spotify* provides the API *Spotify for Developers*[9] to get information about the user and all songs available on *Spotify*, led to the decision to use it as the data source for the app *M'Usic*.

[7]https://spotify.com/
[8]https://www.statista.com/chart/5152/music-streaming-subscribers/
[9]https://developer.spotify.com/

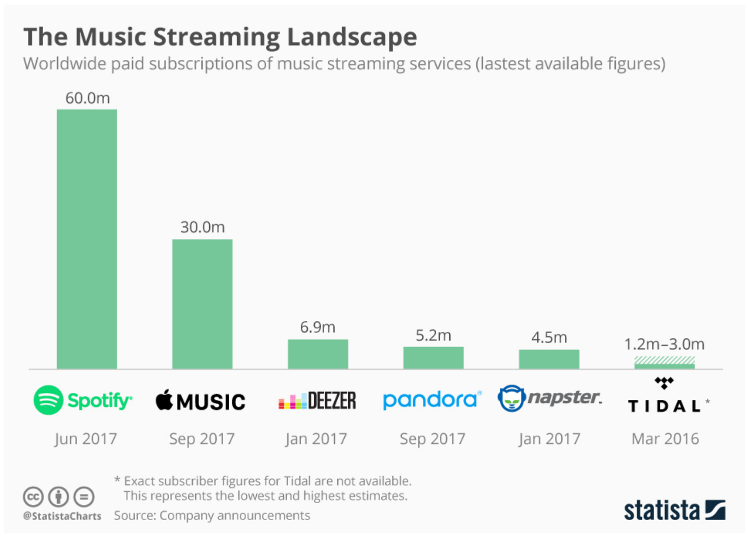


Figure 4.23.: The Music Streaming Landscape[8]

Authorization

To get information over the API of *Spotify*[10], they provide three different authorization flows to grant access. The *implicit grant flow* is a flow designed for JavaScript and only for short-life-circle applications suitable because an access token is not included. The *client credentials flow* is used for user-independent information and is used in server-to-server authentication. A lot of information requested by the app *M'Usic* is user-independent, indeed also available without authorization, but authorized requests can be done more often before they get rejected than unauthorized. The last one is the *authorization code flow* which allows getting long-term access to user information with a onetime user log-in granting permission. This is possible with a refresh token, allowing to get a new access token in case of an expired one. The flow is defined in RFC-6749[11] and is shown in Figure 4.24.

Cause of the appropriate features, the app *M'Usic* has implemented the *authorization code flow*.

[10]https://spotify.com/
[11]https://tools.ietf.org/html/rfc6749#section-4.1

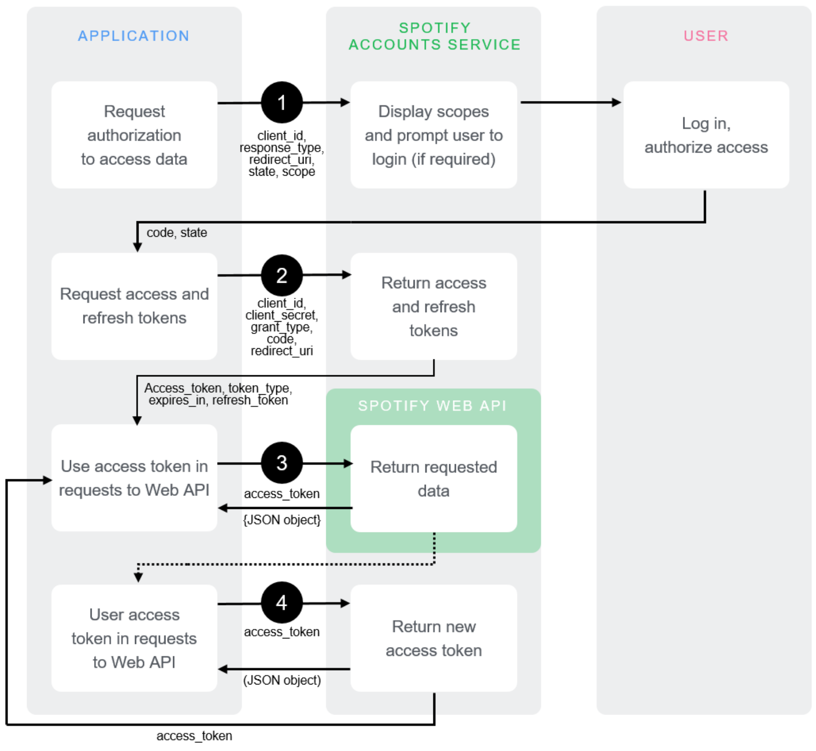

APPLICATION
SPOTIFY ACCOUNTS SERVICE
USER
Request authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Display scopes and prompt user to login (if required)
Log in, authorize access
code, state
Request access and refresh tokens
2
client_id, client_secret, grant_type, code, redirect_uri
Return access and refresh tokens
Access_token, token_type, expires_in, refresh_token
SPOTIFY WEB API
Use access token in requests to Web API
3
access_token
{JSON object}
Return requested data
User access token in requests to Web API
4
access_token
(JSON object)
Return new access token
access_token

Figure 4.24.: Spotify Authorization Code Flow

To protect specific information of users from third-party applications, they have to define the scope of information they need. The user sees the requested scope in the authorization flow and can decide to grand access to this information or not. Each application, which wants to use the API, has to be registered. A signed-up application gets a *Client ID* and a *Client Secret* which are needed in the authorization flow.

Web API

After a successful authorization, the app can use the granted scope of the *Spotify* Web-API. This is based on REST principles and secured by standard HTTPS with the base-address https://api.spotify.com. It provides the HTTP methods *GET*, *POST*, *PUT* and *DELETE*. The API is protected with a *Rate Limit* to avoid that the API block the request of the app. The structure of objects returned by the endpoints is defined in an *Object Model* and is formatted in *JSON*. There are several libraries of the API for different programming languages available.

4.3. Data Source

When recommending music to groups it is a big advantage to know the preferences of the group members without asking them. *Spotify*, as mentioned in Section 4.2.3, is the biggest music streaming service, and therefore the largest knowledge base for the music preferences of users. Section 4.3.1 shows the available information about the music taste of users over the *Spotify* API. Section 4.3.2 further lists the contextual information of the tracks provided by *Spotify*.

4.3.1. Provided Information

With the permission of the user, the *Spotify* API provides different interfaces to get information about the user and his/her preferences. The following list is an extract of endpoints of the API which are used in the app *M'Usic*:

- **User's Top Artists and Tracks** This request provides either the users top artists or tracks. With the help of parameters, the number of artists or tracks can be defined, and also the *time range*. The *time range* can be *long term* which are the top tracks or artists from several years ago till now, *medium term* includes the listened songs from the last six months, or the top tracks or artists of the last four weeks defined as *short term*.
- **User's Followed Artists** This endpoint provides a list of all artists a user is following.
- **User's Playlist** A list of a user's playlist can be requested over this endpoint.

Additionally, the following non-personalized information is provided by *Spotify*:

- **List of Featured Playlists** This endpoint returns a list of featured playlists specified by the country and the timestamp. This allows to get appropriate tracks for a certain time and location.
- **Recommendations Based on Seeds** To get similar tracks the API offers this endpoint. The *seeds* can be artists, genres, or tracks, and are in sum up to five. They and a *tunable track* are the base to get the similar tracks. The properties of the *tunable track* are concurrent to the *Audio Features Object* which is explained in Section 4.3.2.
- **Search for an Item** The *Spotify* API provides a search for playlists, tracks, or artists.
- **Audio Analysis for a Track** This endpoint provides an audio analysis of a track and is described in detail in Section 4.3.2.
- **Audio Features for a Track** Audio Features contain context information of a track. This endpoint is described in Section 4.3.2.

Finally, there are two endpoints to create a playlist containing the recommended tracks of a group on *Spotify* to provide the possibility to listen to the playlist.

- **Create a Playlist** By this endpoint, it is possible to create an empty playlist.
- **Add Tracks to a Playlist** This endpoint is used to add tracks to a user's playlist.

4.3.2. Context of a Track

The *Spotify* API defines several object models for receiving and sending in *JSON* format. The following two objects are described, the *Audio Feauters Object* and the *Audio Analysis Object*. Both represent a track with different contextual information. Based on this information, the recommendation algorithm of the app *M'Usic* calculates results.

Audio Features Object

The object values described in this section are an extract of the *audio features object* only consisting of the important values for the app *M'Usic*. These values are also part of the *tunable track object*, which is therefore not additionally mentioned.

- **acousticness** Measure the accusticness of a track.
- **danceability** The value of danceability is a combination of the elements rhythm stability, tempo, beat strength and overall regularity and described how danceable a track is.
- **energy** The energy is a measure of activity and intensity. A high energy value stands for loud, noisy, and fast tracks.
- **instrumentalness** High values of instrumentalness have no vocals in all probability. Samples for vocal tracks are rap or spoken word tracks, but sounds like "ooh" and "ahh" are in this context treated as instrumental.
- **liveness** If the presence of audience is noticed in a recording, the value of liveness is higher.
- **loudness** The loudness is the average value of loudness in decibels across the entire track.
- **speechiness** Speechiness is the opposite of instrumentalness.
- **tempo** The tempo is the estimated tempo in beats per minute over the whole track.
- **valence** Valence describes if a positive or negative mood is conveyed to the listener of the track. High values stand for happy, euphoric or cheerfull songs, whereas low values indicating a sad, angry or even depressive song.

Audio Analysis

The *audio analysis object* contains all measurable context information, which is not part of the *full track object*. This information is wide-ranging from timbre and pitches to bars and beats.

4.4. Testing and Monitoring

To deliver good software in short update-iterations, continuous integration and delivery is absolutely essential. During the software development process, many developers test applications only in one environment and risk to overlook side effects caused by other environments leading to bugs in the software. *Continuous integration* is a process to test the software immediately on different environments, which allows to write and test the software in small steps going only further in the development if all tests pass. This leads to high quality software from the beginning. *Continuous delivery* goes one step further and includes the software delivery in the process. Only software which passed all tests is automatically deployed in production environment and is rolled out to end users. Versluis, 2017

The app *M’Usic* uses several approaches to reach continuous delivery. Caused by the two main parts, the app and the backend, there are two different processes to deliver the software.

4.4.1. Client-Side Continuous Delivery

To make client-side continuous delivery possible, three steps are necessary. The first step is to write the tests. These are the basis to set up a automated test environment. This environment can release the software to make it available in the app stores.

Writing Tests

Writing the tests is probably the most important step in continuous delivery. The process is only that good as the tests for it. *Xamarin* apps can be tested with *Unit Tests* and *UI Tests*.

- **Unit Tests** A Unit Test is typically split into three steps. The first step is the initialization of the objects and the setting of the values needed for a test. The second step is the act, which is an invoke of the method to test. Finally, step three is the verification, which tests if the behavior of the method was as expected. The main focus of the *Unit Tests* should be on the *MVVM* pattern implementation, testing the interaction between the components, and additionally, the platform-specific functions.
- **UI Tests** *Xamarin* provides a testing framework to test the user interface (UI) of an app called *Xamarin.UITest*. Similar to *Unit Tests*, the *UI Tests* are based on three steps called *Arrange*, *Act* and *Assert*. The framework allows to interact with the UI and therefore to test if the assumed behavior happens.

Test Automation

To set up an environment for automating tests for *Xamarin*, it requires the following two cloud applications:

- **Azure DevOps** Azure DevOps[12] is a cloud-based platform developed to facilitate the development of software in a team. A main part is the provision of repository services. These are the interface to the described following *App Center*. Commits in defined branches of the repository trigger the *App Center* to start tests on these commits.
- **Visual Studio App Center** Visual Studio App Center[13] is also a cloud-based platform which provides a lot of functionality around testing, continuously build, releasing, and monitoring apps for mobile applications. The interaction between the *App Center* and *Azure DevOps* is deeply linked, resulting in an easy setup for automated tests.

