Chapter 1
Music Recommendation

Dietmar Jannach and Geoffray Bonnin

Abstract This is a preliminary version of Chapter 26 of "Music Data Music Data Analysis: Foundations and Applications" by (Claus Weihs, Dietmar Jannach, Igor Vatolkin, Guenter Rudolph, eds.), CRC Press, 2016

1.1 Introduction

Until recently, music discovery was a difficult task. We had to listen to the radio hoping one track will be interesting, actively browse the repertoire of a given artist, or randomly try some new artists from time to time. With the emergence of personalized recommendation systems, we can now discover music just by letting music platforms play tracks for us. In another scenario, when we wanted to prepare some music for a particular event, we had to carefully browse our music collection and spend significant amounts of time selecting the right tracks. Today, it has become possible to simply specify some desired criteria like the genre or mood and an automated system will propose a set of suitable tracks.

Music recommendation is however a very challenging task, and the quality of the current recommendations is still not always satisfying. First, the size of the pool of tracks from which to make the recommendations can be quite huge. For instance, Spotify[1], Groove[2], Tidal[3] and Qobuz[4], four of the currently most successful web music platforms, all contain more than 30 millions tracks[5]. Moreover, most of the

Department of Computer Science, TU Dortmund, Germany, e-mail: firstname.lastname@tu-dortmund.de

[1] http://www.spotify.com

[2] http://music.microsoft.com

[3] http://tidal.com

[4] http://www.qobuz.com

[5] This information can be obtained using the API's search services provided by these platforms.

tracks on these platforms typically have a low popularity[6] and hence few information is available about them, which makes them even harder to process for the task of automated recommendation. Another difficulty is that the recommended tracks are immediately consumed, which means the recommendations must be made very fast, and have at the same time to fit the current context.

Music recommendation was one early application domain for recommendation techniques, starting with the *Ringo* system presented in 1995 [34]. Since then however, most of the research literature on recommender systems (RS) has dealt with the recommendation of movies and commercial products [17]. Although the corresponding core strategies can be applied to music, music has a set of specificities which can make these strategies insufficient.

In this chapter, we will discuss today's most common methods and techniques for item recommendation which were developed mostly for movies and in the e-commerce domain, and talk about particular aspects of the recommendation of music. We will then show how we can measure the quality of recommendations and finally give examples of real-world music recommender systems. Parts of our discussion will be based on [5], [8] and [22], which represent recent overviews on music recommendation and playlist generation.

1.2 Common Recommendation Techniques

Generally speaking, the task of a recommender system in most application scenarios is to generate a ranked list of items which are assumedly relevant or interesting for the user in the current context[7]. Recommendation algorithms are usually classified according to the types of data and knowledge they process to determine these ranked lists. In the following, we will introduce two common recommendation strategies found in the literature.

1.2.1 Collaborative Filtering

The most prominent class of recommendation algorithms in research and maybe also in industry is called *Collaborative Filtering* (CF). In such systems, the only type of data processed by the system to compute recommendation lists are *ratings* provided by a larger user community. Table 1.1 shows an example for such a rating database, where 5 users have rated 5 songs using a rating scale from 1 (lowest) to 5 (highest), e.g., on an online music platform or using their favorite music player.

[6] This information can be obtained using for instance the API of last.fm or The Echo Nest, two of the currently richest sources of track information on the Web.

[7] In Section 1.4 we will discuss in more detail what relevance or interestingness could mean for the user.

In this simple example the task of the recommender is to decide whether or not the *Song5* should be put in Alice's recommendation list and – if there are also other items – at which position it should appear in the list.

Table 1.1: A Simple Rating Database, Adapted from [20]. When Recommendation is Considered as a *Rating Prediction* Problem, the Goal is to Estimate the Missing Values in the Rating "Matrix" (Marked with '**?**')

	Song1	Song2	Song3	Song4	Song5
Alice	5	3	4	4	**?**
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Many CF systems approach this problem by first *predicting* Alice's rating for all songs which she has not seen before. In the second step, the items are ranked according to the prediction value, where the songs with the highest predictions should obviously appear on top of the list[8].