[12]https://azure.microsoft.com/en-us/services/devops/
[13]https://appcenter.ms/

The *App Center* allows to test apps on many different smartphone devices, even physically in a hosted device lab. Besides the test results, the *App Center* provides full-resolution screenshots and performance metrics.

Releasing the App

Releasing apps in the *App Stores* is also part of the functionality of the *App Center*. To enable the releasing process it is necessary to connect the *App Center* with the iTunes Connect[14] for apps published in the Apple App Store[15] and with the Google Play Console[16] for publishing apps in Google Play Store[17]. There is the possibility to define different branches in the *Azure DevOps* to publish the apps either in testing channels for bigger groups of testers or in releasing channels to provide the app for everyone.

4.4.2. Monitoring the App

There are two different kinds of monitoring an app. Either by concentrating on crashes, daily user counts by device and session duration. Or on code-base, by logging several states or failing functions insight the app to avoid a crash of the app with nearly endless possibilities of analysis. The *App Center* provides the first variant of monitoring. In *Xamarin* it is possible to use Application Insights[18], a service of the cloud solution *Microsoft Azure*, to integrate a powerful analytic tool.

4.4.3. Server-Side Continuous Delivery

As the app *M’Usic*, the backend server application was developed in an agile development process including continuous integration and later on a

[14]https://itunesconnect.apple.com/
[15]https://www.apple.com/at/ios/app-store/
[16]https://developer.android.com/distribute/console/
[17]https://play.google.com/store
[18]https://azure.microsoft.com/en-us/services/application-insights/

continuous delivery. Similar to the app, the process contains tests, automatic testing, and delivery.

Unit Tests

The backend software has no UI and so there is only the need for testing the code. The principle is the same as in client-side unit tests containing the arrange, act and assert sections. The tests should focus on code coverage, trying to reach every line of code in the tests, and additionally, the boundary conditions. These are tested by initializing the tests with values under, on and above barriers in conditions.

Test Automation

The important point on successful development is to run automatic tests after every small step in the development process and go only further when passing all tests. Versluis, 2017

The backend software of the app *M'Usic* is developed in C# and uses Azure DevOps[19] as the source control system. The advantage of this solution is the provision of services for test automation. This allows to test automatically *commits* on previously defined *branches* without setting up an additional service.

Continuous Deployment

Also, continuous delivery has an underlying premise. Only builds, that are successfully tested by test automation, are released for deploying on productive systems. The app *M'Usic* uses Azure DevOps also to deploy the builds automatically on the server after committing changes of the source code in a specific branch and passing all tests.

[19]https://azure.microsoft.com/en-us/services/devops/

Monitoring

To monitor the backend application, the same tool is used as for the app: Application Insights[20]. It allows implementing a logging system, to supervise the performance of the application and to observe the data transactions between parts of the software and external services. It informs about failures and resource consumption. To reach this, *Application Insights* is deeply integrated in the source-code of the backend application. An additional benefit is to analyze failures of the software by a snapshot debugger, where it is possible for developers to debug code in the same environment as the failure occurred.

[20]https://azure.microsoft.com/en-us/services/application-insights/

5. Evaluation of the Algorithm

There has been an evaluation to validate the performance of the app *M'Usic*. The system under test was a modification of the app to allow the selection of the algorithm. There was the base algorithm and some variations of this. In this chapter the environment, the execution, and the results of the evaluation are shown and explained.

5.1. Evaluation Environment

To get comparable results out of evaluation it is important to define an environment, and additionally, several categories that can be determined from the environment. The evaluation of the app *M'Usic* is defined with regard to the dimensions of *Evaluation Technique*, *Test Users*, and *Execution Environment*.

5.1.1. Evaluation Technique

The evaluation is a mixture of a *User Study* and an *Online evaluation*, both described in Section 2.6.1. The study participants had to perform defined tasks and answered questions at specific points in time during the execution. As the results of the evaluation were investigated under the aspect of a group, all members had to synchronize their tasks. Uncommon for *User Studies*, the participants were not observed and so, only the results of the questionnaires were investigated. Characteristic for the *Online Evaluation* is the use of the final app with variations of the algorithm. This is necessary for the comparability of the modifications without an extraneous influence.

The evaluation was a within-subject study, so each participant tested each variation.

5.1.2. Study Participants

The study participants had to fulfill several criteria, to avoid biases:

- **Familiar with the use of smartphones and apps.** The users have to be confident in dealing with smartphones and apps. This is important to avoid frustration on executing the tasks, which can result in bad scores.
- **Spotify account and regularly listening to music on the platform.** To use the app *M'Usic* a *Spotify* account is necessary. Indeed, the app has a fallback for users not listening to music on *Spotify*, but to get appropriate recommendations, a knowledge about the user's taste is required.
- **Time and pleasure.** Evaluations take time and are not always funny. Therefore, it is important that the participants take part on the evaluation voluntarily, without stress and thinking positive to avoid falsification of the answers.

The participants are between 20 and 29 years old and are all interested in music. All members within a group know each other. Overall there were eleven participants, split into one group of three members and two of four members. As the study was within-subject, all participants tested every variation.

5.1.3. Execution Environment

Execution Environment is in this case defined as *the tools used to perform the evaluation*. This is, on the one hand, the platform, the app is running on, and on the other hand, the survey tool.

Platform

All participants used their own smartphone to run the test. This is an important point because of the familiarity with the platform. The ratio between *Android* and *iPhone* users was balanced. On the smartphones the app *M'Usic* was installed with a small modification, which made it possible to choose different variations of the algorithm on requesting a new playlist.

Survey Tool

The tool LimeSurvey[1] was used to create the survey. It is an open-source application to create online surveys. The participants answered their questions in a form and the results were exported to Microsoft Excel[2]. Closed questions with the possible answers Yes and No were rendered in Excel whereas other questions were rendered with the tool Plotly Online[3].

5.1.4. Questions

The questions of the survey aim at the properties of RSs discussed in Section 2.6.2. The variations of the algorithm are split into different sections in the survey. The basic algorithm section contains all questions whereas the adaptions of the algorithm contain the main questions, and additionally, questions related to the properties, the adaption tries to improve. The closing questions deal with the listening habits and the order of the tracks of the recommended playlist.

Questions about the Algorithm

The following questions were asked according to the results of the personal recommendation and the final playlist recommended by the basic algorithm:

[1]https://www.limesurvey.org/
[2]https://products.office.com/excel
[3]https://plot.ly/

- What is the name of the playlist to answer the questions for this type of algorithm?
- How much do you like your personal recommendation? From 1 (very much) to 5 (not at all)
- How many songs of your personal recommendation fit to your taste in music for this event?
- Do the songs of your personal recommendation correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit)
- Do the songs of your personal recommendation correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit)
- How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre)
- Would you like to get more diversity in your personal recommendation?
- How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good)
- Did you get an unknown song of an unknown interpret which you like in your personal recommendation?
- Did you get an unknown song which you really like in your personal recommendation?
- How many songs of your personal recommendation did you already know?
- Would you like to get more unknown songs in your personal recommendation?
- How much do you like the playlist? From 1 (very much) to 5 (not at all)
- How many songs of the playlist do you like?
- Do the songs of the playlist correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit)
- Do the songs of the playlist correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit)
- How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre)
- Would you like to have more diversity in the playlist?
- How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good)
- Would you use this algorithm again?

Questions about the Order of the Tracks

The following questions try to identify if the study participant usually listens to music in the shuffle mode or in the recommended order, and check the improvement of the order of the tracks in the recommended playlist.

- Are you used to listen to songs mostly in the order of the playlist, or in shuffle mode?
- Please do have a look at the order of the songs in the calculated playlists - Would you listen to these playlists in the recommended order, or in shuffle-mode?

5.2. Evaluation Execution

In this section, the execution of the evaluation is described. It contains the parts *Preparation*, *Conduction*, and *Reflection*.

5.2.1. Preparation

To perform a successful survey, thorough preparation is necessary. This has shown the conduction of the survey with the first test group. It ended with a termination causing multiple reasons: The users were not familiar with the app, so they had troubles to handle it at the beginning. The app and the algorithm were too slow resulting in frustrated test participants. Finally, the participants had no earphones, so for playing the recommended songs all members of the test group had to take the speaker of their smartphones resulting in extremely loud background noise, and no one could concentrate on their songs.

Because of these recently mentioned reasons, the following tasks were defined:

- **Provide the app the test users early enough.** Give the user enough time to get familiar with the app and let them test it and perform real world scenarios to understand the functionality of the app.

- **Performance improvements.** Although the development of the algorithm and its variations were finished, the first test was in a too early stadium of the app development what caused a poor performance. A special effort on increasing performance resulted in an acceptable response time of the recommendations.
- **Let the participants use earphones.** To avoid a loudness overload, it is important that the participants of the study use earphones. Either they bring their own or they have to be provided.
- **Provide activities.** A user study can take up to a few hours and the concentration of the users cannot be that long. It is important to make breaks and provide them activities helping them to calm down and relax to go further focused and efficient after the break.

5.2.2. Survey Conduction

The members of a team met up to perform the survey. They choose a team leader and agreed on a real-life scenario using the app *M'Usic*. The team leader created a group in the app and added all members. For the base algorithm and each adaption the team leader had to create a request for a new playlist with the time of the listening defined in the real-life scenario and the amount of 30 songs. Additionally, the base algorithm was tested with the amount of 50 and 80 songs to proof the change of the results based on the number of tracks. The survey started with some initial questions like the username and the group's name in the app *M'Usic*. For each playlist request, the app calculated a personal recommendation per member. Every section in the survey belongs to a variation of the algorithm and each section is split up in two parts. The personal recommendation was the first one. After they had answered these questions, the members confirmed their personal recommendation without removing or adding songs. This is important to get the results of the algorithm. In the following, all members confirmed their personal recommendation for an algorithm. At this point the corresponding final playlist was calculated. The result is evaluated with questions in the second part of the appropriate algorithm section of the survey. After the participants had answered all questions to the personal recommendation and the resulting playlist of all requested playlists, some closing questions about the listening habits had to be answered.

5.2.3. Reflection

The reflection of the survey resulted in the following insights:

- To conduct a survey a thorough preparation is necessary.
- Various aspects have to be considered in order to ask questions that do not have an influential effect.
- Even participants with a positive attitude towards the survey can get frustrated by too long surveys.

5.2.4. Expected Conclusions

The focus of the evaluation was on the basic algorithm and its variation, and additionally, on the order of the playlist. The variations differ from the original algorithm either on the used user information or follow different approaches to calculate the playlist. These variations have no explicit expected outcome. The aim is to form an aggregated algorithm by combining the approaches of the successful variations. Other variations have the goal to improve some properties of RSs. The expected outcome is to reach better results in the properties of a GRS.

In the following, all algorithms are listed:

MediumTerm

The *MediumTerm* algorithm is the basic one. It was developed first and uses the most listened tracks of the user of the last six months. It also includes techniques trying to recommend songs fitting to the planned date and time of the listening. The tracks for the final playlists are the ones with the lowest distance between the song and the group's *preference model*.

ShortTerm

The *ShortTerm* algorithm differs from the basic algorithm in the data source which is used to calculate the user preferences. It uses the most listened tracks by the user the past four weeks.

LongTerm

The *LongTerm* algorithm distinguishes from the basic algorithm by the use of the top tracks over all time as the basis to calculate the preferences of the users.

UserBasedAggregation

The *UserBasedAggregation* algorithm calculates a specific *preference model* for each member of the group. The tracks of the final playlists are the songs with the lowest sum of distances between a model representing the song and the specific user *preference model*.

TimeIndependent

The basic algorithm implements approaches to reach better results considering the date and time of the event. To proof this approaches, the *TimeIndependent* algorithm was implemented. Both differ only on the approaches based on date and time. The expected conclusion is that the questions relating to the correlation between time and playlist result in poorer rates than the basic algorithm.

Valence

The *Valence* algorithm includes the value of valence of the songs in the recommendation. The sense of valence is described in Section 4.3.2. The expected conclusion is that the algorithm reaches higher ratings in satisfaction as the basic algorithm.

Genre

The *Genre* algorithm prefers songs which are attributable to genres, listened by many members of a group. The expected outcome is a higher overall satisfaction.

RecommendGenre

The *RecommendGenre* algorithm calculates possible songs for the playlist based on the *preference model* of the group correlating to the genres most of the group members listen to.

RecommendArtist

The *RecommendArtist* algorithm calculates possible songs for the personal recommendation of each member which are close to the *preference model* of the group and similar to the top interpreters of the member.

5.3. Evaluation Results

In this section the results of the questions are shown. The yes/no questions are presented in a bar chart, the results of the other questions are shown in box plot diagrams.

The result of the question *How much do you like your personal recommendation?* is shown in Figure 5.1. The variations *ShortTerm*, *LongTerm*, and *Valence*

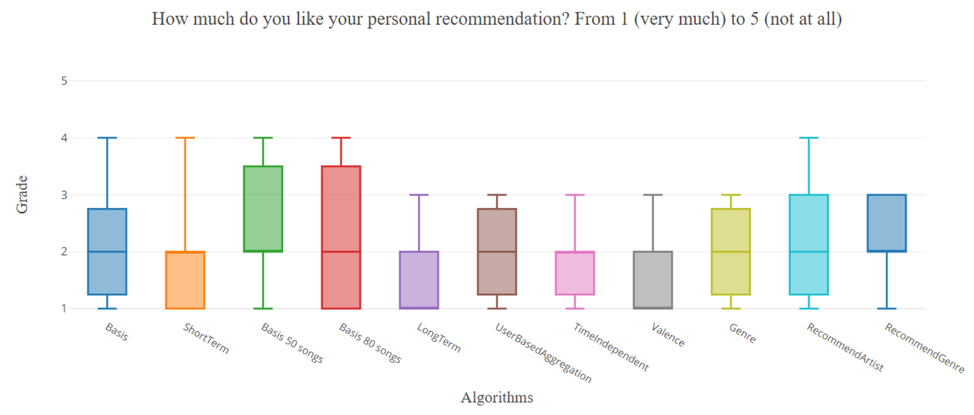


Figure 5.1.: Evaluation Result - How much do you like your personal recommendation?

got the best results. All variations received an at least similar or better outcome.

The result of the question *How many songs of your personal recommendation fit to your taste in music for this event?* is shown in Figure 5.2. The variation *Valence* received the best results. Additionally, there were no negative outliers on the variations *LongTerm* and *Genre*.

The result of the question *Do the songs of your personal recommendation correlate to the time of the event?* is shown in Figure 5.3. Not all questions were asked for each variation. Besides the basic algorithm, questions concerning specific properties were only asked on the variations which try to improve these properties. In this question, the focus was on the accuracy of time. As the basic algorithm is the only algorithm that implemented approaches to improve the recommendations in the accuracy of time, only the variation of *TimeIndepended* was, additionally, evaluated to confirm the approach. Furthermore, the basic algorithm was evaluated with a different number of recommended tracks to proof the independence of them. Unfortunately, there were no significant better results on the basic algorithm than on the *TimeIndependend*.

The result of the question *Do the songs of your personal recommendation correlate to the reason of the event?* is shown in Figure 5.4. Like in 5.3, the basic algorithm did not receive better results than the *TimeIndependend*. All evaluated

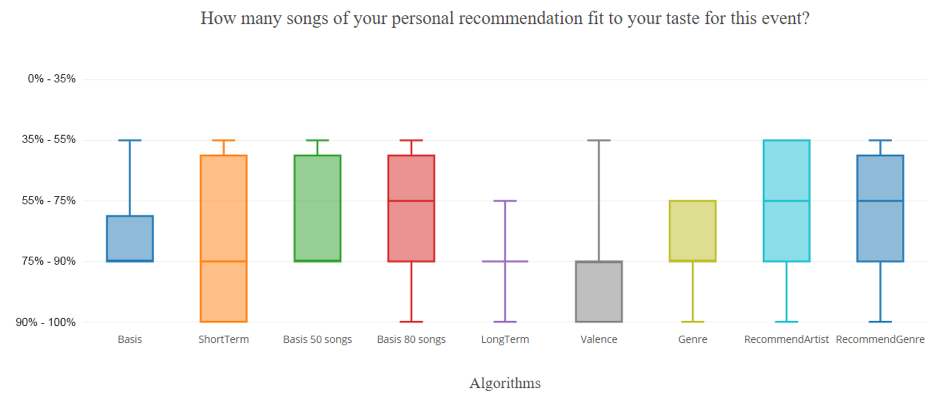


Figure 5.2.: Evaluation Result - How many songs of your personal recommendation fit to your taste in music for this event?

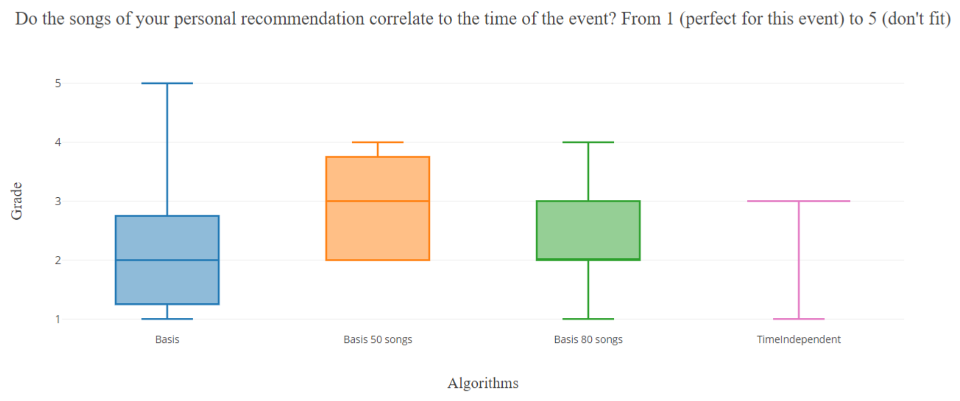


Figure 5.3.: Evaluation Result - Do the songs of your personal recommendation correlate to the time of the event?

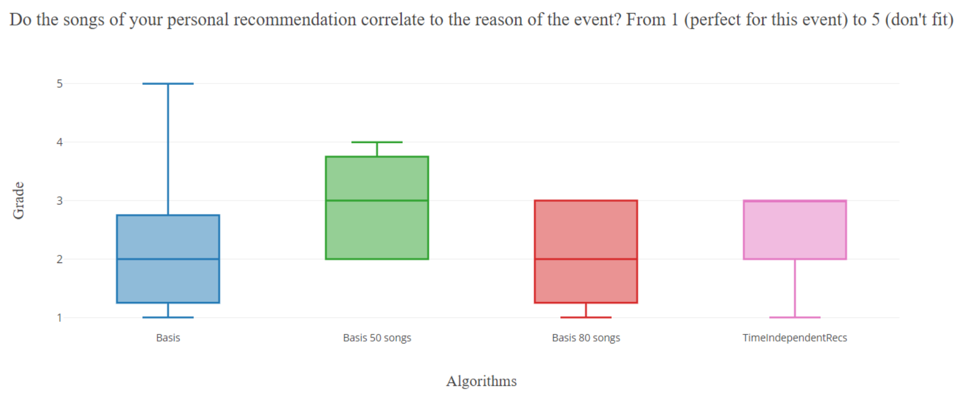


Figure 5.4.: Evaluation Result - Do the songs of your personal recommendation correlate to the reason of the event?

variations received similar results.

The result of the question *How diverse is your personal recommendation?* is shown in Figure 5.5. All evaluated variations received similar results. Only the basic algorithm got a wide range of results containing some high rates of diversity.

The result of the question *Would you like to get more diversity in your personal recommendation?* is shown in Figure 5.6. The figure shows that most of the participants would not get more diversity in any variation. Only in the *RecommendGenre* variation, the difference between the number of yes and no answers is not that high.

The result of the question *How bad is the worst song in your personal recommendation?* is shown in Figure 5.7. The worst results got the basic algorithm and the variations *TimeIndependend* and *ShortTerm*. The variations *Valence* and *Genre* have the fewest bad songs in their recommendation.

The result of the question *Did you get an unknown song of an unknown interpret which you like in your personal recommendation?* is shown in Figure 5.8. Often participants found unknown songs they like in the basic algorithm and the variation *RecommendArtist*.

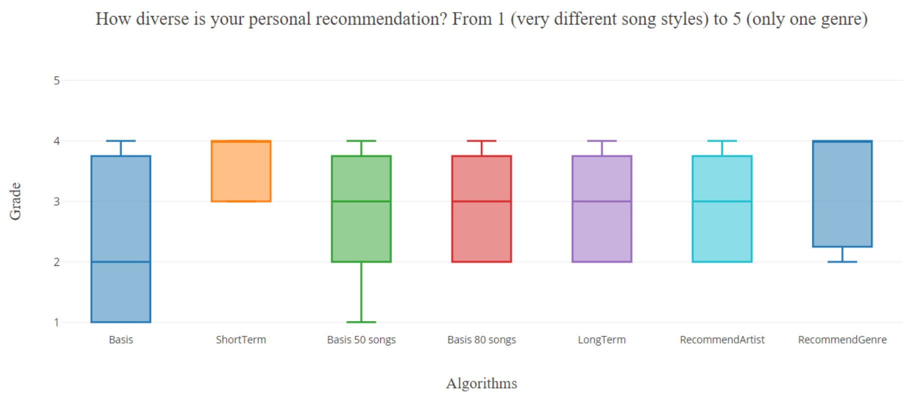


Figure 5.5.: Evaluation Result - How diverse is your personal recommendation?

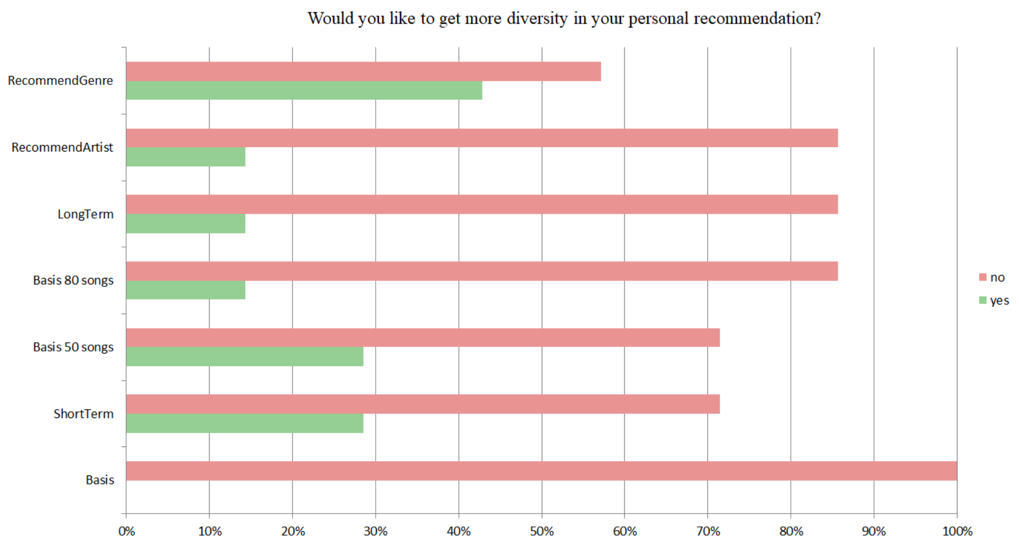


Figure 5.6.: Evaluation Result - Would you like to get more diversity in your personal recommendation?