1.2.1.1 CF Algorithms

One of the earliest and still relatively accurate schemes to predict Alice's missing ratings is to base the prediction on the opinion of other users, who have liked similar items as Alice in the past, i.e., who have the same taste. The users of this group are usually called "neighbors" or "peers". When using such a scheme, the question is (a) how to measure the similarity between users and (b) how to aggregate the opinions of the neighbors. In one of the early papers on RS [31], the following approach was proposed, which is still used as a baseline for comparative evaluation today. To determine the similarity, the use of Pearson's correlation coefficient was advocated. The similarity of users u_1 and u_2 can thus be calculated via

$$sim(u_1,u_2) = \frac{\sum_{i\in\widehat{I}}(r_{u_1,i}-\overline{r_{u_1}})(r_{u_2,i}-\overline{r_{u_2}})}{\sqrt{\sum_{i\in\widehat{I}}(r_{u_1,i}-\overline{r_{u_1}})^2}\sqrt{\sum_{i\in\widehat{I}}(r_{u_2,i}-\overline{r_{u_2}})^2}} \tag{1.1}$$

where $\widehat{I}$ denotes the set of products, which have been rated both by user u_1 and user u_2, $\overline{r_{u_1}}$ is u_1's average rating and $r_{u_1,i}$ denotes u_1's rating for item i.

Beside using Pearson's correlation, also other metrics such as cosine similarity have been proposed. One of the advantages of Pearson's correlation however is that it takes into account the tendencies of individual users to give mostly low or high ratings.

[8] Taking the general popularity of items into account in the ranking process is however also common in practical settings because of the risk that only niche items are recommended.

Once the similarity of users is determined, the remaining problem is to predict Alice's missing ratings. Given a user u_1 and an unseen item i, we could for example compute the prediction based on u_1's average rating and the opinion of a set of N closest neighbors as follows:

$$\hat{r}(u_1,i) = \overline{r_{u_1}} + \frac{\sum_{u_2 \in N}(sim(u_1,u_2)(r_{u_2,i} - \overline{r_{u_2}}))}{\sum_{u_2 \in N} sim(u_1,u_2)}. \tag{1.2}$$

The prediction function in Equation (1.2) uses the user's average rating $\overline{r_{u_1}}$ as a baseline. For each neighbor u_2 we then determine the difference between u_2's average rating and his rating for the item in question, i.e., $(r_{u_2,i} - \overline{r_{u_2}})$ and weight the difference with the similarity factor $(sim(u_1,u_2))$ computed using Equation (1.1).

When we apply these calculations to the example in Table 1.1, we can identify *User2* and *User3* as the closest neighbors to Alice ($sim(User2, User3)$ is 0.85 and 0.7). Both have rated *Song5* above their average and predict an above-average rating between 4 and 5 (exactly 4.87) for Alice, which means that we should include the song in a recommendation list.

While the presented scheme is quite accurate – we will see later on how to measure accuracy – and simple to implement, it has the disadvantage of being basically not applicable for real-world problems due to its limited scalability, since there are millions of songs and millions of users for which we would have to calculate the similarity values.

Therefore a large variety of alternative methods have been proposed over the last decades to predict the missing ratings. Nearly all of these more recent methods are based on offline data preprocessing and on what is called "model-building". In such approaches, the system learns a usually comparably compact model in an offline and sometimes computationally intensive training phase. At run-time, the individual predictions for a user can however be calculated very fast. Depending on the application domain and the frequency of newly arriving data, the model is then re-trained periodically. Among the applied methods we find various data mining techniques such as association rule mining or clustering, support vector machines, regression methods and a variety of probabilistic approaches. In recent years, several methods were designed which are based on *matrix factorization* (MF) as well as ensemble methods which combine the results of different learning methods [23].