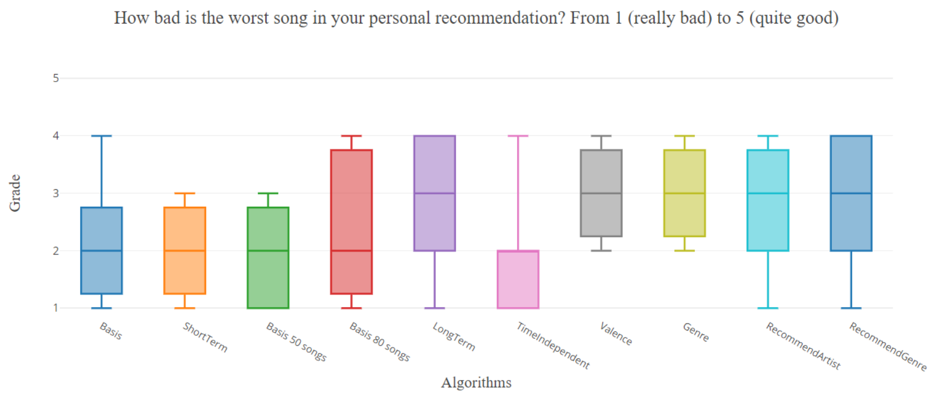


Figure 5.7.: Evaluation Result - How bad is the worst song in your personal recommendation?

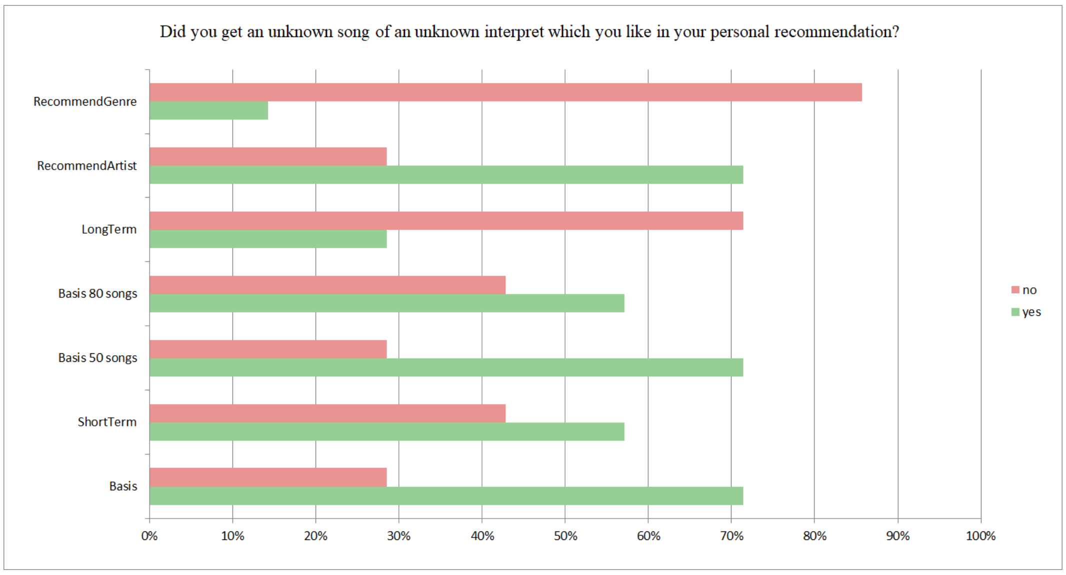


Figure 5.8.: Evaluation Result - Did you get an unknown song of an unknown interpret which you like in your personal recommendation?

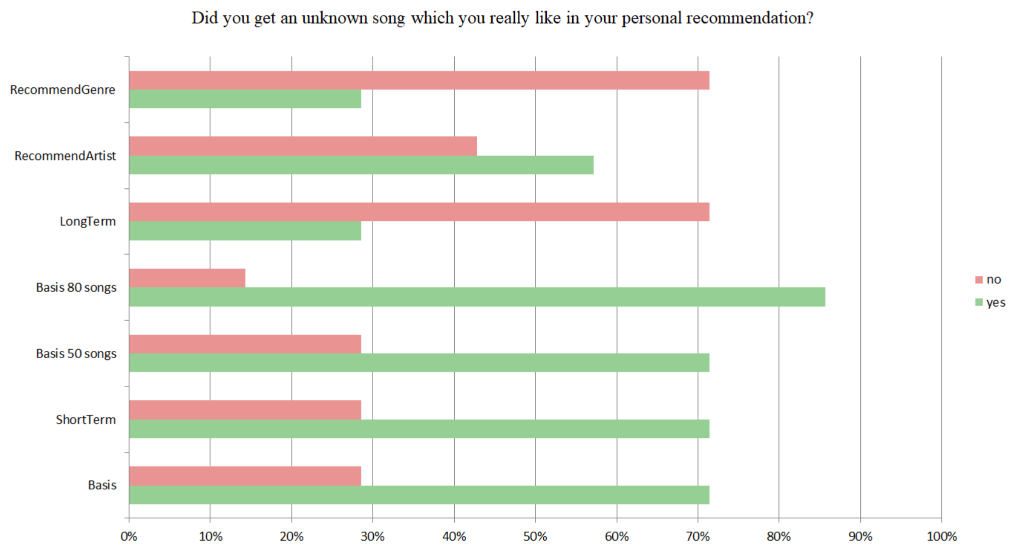


Figure 5.9.: Evaluation Result - Did you get an unknown song which you really like in your personal recommendation?

The result of the question *Did you get an unknown song which you really like in your personal recommendation?* is shown in Figure 5.9. Similar to the previous question, the figure shows that the basic algorithm recommends many unknown songs the users like. Additionally, *ShortTerm* received good results.

The result of the question *How many songs of your personal recommendation did you already know?* is shown in Figure 5.10. The results of the variations are very similar. Only some participants did know fewer songs in the variation *ShortTerm* and the basic algorithm with a higher number of recommended tracks.

The result of the question *Would you like to get more unknown songs in your personal recommendation?* is shown in Figure 5.11. Most of the variations got a slightly higher vote for no further unknown songs. Only the basic algorithm with 50 recommended tracks and the variation *RecommendGenre* got a clear result to avoid further unknown songs.

The result of the question *How much do you like the playlist?* is shown in Figure 5.12. The variation *Valence* is the winner of this evaluation. Addition-

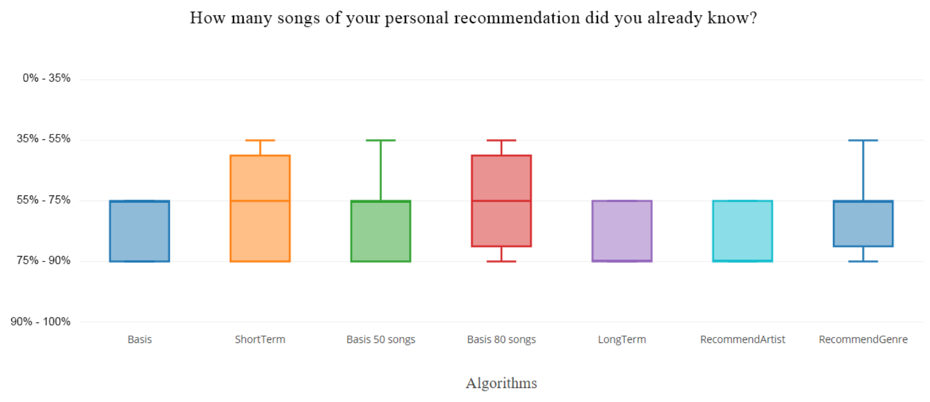


Figure 5.10.: Evaluation Result - How many songs of your personal recommendation did you already know?

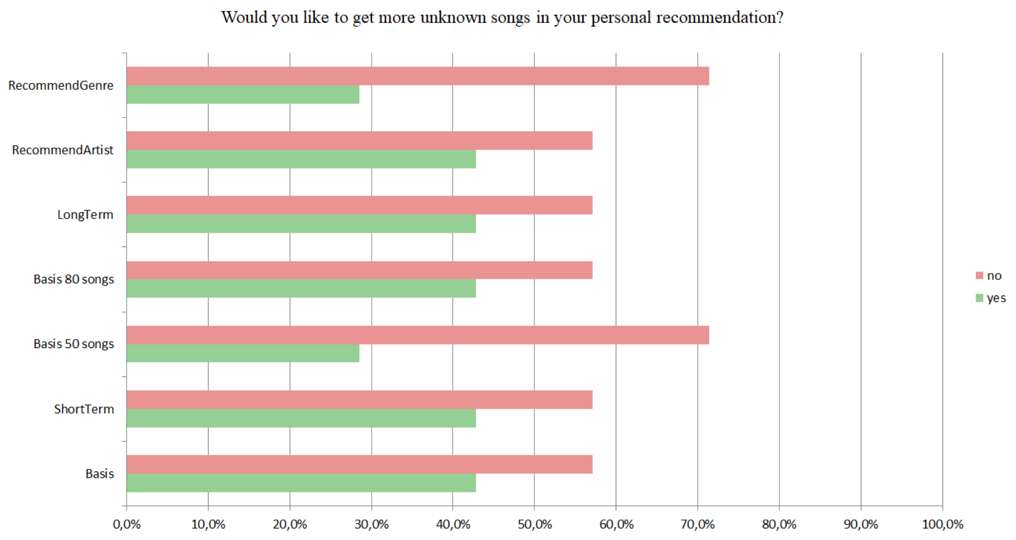


Figure 5.11.: Evaluation Result - Would you like to get more unknown songs in your personal recommendation?

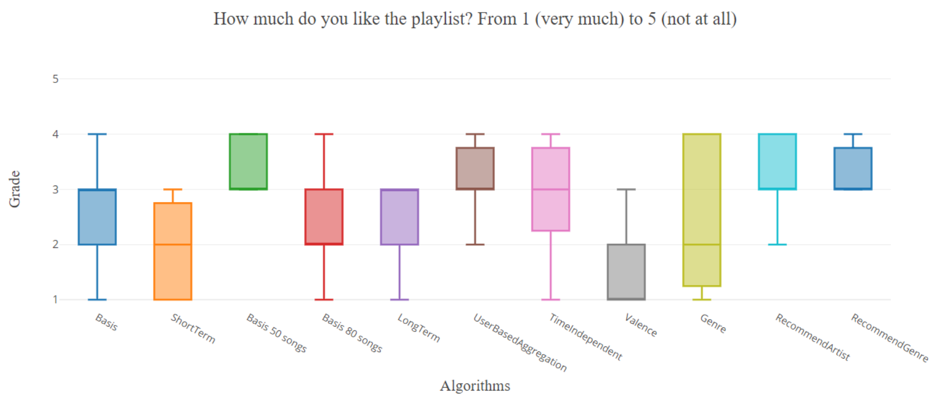


Figure 5.12.: Evaluation Result - How much do you like the playlist?

ally, the variations *ShortTerm* and *Genre* got good results from some of the participants.

The result of the question *How many songs of the playlist do you like?* is shown in Figure 5.13. Only in the variation *Valence*, nearly all participants like the majority of the recommended songs. Some of the participants also liked the majority of the songs in the variation *Genre*, but the result is very widespread.

The result of the question *Do the songs in the playlist correlate to the time of the event?* is shown in Figure 5.14. As the results of the similar question on personal recommendation (5.3) also show, the time-dependent basic algorithm does not achieve better results than the variation *TimeIndependend*.

The result of the question *Do the songs of the playlist correlate to the reason of the event?* is shown in Figure 5.15. As well as in the evaluation of the previous question, the results show no differences between the basic algorithm and the variation *TimeIndependend*.

The result of the question *How diverse is the playlist for you?* is shown in Figure 5.16. The results show that the basic algorithm recommend the most diverse songs. In the variation *ShortTerm* the results are not that clear because the received ratings are very scattered.

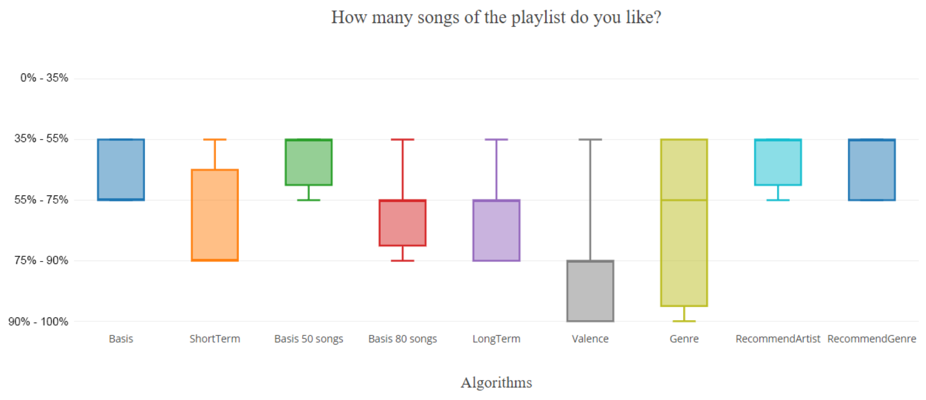


Figure 5.13.: Evaluation Result - How many songs of the playlist do you like?

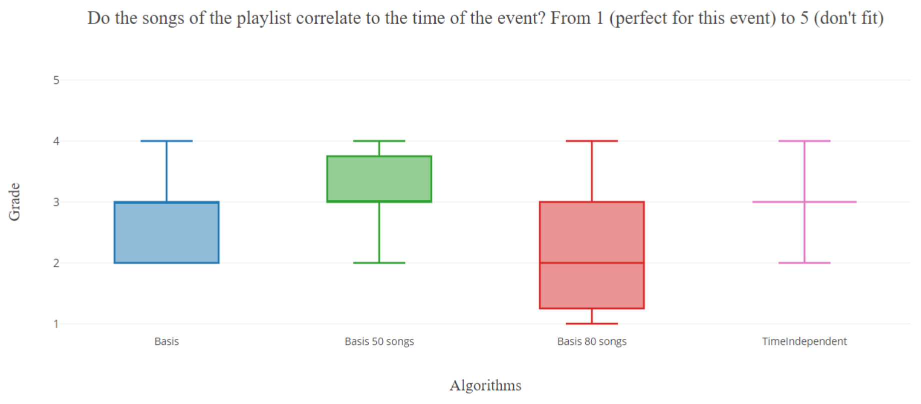


Figure 5.14.: Evaluation Result - Do the songs in the playlist correlate to the time of the event?

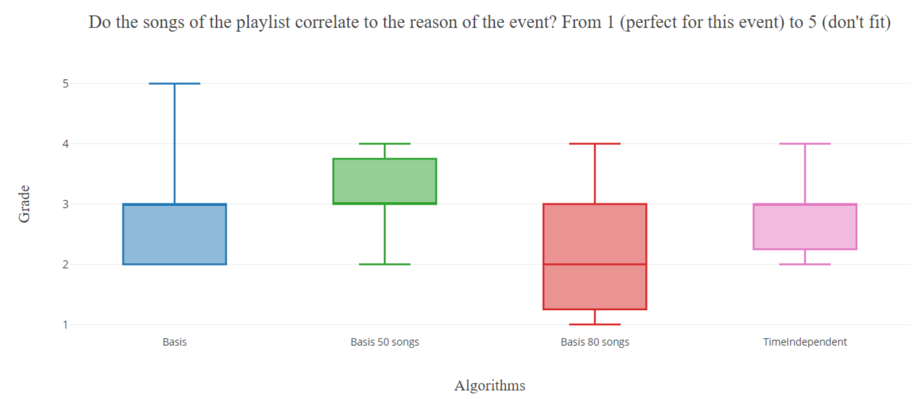


Figure 5.15.: Evaluation Result - Do the songs of the playlist correlate to the reason of the event?

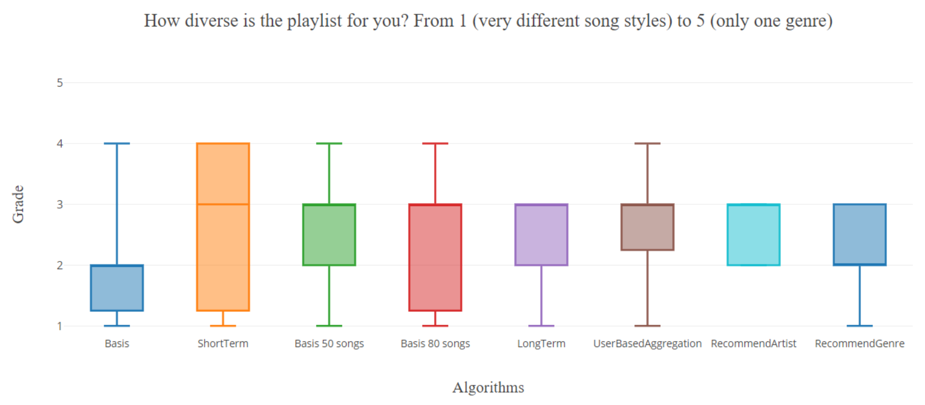


Figure 5.16.: Evaluation Result - How diverse is the playlist for you?

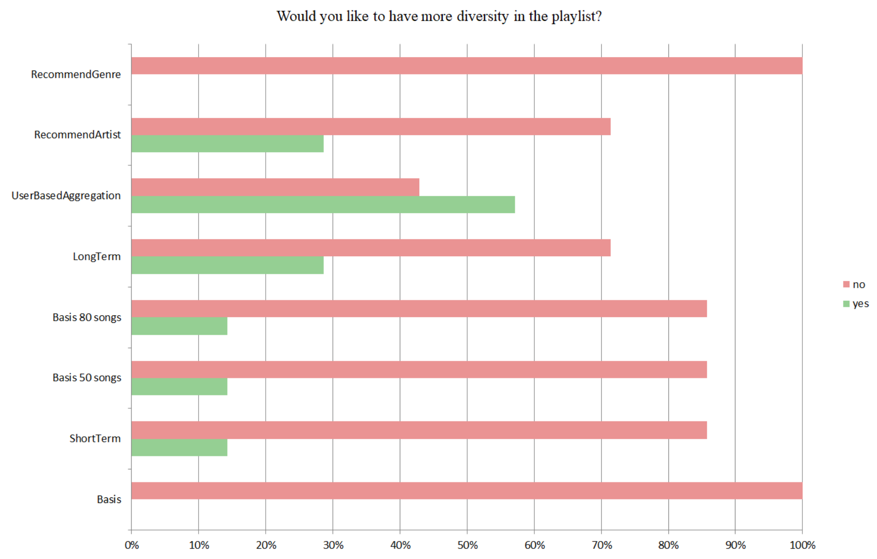


Figure 5.17.: Evaluation Result - Would you like to have more diversity in the playlist?

The result of the question *Would you like to have more diversity in the playlist?* is shown in Figure 5.17. Conforming with the previous question, all participants agree to have enough diversity in the basic algorithm. This is the majority opinion on nearly all other variations. The variation *UserBasedAggregation* is the only one which received the opposite results to the majority.

The result of the question *How bad is the worst song in the playlist?* is shown in Figure 5.18. In the variations *ShortTerm* and *Genre* the worst songs are not as bad as in the other variations.

The result of the question *Would you use this algorithm again?* is shown in Figure 5.19. The final question for each variation shows that the variations *ShortTerm*, *Valence*, and *Genre* convince the participants the most.

The result of the question *Are you used to listen to songs mostly in the order of the playlist, or in shuffle mode?* is shown in Figure 5.20. The result shows that nearly half of the participants listen to songs in shuffle mode most of the time.

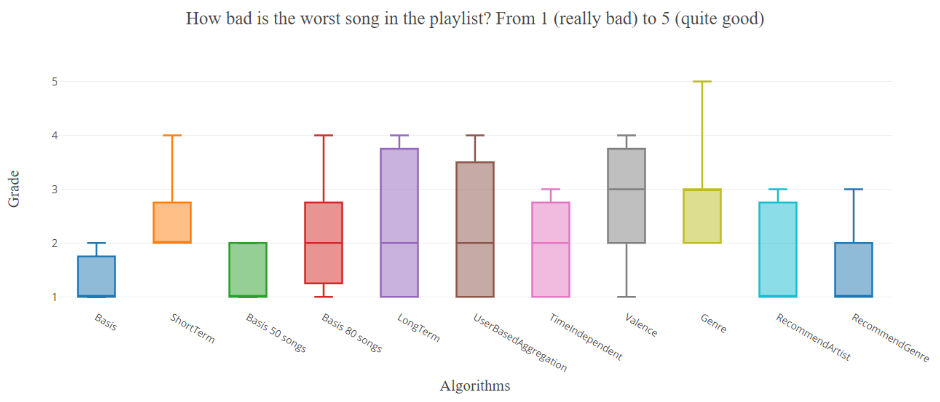


Figure 5.18.: Evaluation Result - How bad is the worst song in the playlist?

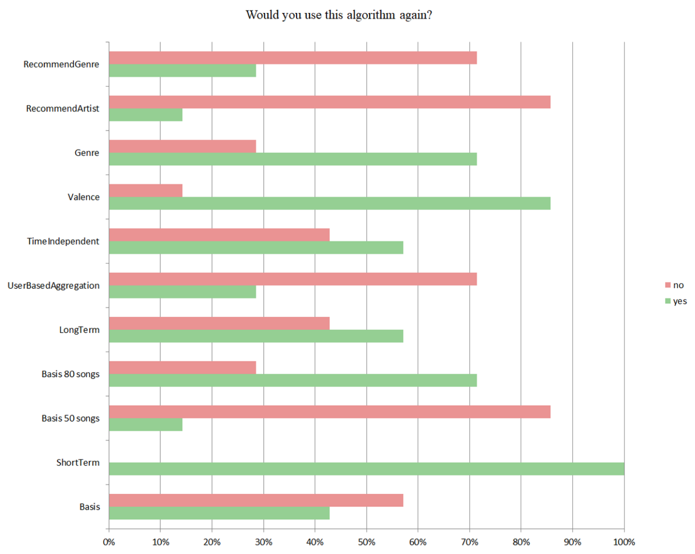


Figure 5.19.: Evaluation Result - Would you use this algorithm again?

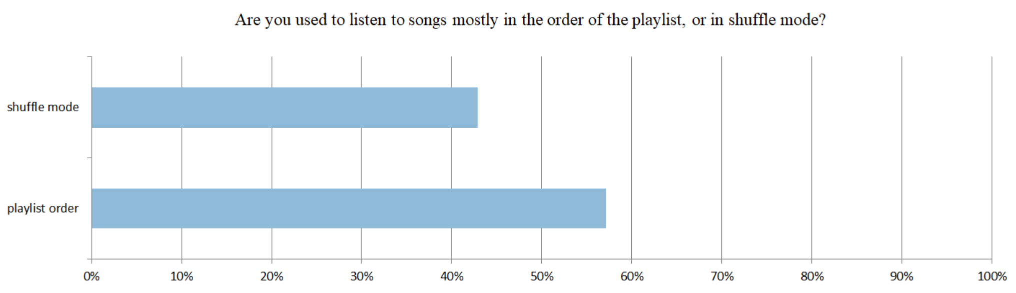


Figure 5.20.: Evaluation Result - Are you used to listen to songs mostly in the order of the playlist, or in shuffle mode?