In general, the ratings that the users assigned to items can be represented as a matrix, and this matrix can be factorized, i.e., it is possible to write this matrix R as the product of two other matrices Q and P:

$$R = Q^T \cdot P$$

Matrix Factorization techniques determine approximations of Q and P using different optimization procedures. Implicitly these methods thereby map users and items to a shared factor space of a given size (dimensionality) and use the inner product of the resulting matrices to estimate the relationship between users and items [23]. Using such factorizations makes the computation times much shorter and at the same time implicitly reveals some *latent* factors. A latent aspect of a song

could be the artist or the musical genre the song belongs to; in general, however, the semantic meanings of the factors are unknown. After the factorization process with f latent factors (for example $f = 100$), we are given a vector $\boldsymbol{q}_i \in \mathbb{R}^f$ for each item i and a vector $\boldsymbol{p}_u \in \mathbb{R}^f$ for each user. For the user vectors, each value of the vector corresponds to the interest of a user in a certain factor; for item vectors, each element indicates the degree of "fit" of the item to the factor. Given a user u and an item i, we can finally estimate the "match" between the user and the item by using the dot product $\boldsymbol{q}_i^T \boldsymbol{p}_u$.

Different heuristic strategies exist for determining the values for the latent factor vectors $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$. The most common ones in RS, which also scale to larger scale rating data bases, are stochastic gradient descent optimization and *Alternating Least Squares* [23].

In order to estimate a rating $\hat{r}_{u,i}$ for user u and item i, we can use the following general equation, where μ is the global rating average, b_i is the item bias and b_u is the user bias.

$$\hat{r}_{u,i} = \mu + b_i + b_u + \boldsymbol{q}_i^T \boldsymbol{p}_u \tag{1.3}$$

The reason for modeling user and item biases is that there are items which are generally more liked or disliked than others, and there are, on the other hand, users who generally give higher or lower ratings than others. For instance, Equation (1.3) with $f = 2$ corresponds to the assumption that only two factors are sufficient to accurately estimate the ratings of users. These factors may be, for instance, the genre and the tempo of tracks, or any other factors, which are inferred during the factorization step.

The learning phase of such an algorithm consists in estimating the unknown parameters based on the data. This can be achieved by searching for parameters which minimize the squared prediction error, given the set of known ratings K:

$$\min_{q*,p*,b*} \sum_{(u,i)\in K} \left(r_{u,i} - (\mu + b_u + b_i + q_i^T p_u)\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2\right) \tag{1.4}$$

The last term in the function is used for regularization and to "penalize" large parameter values.

Overall, in the past years much research in the field of recommender systems was devoted to such rating prediction algorithms. It however becomes more and more evident that rating prediction is very seldom the goal in practical applications. Finding a good ranking of the tracks based on observed user behavior is more relevant, which led to an increased application of "learning to rank" methods for this task or to the development of techniques that optimize the order of the recommendations according to music-related criteria such as track transitions or the coherence of the playlists [19].

1.2.1.2 Collaborative Filtering for Music Recommendation

As mentioned in the introduction, although music was one of the earliest application domains for recommender systems, music recommendation has until recently remained a niche topic. The first application, the *Ringo* system, actually used a CF technique [34], and modern online music services such as *Spotify*, *last.fm* or *iTunes Genius* use – among other techniques – collaborative filtering methods to generate playlists and recommend songs.

When compared to other approaches to (music) recommendation, collaborative filtering methods have some well-known advantages and limitations. From a system provider's perspective, one advantage of CF lies in the fact that beside the users' rating feedback no additional information (about the musical genre, the authors or any sort of low-level data) has to be acquired and maintained. At the same time, CF methods are well understood and have been successfully applied in a variety of domains, including those where massive amounts of data have to be processed and a large number of parallel users have to be served. The inherent characteristic of CF-based algorithms in addition can lead to recommendations that are surprising and novel for the user, which can be a key feature for a music recommender in particular when the user is interested in discovering new artists or musical sub-genres.