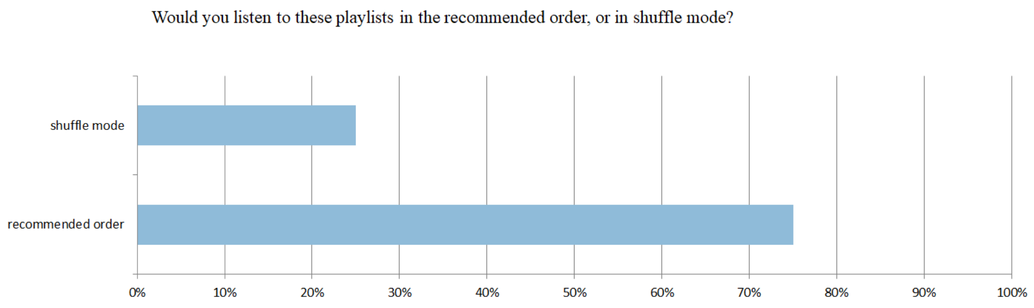


Figure 5.21.: Evaluation Result - Would you listen to these playlists in the recommended order or in shuffle mode?

The participants who answered the question *Are you used to listen to songs mostly in the order of the playlist, or in shuffle mode?* with *order of the playlist* were asked if they would listen to the given playlist in the recommended order or in shuffle mode. The result of the question is shown in Figure 5.21. The result shows that the approach to order the tracks is successful. More than 70 percent of the participants, who answered this question, see an improvement in the order.

6. Outlook

In this chapter, the first part includes a summary of the essential findings of this thesis. The second part includes a discussion of the future work which can be done to improve the recommendation of music to groups.

6.1. Conclusion

The aim of this thesis was to create a system which recommends music to groups. The system should be usable for many people. Further aims were to create a system which reaches a high value of accuracy without asking the users about their preferences, and the results should be easily playable without the need of music licenses for the system.

This section reflects the aims and summarizes the insights made during the process of creating this thesis.

6.1.1. Usable for a large Number of People

The system was created as an app for *iOS* and for *Android*, which cover now 100 percent of the global mobile OS market as shown in Figure 6.1. A constraint of the system is that it is usable only with Spotify[1]. This is caused by another aim what means to reach good recommendations without asking the user for the preferences. Nevertheless, *Spotify* is currently the biggest music streaming service by the amount of paid subscriptions as seen in Figure 4.23. Additionally, *Spotify* is usable without a paid subscription. This lead to a high number of people who can use the system.

[1]https://spotify.com/

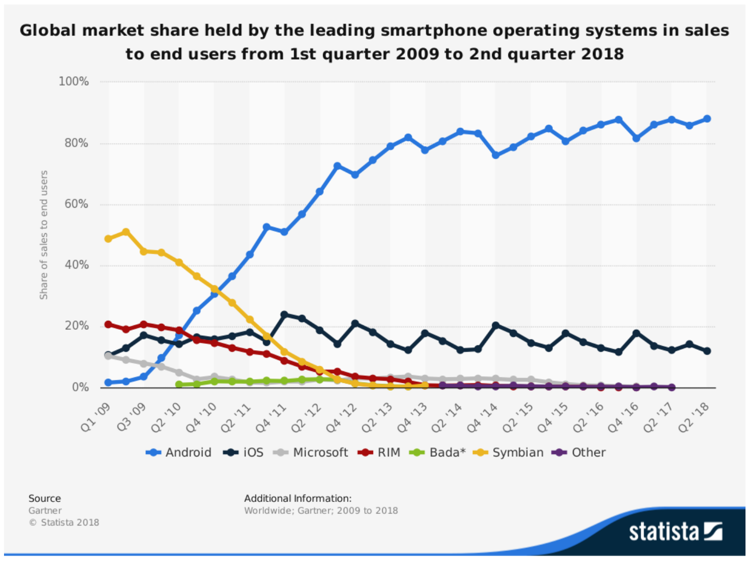


Figure 6.1.: Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018

6.1.2. Recommendation without Explicit User Feedback

An aim was to create a system without an annoying initial rating of several tracks up to the moment a recommendation can be created. Additionally, in GRSs all members of a group have to initially rate items. The solution was to find a service with a high number of users which has already a knowledge of the preferences of the users, and additionally, provide this information to external systems. Fortunately, the music streaming services with the highest amount of paid subscriptions provide an API called Spotify for Developers[2] to request the needed information over interfaces. This allows the system to get enough information about the preferences of users to create accurate recommendations.

[2]https://developer.spotify.com/

6.1.3. Create Accurate Group Recommendations

Creating good recommendations is the basic aim of a RS. The accuracy of the system was evaluated by the execution of a user study including a survey. There is a basic algorithm and different variations which follow different approaches. The original algorithm creates a user preference model based on the top tracks of a user. As an additional step, it calculates the top five genres of the whole group. In the next step, the algorithm creates a user individual candidate list based on the user's preferences assignable to the top genres. Out of the top tracks in combination with the candidate list, a user individual suggestion list is created. In this step, the group members have the possibility to manipulate this suggestion list. When the users finish this step, the algorithm creates a group preference model based on all manipulated suggestion lists. The songs with the shortest distance to the group preference model have been part of the final recommendation. An approach to avoid unfairness over time is additionally implemented.

The evaluation pointed out the successful variations which were finally integrated in the resulting algorithm. The successful approaches are the following.

- *Using short-term top tracks as a part of information for user preferences.* The evaluation shows increasing accuracy on results compared to medium-term top tracks.
- *Using long-term top tracks as a part of information for user preferences.* Additionally, short-term top tracks and long-term top tracks can improve the performance of the algorithm. This lead to a combined source for the preference model of a single user which now has a stable factor of user preference in addition to a fast-changing factor of the user preference based on the music the user has listened to the last four weeks.
- *Genre* The results of the variation show mostly a wide range of answers. Nevertheless, it reaches consistently positive answers in the question of using this variation again.
- *Valence.* This approach reached outstanding improvements in the accuracy of the RS. The thought was that groups want to listen to happy,

euphoric and cheerful songs rather than to sad and depressive songs. As the evaluation showed, the assumption was correct.

The evaluation shows that even the initial algorithm achieves quite good results. Different variations reach improvements on various properties of a RS. The combination of these will achieve very appropriate music recommendations for groups.

6.1.4. Easily Listenable Playlists

The last aim is an essential one. A system can make perfect accurate music recommendations for groups, but it is useless without the possibility to listen to these songs. The problem in such systems is the license to play music and the costs of dues when a user is listening to songs. The system of this thesis solves this, by exporting the playlist to the music streaming service Spotify[3] and not playing the songs itself. As every user of the system has to have an account on *Spotify*, the user can switch to the streaming service app and listen to the playlist immediately. Additionally, the system integrates an algorithm to order the recommended tracks for the playlist. The approach was to link similar songs based on the amount of similar classifiable songs and finally orders these linked sub-lists by the value of happiness of the individual songs.

6.1.5. Availability of the App M'Usic

The app *M'Usic* is currently only available for selected test-users in the Google Play Store[4] as well as in the *TestFlight*[5] app. This is a way to share a beta version of an app with test-users in *iOS*. The app is still in the beta stores because there has to be done an evaluation of the usability of the app, which is part of another thesis.

[3]https://spotify.com/
[4]https://play.google.com/store
[5]https://developer.apple.com/testflight/

6.2. Future Work

This section discusses the possible topics of future work in recommending music to groups.

6.2.1. Explicit User Information

A benefit of the algorithm presented in this thesis is that the RS achieves good results without any explicit user information. However, this also means that except the manipulation of the personal suggestion list, an explicit user feedback is not possible. A possible improvement of the algorithm would be to give users the possibility to tell the system their preferences. This could be in different situations. Moreover, the impacts of such feedback on the results have to be clarified. Furthermore, explicit user feedback can be a weakness in the term of manipulation. Possibilities to add user feedback are:

- *Ask the group to define their preferred Genres*. This would be a possibility to replace the top genres of the group with the ones from the selection. The problem is that the majority or a group leader define these genres which can result in decreasing the value of the preferences of some members.
- *Ask the group member about the preferred Genres for this Group*. Some users might have widespread musical preferences, but want to influence the group preferences only with a part of these genres. For example, if a user likes to listen to rock and classical music and knows that the other group members like to listen to rock, the user can prevent classical music from being included in the group recommendation as one of his/her preference.
- *Ask the user about the event type of the recommendation*. Group recommendations can be requested in different situations. For example for a party, a study group or a road trip. Depending on the event type, a user has different expectations about the resulting playlist. For study groups, calm songs are preferred whereas on a party, songs should be danceable. The challenge on defining event types is to define a model appropriate to the type.

- *Ask the group members about the accuracy of the recommended playlist*. This can be done in two different ways. Either ask the user about the overall accuracy or let the user rate individual songs. The results can be compared with the predicted accuracy. This is the final playlist compared to the group member preference model. If there are differences, the preference model of the user can be adjusted.

As mentioned before, the only feedback the app is getting from the users is the manipulation of the personal suggestion list. Currently the algorithm of the app is not using this information to adapt future preference models. This is because it has to be clarified which reason the deletion of a track or the adding of a song has. Reasons can be that a song like this one is not accurate for this special event, for the group, or the song is not accurate to the preferences of the user. Each of these reasons has to be handled differently by the algorithm.

Although explicit user feedback can improve the accuracy of the recommendation, users often are sensitive in giving a reply. Too many requests on feedback can have a negative affect. This means that user feedback has to be requested meticulously to reach the maximum information with a minimum number of questions.

6.2.2. Improved Preference Learning

Preference Learning can be a process over time to improve the calculation of a preference model. It handles user feedback to adopt the model. Techniques for model learning were described in Section 2.4.2. Especially the manipulation of the personal suggestion list and a feedback about the accuracy of the recommended playlist can be used to improve the preference learning. Additionally, in the app *M'Usic* it is possible to observe the exported playlist in Spotify[6] to check if the track lists were manipulated and let this affect the preference learning.

[6]https://spotify.com/

6.2.3. Time-Aware Recommendation

An unfortunately failed improvement of the algorithm was the approach of an impact on the recommendation based on the time. As the evaluation brings no clear result why the implementation failed, there could be made further researches. Especially in the field of music the inclusion of the time of the event can play a big role. Another approach is to adjust the preference model based on the type of the event. Both approaches have unfortunately the same challenge, to investigate how the time or type affects the preference model.

6.2.4. Collaborative Techniques

Another improvement would be the integration of collaborative techniques in the algorithm. As the data basis is stored in Spotify for Developers[7] and not accessible as needed, the collaborative approaches can only be made with the data the app has already requested and calculated. The similarity between group preference models can be used to improve the algorithm. In combination with explicit user feedback, groups can profit by ratings on recommending playlists of members of similar groups.

6.2.5. Long-Term Evaluation

An important task for the future is to evaluate the performance of the algorithm over time. Especially the impacts of long-term fairness have to be investigated. The focus of the evaluation in Chapter 5 was the comparison of different algorithm variations. The goal was to combine all approaches, which led to positive feedback, to an aggregated RS. Further evaluations should investigate the improvements made by combining these approaches, besides long-term fairness. Additionally, further evaluation should be done on the phenomenon of different results of short and long playlists compared to playlists with a medium number of tracks.

[7]https://developer.spotify.com/

Appendix

Appendix A.

Survey

Welcome to the survey about the performance of the algorithm to recommend music to groups. We would like to answer each question honestly and objective. The survey has opening questions, some questions for each variation of the algorithm, and a few closing questions.