On the down side, CF methods require the existence of a comparably large user community in order to be able to provide useful recommendations. Related to that is the typical issue of data sparsity. In many domains, a large number of items in the catalog only have very few (or even no) ratings, which can lead to the effect that they are never recommended to users. At the same time, some users only rate very few items, which makes it hard for CF methods to develop a precise enough user profile. Situations in which there are no or only a few ratings available for an item or a user are usually termed "cold start" situations. A number of algorithms have been proposed to deal with this problem in the literature. Many of them rely for example on hybridization strategies, where different algorithms or knowledge sources are used as long as the available ratings are not sufficient. Finally, as also discussed in [8], some CF algorithms have a tendency to boost the popularity of already popular items so that, based on the chosen algorithm, some niche items have a low chance to be ever recommended.

CF-based music recommendation has some aspects which are quite specific for the domain. Beside the fact that in the case of song recommendation it is plausible to recommend the same item multiple times to a user, it is often difficult to acquire good and discriminative rating information from the user. Analyses have shown that for example on *YouTube* users tend to give ratings only to items they like so that the number of "dislike" statements is very small. While this bias towards liked items can also be observed in other domains, it appears to be particularly strong for multimedia content as provided on *YouTube*, which "degrades" the user feedback basically to unary ratings ("like" statements). With respect to data sparsity as mentioned above, a common strategy in CF recommender systems is to rely on implicit item ratings, that is, one interprets actions performed by users on items as positive or negative feedbacks. In the music domain, such implicit feedback is often collected by mon-

itoring the user's listening behavior, and in particular listening times are used to estimate to which extent a user liked a song.

One of the most popular online music services that uses – among other techniques – collaborative filtering is *Spotify*. Spotify provides several types of radios such as genre radios, artist radios and playlist radios. Once the user has chosen a radio, the system automatically plays one recommended song after the other. The user can give some feedback (like, dislike or skip) on the tracks and this feedback is taken into account and used to adapt the selection of the next recommendations. All these radios use collaborative filtering, and more precisely Matrix factorization [4]. Another interesting feature of Spotify is the Discover weekly playlist, a playlist that is automatically generated each week and that the user can play to discover music he may like. This feature also uses collaborative filtering to select the tracks which are "around" the favorite tracks of the users in the similar users' listening logs [36]. An interesting aspect of Spotify is that it has a number of "social" or community features. Users can create a network of friends and follow the playlists they share.

1.2.2 Content-based Recommendation

Content-based (CB) techniques are rooted in Information Retrieval (IR) and exploit additional information about the available catalog items to generate personalized recommendations. The "content" of an item (e.g., a song or album) can in principle be an arbitrary piece of information describing a certain aspect of the item. Historically, the term *content* was used to refer to the goal of many methods developed in field of IR, which is the recommendation of text documents or web pages, whose content can be automatically extracted. In the context of music recommendation, however, we would also consider information about the artist, the musical genre or any other type of information that can be extracted with music analysis methods as content.

The rough idea of CB recommenders is to look at items which the current user has liked in the past and then scan the catalog for further items which are similar to these liked items. A CB recommender has to implement at least two functionalities: (A) The system first has to acquire and update a "user profile", which captures the user's interests and preferences[9]. (B) The system has to implement a retrieval function, which determines the estimated relevance of a given item for a certain user profile.

Regarding the maintenance of the user profile, one option for new users of an online music site would be to ask them to explicitly specify their favorite artists or genres or rate some songs. Alternatively, existing profile information taken, e.g., from social networks such as *Facebook* could be used as a starting point. After the initial ramp-up, the user profile should be continually updated based on implicit or explicit feedback.

[9] In contrast to CF methods, the user profile in CB approaches is not based on the behavior of the community but only on the actions of the individual user.

How to represent and learn the user profile and how to retrieve suitable items depends on the available information. The most common approach is to represent the user profile and an item's content information in the same way and along the same dimensions.

Table 1.2: Content Information in a Song Database

Title	Artist	Genre	Feel	Liked?
Old man	Neil Young	Country	Melancholy	✓
Perhaps Love	John Denver	Country	Melancholy	✓
On the road again	Willie Nelson	Country	Driving Shuffle	
Harlem Shuffle	Rolling Stones	Rock	Use of Groove	×
...	...	...	...	
Redemption Song	Bob Marley	Reggae	Reggae feel	✓

Table 1.2 shows an example for a content-enhanced music catalog, where the items marked with a tick (✓) correspond to those which the user has liked. A basic strategy to derive a user profile from the liked items would be to simply collect all the values in each dimension (artist, genre, etc.) of all liked items. The relevance of unseen items for the user can then be based for example on the overlap of keywords. In the example, recommending the Willie Nelson song appears to be a reasonable choice due to the user's preference for country music.