This survey is part of the master's thesis of Johannes Singer

Section A: Initial questions

A1. **What is your name?**

A2. **What is your username or email address in M'Usic?**

A3. **In which group do you add your events to answer the questions of the survey?**

Section B: ShortTerm

B1. **What is the name of the playlist to answer the questions for this type of algorithm?**

B2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

B3. **How many songs of your personal recommendation fit to your taste for this event?**

- 0%-35% ☐
- 35%-55% ☐
- 55%-75% ☐
- 75%-90% ☐
- 90%-100% ☐

B4. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

- 1 ☐
- 2 ☐
- 3 ☐
- 4 ☐
- 5 ☐

B5. **Would you like to get more diversity in your personal recommendation?**

- Yes ☐
- No ☐

B6. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

- 1 ☐
- 2 ☐
- 3 ☐
- 4 ☐
- 5 ☐

B7. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

- Yes ☐
- No ☐

B8. **Did you get an unknown song which you really like in your personal recommendation?**

- Yes ☐
- No ☐

B9. **How many songs of your personal recommendation did you already know?**

- 0%-35% ☐
- 35%-55% ☐
- 55%-75% ☐
- 75%-90% ☐
- 90%-100% ☐

B10. **Would you like to get more unknown songs in your personal recommendation?**

- Yes ☐
- No ☐

B11. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

- 1 ☐
- 2 ☐
- 3 ☐
- 4 ☐
- 5 ☐

B12. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

B13. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

B14. **Would you like to have more diversity in the playlist?**

Yes	☐
No	☐

B15. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

B16. **Would you use this algorithm again?**

Yes	☐
No	☐

Section C: MediumTerm30

C1. **What is the name of the playlist to answer the questions for this type of algorithm?**

C2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

C3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

C4. **Do the songs of your personal recommendation correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

C5. **Do the songs of your personal recommendation correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

C6. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

C7. **Would you like to get more diversity in your personal recommendation?**

Yes ☐
No ☐

C8. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

C9. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes ☐
No ☐

C10. **Did you get an unknown song which you really like in your personal recommendation?**

Yes ☐
No ☐

C11. **How many songs of your personal recommendation did you already know?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

C12. **Would you like to get more unknown songs in your personal recommendation?**

Yes ☐
No ☐

C13. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

C14. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

C15. **Do the songs of the playlist correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1	☐
2	☐
3	☐
4	☐
5	☐

C16. **Do the songs of the playlist correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1	☐
2	☐
3	☐
4	☐
5	☐

C17. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

C18. **Would you like to have more diversity in the playlist?**

Yes	☐
No	☐

C19. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

C20. **Would you use this algorithm again?**

Yes	☐
No	☐

Section D: MediumTerm50

D1. What is the name of the playlist to answer the questions for this type of algorithm?

D2. How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D3. How many songs of your personal recommendation fit to your taste for this event?

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

D4. Do the songs of your personal recommendation correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D5. Do the songs of your personal recommendation correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D6. How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D7. Would you like to get more diversity in your personal recommendation?

Yes ☐
No ☐

D8. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D9. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes ☐
No ☐

D10. **Did you get an unknown song which you really like in your personal recommendation?**

Yes ☐
No ☐

D11. **How many songs of your personal recommendation did you already know?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

D12. **Would you like to get more unknown songs in your personal recommendation?**

Yes ☐
No ☐

D13. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D14. **How many songs of the playlist do you like?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

D15. **Do the songs of the playlist correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

D16. **Do the songs of the playlist correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1	☐
2	☐
3	☐
4	☐
5	☐

D17. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

D18. **Would you like to have more diversity in the playlist?**

Yes	☐
No	☐

D19. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

D20. **Would you use this algorithm again?**

Yes	☐
No	☐

Section E: MediumTerm80

E1. **What is the name of the playlist to answer the questions for this type of algorithm?**

E2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

E3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

E4. **Do the songs of your personal recommendation correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1	☐
2	☐
3	☐
4	☐
5	☐

E5. **Do the songs of your personal recommendation correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1	☐
2	☐
3	☐
4	☐
5	☐

E6. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

E7. **Would you like to get more diversity in your personal recommendation?**

Yes	☐
No	☐

E8. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

E9. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes	☐
No	☐

E10. **Did you get an unknown song which you really like in your personal recommendation?**

Yes	☐
No	☐

E11. **How many songs of your personal recommendation did you already know?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

E12. **Would you like to get more unknown songs in your personal recommendation?**

Yes	☐
No	☐

E13. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

E14. **How many songs of the playlist do you like?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

E15. **Do the songs of the playlist correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

E16. **Do the songs of the playlist correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

E17. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

E18. **Would you like to have more diversity in the playlist?**

Yes ☐
No ☐

E19. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

E20. **Would you use this algorithm again?**

Yes ☐
No ☐

E21. **Is one of the playlists of this algorithm much worse than the others?**

Yes ☐
No ☐

Section F: LongTerm

F1. **What is the name of the playlist to answer the questions for this type of algorithm?**

F2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

F3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

F4. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

F5. **Would you like to get more diversity in your personal recommendation?**

Yes ☐
No ☐

F6. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

F7. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes ☐
No ☐

F8. **Did you get an unknown song which you really like in your personal recommendation?**

Yes ☐
No ☐

F9. **How many songs of your personal recommendation did you already know?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

F10. **Would you like to get more unknown songs in your personal recommendation?**

Yes	☐
No	☐

F11. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

F12. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

F13. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

F14. **Would you like to have more diversity in the playlist?**

Yes	☐
No	☐

F15. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

F16. **Would you use this algorithm again?**

Yes	☐
No	☐

Section G: UserBasedAggregation

G1. What is the name of the playlist to answer the questions for this type of algorithm?

G2. How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

G3. How much do you like the playlist? From 1 (very much) to 5 (not at all).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

G4. How many songs of the playlist do you like?

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

G5. How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

G6. Would you like to have more diversity in the playlist?

Yes ☐
No ☐

G7. How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

G8. Would you use this algorithm again?

Yes ☐
No ☐

Section H: TimeIndependentRecommendation

H1. **What is the name of the playlist to answer the questions for this type of algorithm?**

H2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H3. **Do the songs of your personal recommendation correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H4. **Do the songs of your personal recommendation correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H5. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H6. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H7. **Do the songs of the playlist correlate to the time of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H8. **Do the songs of the playlist correlate to the reason of the event? From 1 (perfect for this event) to 5 (don't fit).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H9. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

H10. **Would you use this algorithm again?**

Yes ☐
No ☐

Section I: Valence

I1. **What is the name of the playlist to answer the questions for this type of algorithm?**

I2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

I3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35% ☐
35%-55% ☐
55%-75% ☐
75%-90% ☐
90%-100% ☐

I4. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

I5. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

I6. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

I7. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

I8. **Would you use this algorithm again?**

Yes	☐
No	☐

Section J: Genre

J1. **What is the name of the playlist to answer the questions for this type of algorithm?**

J2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

J3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

J4. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

J5. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

J6. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

J7. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

J8. **Would you use this algorithm again?**

Yes	☐
No	☐

Section K: RecommendArtist

K1. **What is the name of the playlist to answer the questions for this type of algorithm?**

K2. **How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

K3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

K4. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

K5. **Would you like to get more diversity in your personal recommendation?**

Yes	☐
No	☐

K6. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

K7. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes	☐
No	☐

K8. **Did you get an unknown song which you really like in your personal recommendation?**

Yes	☐
No	☐

K9. **How many songs of your personal recommendation did you already know?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

K10. **Would you like to get more unknown songs in your personal recommendation?**

Yes	☐
No	☐

K11. How much do you like the playlist? From 1 (very much) to 5 (not at all).

1	☐
2	☐
3	☐
4	☐
5	☐

K12. How many songs of the playlist do you like?

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

K13. How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).

1	☐
2	☐
3	☐
4	☐
5	☐

K14. Would you like to have more diversity in the playlist?

Yes	☐
No	☐

K15. How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).

1	☐
2	☐
3	☐
4	☐
5	☐

K16. Would you use this algorithm again?

Yes	☐
No	☐

Section L: RecommendGenre

L1. What is the name of the playlist to answer the questions for this type of algorithm?

L2. How much do you like your personal recommendation? From 1 (very much) to 5 (not at all).

1	☐
2	☐
3	☐
4	☐
5	☐

L3. **How many songs of your personal recommendation fit to your taste for this event?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

L4. **How diverse is your personal recommendation? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

L5. **Would you like to get more diversity in your personal recommendation?**

Yes	☐
No	☐

L6. **How bad is the worst song in your personal recommendation? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

L7. **Did you get an unknown song from an unknown interpret which you like in your personal recommendation?**

Yes	☐
No	☐

L8. **Did you get an unknown song which you really like in your personal recommendation?**

Yes	☐
No	☐

L9. **How many songs of your personal recommendation did you already know?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

L10. **Would you like to get more unknown songs in your personal recommendation?**

Yes	☐
No	☐

L11. **How much do you like the playlist? From 1 (very much) to 5 (not at all).**

1	☐
2	☐
3	☐
4	☐
5	☐

L12. **How many songs of the playlist do you like?**

0%-35%	☐
35%-55%	☐
55%-75%	☐
75%-90%	☐
90%-100%	☐

L13. **How diverse is the playlist for you? From 1 (very different song styles) to 5 (only one genre).**

1	☐
2	☐
3	☐
4	☐
5	☐

L14. **Would you like to have more diversity in the playlist?**

Yes	☐
No	☐

L15. **How bad is the worst song in the playlist? From 1 (really bad) to 5 (quite good).**

1	☐
2	☐
3	☐
4	☐
5	☐

L16. **Would you use this algorithm again?**

Yes	☐
No	☐

Section M: Closing questions

M1. **Are you used to listen to songs mostly in the order of the playlist, or in shuffle mode?**

playlist order	☐
shuffle mode	☐

M2. **Please do have a look at the order of the songs in the calculated playlists - Would you listen to these playlists in the recommended order, or in shuffle mode?**

recommended order	☐
shuffle mode	☐

Thank you for taking part in the survey. This helped us to improve our algorithm for better user-experience.

Bibliography

Adomavicius, G. and A. Tuzhilin (June 2005). "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions." In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99 (cit. on p. 9).

Adomavicius, G. and A. Tuzhilin (2015). English. In: Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Second edition. Boston, MA: Springer Verlag. Chap. Context-Aware Recommender Systems, pp. 191–226. ISBN: 9781489976369 (cit. on pp. 10, 12).

Adomavicius, Gediminas, Bamshad Mobasher, et al. (2011). "Context-aware recommender systems." English. In: *AI Magazine* 32.3, pp. 67–80 (cit. on pp. 10, 11).

Adomavicius, Gediminas, Ramesh Sankaranarayanan, et al. (2005). "Incorporating contextual information in recommender systems using a multidimensional approach." English. In: *ACM Transactions on Information Systems (TOIS)* 23.1, pp. 103–145 (cit. on p. 5).

Amatriain, X. and J.M. Pujol (2015). English. In: Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Second edition. Boston, MA: Springer Verlag. Chap. Data Mining Methods for Recommender Systems, pp. 227–262. ISBN: 9781489976369 (cit. on pp. 12–14, 18).

Ankolekar, Anupriya and Thomas Sandholm (2011). "Foxtrot: A Soundtrack for Where You Are." In: *Proceedings of Interacting with Sound Workshop: Exploring Context-Aware, Local and Social Audio Applications*. IwS '11. Stockholm, Sweden: ACM, pp. 26–31. ISBN: 978-1-4503-0883-0. DOI: 10.1145/2019335.2019341. URL: http://doi.acm.org/10.1145/2019335.2019341 (cit. on p. 23).

Baltrunas, Linas et al. (2011). "Incarmusic: Context-aware music recommendations in a car." In: *International Conference on Electronic Commerce and Web Technologies*. Springer, pp. 89–100 (cit. on p. 22).