In the area of document retrieval, more elaborate methods are usually employed for determining the similarity between a user profile and an item, which for example take into account how discriminative a certain keyword is for the whole item collection. Most commonly, the TF-IDF (term frequency - inverse document frequency) metric is used to measure the importance of a term in a document in IR scenarios. The main idea is to represent the recommendable item as a weight vector, where each vector element corresponds to a keyword appearing in the document. The TF-IDF metric then calculates a weight that measures the importance of the keyword or aspect, that is, how good it characterizes the document. The calculation of the weight value depends both on the number of occurrences of the word in the document (normalized by the document length) as well as how often the term appears in all documents, thus avoiding to give less weight to words that appear in most documents.

The user profile is represented in exactly the same way, that is, as a weight vector. The values of the vector, which represent the user's interest in a certain aspect can for example be calculated by taking the average vector of all songs that the user has liked.

In order to determine the degree of match between the user profile u and a not-yet-seen item i, we can calculate the cosine similarity as shown in Equation (1.5) and rank the items based on their similarity.

$$sim(\boldsymbol{u},\boldsymbol{i}) = \frac{\boldsymbol{u}\cdot\boldsymbol{i}}{|\,\boldsymbol{u}\,||\,\boldsymbol{i}\,|} \tag{1.5}$$

The cosine similarity between two vectors measures the distance (angle) between them and uses the dot product ($\cdot$) and the magnitudes ($|\ \boldsymbol{u}\ |$ and $|\ \boldsymbol{i}\ |$) of the rating vectors. The resulting values lie between 0 and 1.

Generally, the recommendation could be viewed as a standard IR ranked retrieval problem with the difference that we use the user profile as an input instead of a particular query. Thus, on principle, modern IR methods based, e.g., on Latent Semantic Analysis or classical ones based on Rocchio's relevance feedback can be employed, see [20]. Viewed from yet a different perspective, the recommendation problem can also be seen as a classification task, where the goal is to assess whether or not a user will like a certain item. For such classification tasks, a number of other approaches have been developed in the field of Information Retrieval, based, e.g., on probabilistic methods, Support Vector Machines, regression techniques and so on.

1.2.2.1 Content-based Filtering for Music Recommendation

Content-based techniques are particularly appropriate for textual document recommendation, as the content of the documents can be directly used to induce vectors of keywords. This is not possible for music (except maybe for recommending songs based on the lyrics), and other types of content features have to be acquired. On principle, all the various pieces of information that can be automatically extracted through automated music analysis such as timbre, instruments, emotions, speed or audio features can be integrated into the recommendation procedure.

In general, the content features in CB systems have to be acquired and maintained either manually or automatically. In each case, however, the resulting annotations can be imprecise, inconsistent or wrong. When songs are for example labeled manually with a corresponding genre, the problem exists that there is not even a "gold standard" and that when using annotations from different sources the annotations may be contradicting.

In recent years, additional sources of information have become available with the emergence of Semantic Web technologies, see [10, 5], and in particular with the Social Web. In this frame, users can actively provide meta-information about items, for instance by attaching tags to items, thereby creating so-called folksonomies. This is referred to as Social Tagging, and it is becoming an increasingly valuable source of additional information. Since the manual annotation process of songs does not scale well, "crowdsourcing" the labeling and classification task is promising despite the problems of labeling inconsistencies and noisy tags. An important aspect here is that the tags applied to a resource not only tell us something about the resource, e.g., the song itself, but also about the interests and preferences of the person that tags the item.

Beside expert-based annotation and social tagging, further approaches to annotating music include Web Mining, e.g., from music blogs or by analyzing the lyrics of songs, automated genre classification or similarity analysis [9].