Billsus, Daniel and Michael Pazzani (1996). “Revising user profiles: The search for interesting web sites.” In: *Proceedings of the Third International Workshop on Multistrategy Learning*. AAAI, pp. 232–243 (cit. on p. 17).

Bogdanov, D., M. Haro, and F. Fuhrmann (June 2011). “A content-based system for music recommendation and visualization of user preferences working on semantic notions.” In: *2011 9th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pp. 249–252. DOI: 10.1109/CBMI.2011.5972554 (cit. on pp. 20, 21).

Bogdanov, Dmitry et al. (2013). “Semantic audio content-based music recommendation and visualization based on user preference examples.” In: *Information Processing & Management* 49.1, pp. 13–33. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2012.06.004. URL: http://www.sciencedirect.com/science/article/pii/S0306457312000763 (cit. on p. 21).

Bonnin, Geoffray and Dietmar Jannach (2013). “Evaluating the quality of playlists based on hand-crafted samples.” In: *Proc. ISMIR*, pp. 263–268 (cit. on p. 24).

Burke, Robin (2002). “Hybrid recommender systems: Survey and experiments.” English. In: *User Modelling and User-Adapted Interaction* 12.4, pp. 331–370 (cit. on p. 24).

Cai, Rui et al. (2007). “MusicSense: Contextual music recommendation using emotional allocation modeling.” English. In: *Proceedings of the 15th ACM international conference on Multimedia*. ACM, pp. 553–556. ISBN: 9781595937025 (cit. on p. 22).

Cai, Y. et al. (Mar. 2014). “Typicality-Based Collaborative Filtering Recommendation.” In: *IEEE Transactions on Knowledge and Data Engineering* 26.3, pp. 766–779. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.7 (cit. on p. 6).

Chao, Dennis, Justin Balthrop, and Stephanie Forrest (2005). “Adaptive radio: achieving consensus using negative preferences.” English. In: *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*. ACM, pp. 120–123. ISBN: 1595932232 (cit. on p. 47).

Christensen, Ingrid A. and Silvia Schiaffino (2011). “Entertainment recommender systems for group of users.” English. In: *Expert Systems With Applications* 38.11, pp. 14127–14135 (cit. on p. 48).

Cover, T. and P. Hart (1967). “Nearest neighbor pattern classification.” English. In: *IEEE Transactions on Information Theory* 13.1, pp. 21–27 (cit. on p. 17).

Cristianini, Nello, John Shawe-Taylor, et al. (2000). *An introduction to support vector machines: and other kernel-based learning methods*. English. Reprint. (with corr.) Cambridge: Cambridge Univ. Pr. ISBN: 0521780195 (cit. on p. 18).

Crossen, Andrew, Jay Budzik, and Kristian Hammond (2002). "Flytrap: intelligent group music recommendation." English. In: *Proceedings of the 7th international conference on Intelligent user interfaces*. ACM, pp. 184–185. ISBN: 9781581134599 (cit. on p. 46).

Deshpande, Mukund and George Karypis (Jan. 2004). "Item-based top-N Recommendation Algorithms." In: *ACM Trans. Inf. Syst.* 22.1, pp. 143–177. ISSN: 1046-8188. DOI: 10.1145/963770.963776. URL: http://doi.acm.org/10.1145/963770.963776 (cit. on p. 8).

Felfernig, Alexander, Michael Jeran, et al. (2014). "Basic approaches in recommendation systems." English. In: *Recommendation Systems in Software Engineering*. Springer, pp. 15–37. ISBN: 9783642451355 (cit. on pp. 6–8).

Felfernig, Alexander, Martin Stettinger, et al. (2018). *Group recommender systems: an Introduction*. English. ISBN: 3319750666 (cit. on pp. 1, 23, 32, 34–42).

Ge, Mouzhi, Carla Delgado-Battenfeld, and Dietmar Jannach (2010). "Beyond accuracy: evaluating recommender systems by coverage and serendipity." English. In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM, pp. 257–260. ISBN: 1605589063 (cit. on p. 39).

Gedikli, Fatih, Dietmar Jannach, and Mouzhi Ge (2014). "How should I explain? A comparison of different explanation types for recommender systems." English. In: *International Journal of Human - Computer Studies* 72.4, pp. 367–382 (cit. on p. 40).

Gemmis, Marco De et al. (2009). "Preference learning in recommender systems." In: *In Preference Learning (PL-09) ECML/PKDD-09 Workshop* (cit. on pp. 12, 13, 16).

Goldberg, David et al. (1992). *Using collaborative filtering to weave an information tapestry*. English (cit. on p. 5).

Gunawardana, Asela and Guy Shani (2015). English. In: Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Second edition. Boston, MA: Springer Verlag. Chap. Evaluating Recommender Systems, pp. 265–308. ISBN: 9781489976369 (cit. on pp. 25–32).

Han, Byeong-jun et al. (May 2010). "Music emotion classification and context-based music recommendation." In: *Multimedia Tools and Applications* 47.3, pp. 433–460. ISSN: 1573-7721. DOI: 10.1007/s11042-009-0332-6. URL: https://doi.org/10.1007/s11042-009-0332-6 (cit. on p. 22).

Herlocker, Jonathan et al. (2004). "Evaluating collaborative filtering recommender systems." English. In: *ACM Transactions on Information Systems (TOIS)* 22.1, pp. 5–53 (cit. on pp. 18, 19).

Jameson, Anthony and Barry Smyth (2007). "Recommendation to groups." In: *The adaptive web*. Springer, pp. 596–627 (cit. on pp. 33–35).

Jannach, Dietmar (2011). *Recommender systems: an introduction*. English. 1. publ. New York, NY [u.a.]: Cambridge Univ. Press. ISBN: 0521493366 (cit. on pp. 1, 6–9, 17).

Kilmann, Ralph H. and Kenneth W. Thomas (1977). "Developing a forced-choice measure of conflict-handling behavior: The "Mode" Instrument." English. In: *Educational and Psychological Measurement* 37.2, pp. 309–325 (cit. on p. 41).

Knees, Peter and Markus Schedl (2011). "Towards semantic music information extraction from the Web using rule patterns and supervised learning." English. In: *Workshop on Music Recommendation and Discovery*. Vol. 793, pp. 18–25. ISBN: 1613-0073 (cit. on p. 20).

Koren, Y., R. Bell, and C. Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems." English. In: *Computer* 42.8, pp. 30–37 (cit. on pp. 14–16).

Koren, Yehuda (2008). "Factorization meets the neighborhood: a multifaceted collaborative filtering model." English. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 426–434. ISBN: 1605581933 (cit. on p. 14).

Kouki, Pigi et al. (2017). "User Preferences for Hybrid Explanations." English. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, pp. 84–88. ISBN: 1450346529 (cit. on p. 41).

Kruskal, Joseph B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem." English. In: *Proceedings of the American Mathematical Society* 7.1, pp. 48–50 (cit. on p. 62).

Masthoff, Judith and Albert Gatt (2006). "In pursuit of satisfaction and the prevention of embarrassment: Affective state in group recommender systems." English. In: *User Modeling and User-Adapted Interaction* 16.3-4, pp. 281–319 (cit. on pp. 42, 43).

Maystre, Lucas (2012). *Music Recommendation to Groups*. Tech. rep. (cit. on p. 48).

McCarthy, Joseph and Theodore Anagnost (1998). “MusicFX: an arbiter of group preferences for computer supported collaborative workouts.” English. In: *AAAI Spring Symposium on Intelligent Environments*. ACM, pp. 363–372. ISBN: 1581130090 (cit. on p. 46).

McFee, B., L. Barrington, and G. Lanckriet (Oct. 2012). “Learning Content Similarity for Music Recommendation.” In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.8, pp. 2207–2218. ISSN: 1558-7916. DOI: 10.1109/TASL.2012.2199109 (cit. on pp. 20, 22).

O’Hara, Kenton et al. (2004). “Jukola: democratic music choice in a public space.” English. In: *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*. ACM, pp. 145–154. ISBN: 1581137877 (cit. on p. 46).

O’connor, Mark et al. (2001). “PolyLens: a recommender system for groups of users.” In: *ECSCW 2001*. Springer, pp. 199–218 (cit. on p. 39).

Ortega, Fernando et al. (2016). “Recommending items to group of users using Matrix Factorization based Collaborative Filtering.” English. In: *Information Sciences* 345, pp. 313–324 (cit. on p. 38).

Pazzani, Michael and Daniel Billsus (1997). “Learning and Revising User Profiles: The Identification of Interesting Web Sites.” English. In: *Machine Learning* 27.3, pp. 313–331 (cit. on p. 5).

Peppers, Jonathan, George Taskos, and Can Bilgin (2016). *Xamarin: Cross-Platform Mobile Application Development*. English. 1st ed. Birmingham, England: Packt Publishing. ISBN: 9781787120129 (cit. on pp. 86, 87, 89).

Quinlan, J. R. (1986). “Induction of decision trees.” English. In: *Machine Learning* 1.1, pp. 81–106 (cit. on p. 17).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2015). *Recommender systems handbook*. English. Second edition. Boston, MA: Springer Verlag. ISBN: 9781489976369 (cit. on p. 1).

Schedl, A. et al. (2015). English. In: Ricci, Francesco, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Second edition. Boston, MA: Springer Verlag. Chap. Music Recommender Systems, pp. 453–492. ISBN: 9781489976369 (cit. on pp. 19, 22, 23).

Schedl, Markus (2013). “Ameliorating Music Recommendation: Integrating Music Content, Music Context, and User Context for Improved Music Retrieval and Recommendation.” In: *Proceedings of International Conference on Advances in Mobile Computing &#38; Multimedia*. MoMM ’13. Vienna, Austria: ACM, 3:3–3:9. ISBN: 978-1-4503-2106-8. DOI: 10.1145/2536853.

2536856. URL: http://doi.acm.org/10.1145/2536853.2536856 (cit. on p. 23).

Schedl, Markus, Arthur Flexer, and Julián Urbano (Dec. 2013). "The neglected user in music information retrieval research." In: *Journal of Intelligent Information Systems* 41.3, pp. 523–539. ISSN: 1573-7675. DOI: 10.1007/s10844-013-0247-6. URL: https://doi.org/10.1007/s10844-013-0247-6 (cit. on p. 24).

Soleymani, M. et al. (June 2015). "Content-based music recommendation using underlying music preference structure." In: *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6. DOI: 10.1109/ICME.2015.7177504 (cit. on p. 20).

Son, Jieun and Seoung Bum Kim (2017). "Content-based filtering for recommendation systems using multiattribute networks." In: *Expert Systems with Applications* 89, pp. 404–412. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.08.008. URL: http://www.sciencedirect.com/science/article/pii/S0957417417305468 (cit. on p. 9).

Sprague, David, Fuqu Wu, and Melanie Tory (2008). "Music selection using the PartyVote democratic jukebox." English. In: *Proceedings of the working conference on Advanced visual interfaces*. ACM, pp. 433–436. ISBN: 9781605581415 (cit. on p. 47).

Thayer, Robert E (1990). *The biopsychology of mood and arousal*. Oxford University Press (cit. on p. 23).

Versluis, Gerald (2017). *Xamarin Continuous Integration and Delivery : Team Services, Test Cloud, and HockeyApp*. English. 1st ed. Berkeley, CA: Apress L. P. ISBN: 9781484227152 (cit. on pp. 98, 101).

Xue, Gui-Rong et al. (2005). "Scalable collaborative filtering using cluster-based smoothing." English. In: *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 114–121. ISBN: 9781595930347 (cit. on p. 18).

Yu, Zhiwen, Xingshe Zhou, and Daqing Zhang (2005). "An adaptive in-vehicle multimedia recommender for group users." English. In: *2005 IEEE 61st Vehicular technology conference*. Vol. 61. IEEE. Chap. 5, pp. 2800–2804. ISBN: 1550-2252 (cit. on p. 47).