Content-based recommendation methods have their pros and cons. In contrast to CF methods, for example, no large user community is required to generate recommendations. On principle, a content-based system can start making recommendations based on one single positive implicit or explicit user feedback action or based on a sample song or user query. More precise and more personalized recommendations can of course be made, if more information is available. The obvious disadvantage of content-based methods when compared with CF methods is that the content information has to be acquired and maintained. In that context, the additional problem arises that the available content information might not be sufficiently detailed or discriminative to make good recommendations.

From the perspective of the user-perceived quality of the recommendations, methods based on content features by design recommend items similar to those the user has liked in the past. Thus, recommendation lists can exhibit low diversity and may contain items that are too similar to each other. In addition, such lists might only in rare cases contain elements which are surprising for the user. Being able to make such "serendipitious" and surprising recommendations is however considered as an important quality factor of an RS. On the other hand, recommending at least a few familiar items – as content-based systems will do – can help the user to develop trust in the system's capability of truly understanding the user's preferences and tastes.

With respect to real-world systems, *Pandora Music* is most often cited as a content-based recommendation service and based on the *Music Genome* project. The idea of the project was to codify every song as a vector of up to 400 to 500 different features (genes), relative to the genre. These features include both characteristics of the music itself (e.g., of structure, rhythm and meter, instrumentation, or tonality) as well as other information such as the roots of the style and other influences, the used recording techniques, characteristics of the lead vocals as well as information related to the lyrics.

The interesting aspect of *Pandora* is that these feature values are assigned manually by musical experts over years. Annotating a song can take an expert up to half an hour[10]. The similarity of tracks can then be easily computed based on a distance metric once a sufficient number of genes is available. A profile of a user is learned by collecting implicit and explicit feedback. The underlying assumption is that music can be classified in an objective way, and that the chosen set of genes is sufficient to capture all aspects that make musical tracks similar to each other.

Another limiting factor is that the manual annotation approach does not scale too well and new songs can only appear in the recommendations when the musical genes have been entered. As mentioned in [22], however, this particular aspect and the fact that the genre is not explicitly encoded in the genes, can also lead to surprising recommendations. As of 2014 [7], *Pandora Radio* has more than 250 million registered users and features more than 80,000 artists and 800,000 tracks in its library.

[10] See `http://en.wikipedia.org/wiki/Music_Genome_Project` for details and examples of *genes*, accessed February 2016.

1.2.3 Further Knowledge Sources and Hybridization

Beside user-provided feedback and content-data also different other types of knowledge sources can be taken into account into the recommendation process. In so-called "demographic" approaches, information about the user's age, sex, education or income group can be factored into the algorithms. Similarly, the user's personality, the hobbies or general interests and lifestyle might be relevant. Beside such demographic and "psychographic" systems, in e-commerce settings, also knowledge-based approaches can be found. So-called critiquing-based systems are an example of such systems, where the user can interactively state and revise the requirements with respect to a certain set of item features. Other knowledge-based systems use explicit domain rules to match user requirements with product features. These types of systems however only play a minor role in music recommendation.

Beside the above-mentioned user-provided tags for resources, Social Web platforms can serve as a source for further knowledge to be exploited for music recommendation. One can for instance try to interpret direct friendship relations in a social network as an indicator of a user's trust into the recommendations of another person and amplify the neighborhood weight for such users in a collaborative approach. Explicit "Like" statements for certain artists or songs represent another natural source to learn about a user's musical preferences.

Finally, the incorporation of information about the user's current context appears to be a particularly important aspect for the music recommendation task. The term context may refer to user-independent aspects such as the time of the day or year but also to user-specific ones such as the current geographic location or activity. In particular the second type of information, that is, including the information about whether the user is alone or part of a group, becomes more and more available thanks to GPS-enabled smartphones and corresponding Social Web applications.

Since the different basic recommendation techniques (e.g., collaborative filtering or content-based filtering) have their advantages and disadvantages, it is a common strategy to overcome limitations of the individual approaches by combining them in a hybrid approach. When, for example, a new user has only rated a small number of items so far, applying a neighborhood-based approach might not work well, because not enough neighbors can be identified who have rated the same items. In such a situation, one could therefore first adopt a content-based approach in which one single item rating is enough to start and switch to a CF method later on, when the user has rated a certain number of items. In [6], Burke identifies seven different ways of how recommenders can be combined. Jannach et al. in [20] later on organize them in the following three more general categories:

- *Monolithic designs:* In such approaches, the hybrid system consists of one recommendation component which pre-processes and combines different knowledge sources; hybridization is achieved by internally combining different techniques that operate on the different sources (Figure 1.1).
- *Parallelized designs:* Here, the system consists of several components whose *output is aggregated* to produce the final recommendation lists. An example

is a weighted design where the recommendation lists of two algorithms are combined based on some ranking or confidence score. The above-mentioned "switching" behavior can be seen as an extreme case of weighting (Figure 1.2).
- *Pipelined designs:* In such systems, the recommendation process consists of multiple stages. A possible configuration could be that a first algorithm pre-filters the available items which are then ranked by another technique in a subsequent step (Figure 1.3).

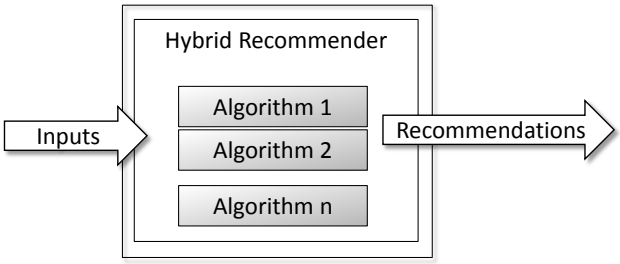

Fig. 1.1: Monolithic hybridization design; adapted from [20].

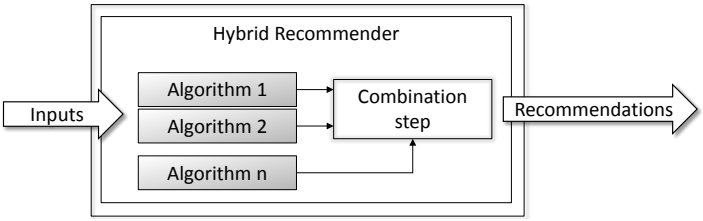

Fig. 1.2: Parallelized hybridization design; adapted from [20].

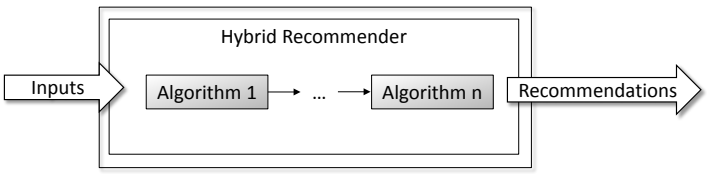

Fig. 1.3: Pipelined hybridization design; adapted from [20].

1.3 Specific Aspects of Music Recommendation

The recommendation techniques discussed so far were typically not designed to work only for a certain class of products and can thus be applied to a variety of domains. There are, however, specific aspects in music recommendation which are slightly different from other domains and have to be considered in the specific system design. In particular, a number of works are devoted on the problem of "playlist generation", which can be considered a special form of music recommendation[11].

Consider the following list of aspects, which is based on Paul Lamere's talk at the *ACM Recommender Systems 2012* conference "I've got 10 million songs in my pocket. Now what?", see also [5, 8].

- *Consumption-related aspects:* First, the recommended items can be either "consumed" immediately or not. In case of the classical book recommender of *Amazon.com*, the delivery of a book needs a couple of days. Thus, the current context of the user at the time of the recommendation is not as important as in situations where the customer immediately wants to listen to a song, e.g., in the case of an online radio station. In that sense, music recommendation shares similarities with video streaming (or IP television) recommendation, where considering the user's context (e.g., the time of the day or whether or not he enjoys the video alone or in a group) is crucial.
  Track consumption time is very low (songs last a few minutes). The systems thus have to generate a lot of recommendations. For instance, most users can listen to more than 20 songs a day, while they rarely read more than 20 books a year.
  A user can play the same song a hundred of times, while, e.g., movies are more rarely watched again and again. Familiarity is a very specific and important feature of music. Users usually like to discover to some new tracks, but at the same time like to listen to the tracks with which they are familiar. A very specific compromise thus exists between familiarity and discovery for user satisfaction.
  Songs are often consumed in sequence. It is important that successive songs form a smooth transition regarding the mood, tempo or style. A good playlist thus not only balances possible quality features like coherence, familiarity, discovery, diversity and serendipity, but must also has to provide smooth transitions.
  Finally, in contrast to many other recommendation domains, tracks can be consumed when doing other things. One can listen to music while working, studying, dancing, etc. and each type of activity fits best with a different musical style.
- *Feedback mechanisms:* With respect to user feedback and user profiling, the *consumption times* (listening durations), track skipping actions and volume adjustments can be used as implicit user feedback in the music recommendation domain. On some music websites users can furthermore actively "ban" tracks in order to avoid listening to tracks which they actually do not like. At the same

[11] See [5] for an in-depth review of approaches for music playlist generation.

time, the *consumption frequency* can be used as another feedback signal. This repeated "consumption" of items seems to be particularly relevant for music, because it is intuitive to assume that the tracks that the users plays the most frequently are the tracks that the users like the most. This information about repeated consumption can also be used in other domains like for instance web browsing recommendation, but it seems to be less relevant for these types of applications [13, 21].
Another typical feature of many music websites is that their users can create and share playlists. Many users create such playlists[12], and the tracks in these playlists usually correspond to the tracks the users like. Playlists can therefore represent another valuable source for an RS to improve the user profiles.

- *Data-related aspects:* Music recommendation deals with very large item spaces. Music websites usually contain tens of millions of tracks. Moreover, other kinds of musical resources can also be recommended, like for instance concerts. Some of these resources should however not appear in recommendation lists, for example karaoke versions, tribute bands, cover versions, etc.
Furthermore, the available music metadata is in many cases noisy and hard to process. Users often misspell or type inappropriate metadata, as for instance "!!!" as an artist's name. At the same time, dozens of bands, artists, albums and tracks can have identical names, which not only makes the interpretation of a user query challenging, but can also lead to problems when organizing and retrieving tracks based on the metadata.
- *Psychological questions and the cost of wrong recommendations:* From a psychological perspective, music represents a popular means of self-expression. With respect to today's Social Web sites, the question however arises if one can trust that all the positive feedback statements on such platforms are true expressions of what users think and what they really like . Additionally, in the music domain, there exists a certain number of "purist" enthusiasts. For such users, music recommendations have to be made very carefully and homogeneity with respect to the musical style and to the artists might be important.

1.4 Evaluating Recommender Systems

One challenging question after developing a novel recommendation algorithm or deploying a new music recommendation service is how to assess the quality or usefulness of the recommendations. In a system with real users the way we measure the service's effectiveness depends on the underlying (business) goal and model. A typical evaluation setting would consist of conducting A/B tests. In such a test, the user community is split into two or more groups and each group receives recommendations using different algorithms. Based on a defined success metric, we can then compare, for example, how long users stay on the site, how many songs they skip,

[12] About 20% of *last.fm* users have created at least one playlist.

how many songs they download, how often they return, etc. An example of such an A/B test where different recommendation strategies were compared – although in a different domain – can be found in [18].

1.4.1 Laboratory Studies

In research and academic settings, unfortunately, such A/B tests can only seldom be conducted as typically no real-world system is available with which such experiments could be made. Researchers therefore often rely on laboratory studies, which usually consist of a few dozen of participants and in which certain aspects of a recommendation system are analyzed.

Let us assume that the design of the recommender system's user interface has an effect on the perceived quality of the recommendations. To that purpose we can design a controlled experiment to test the hypothesis in which we let the subjects interact with a prototype system with two different interfaces. The possible experimental designs include *between-subjects* and *within-subjects*. In a between-subjects design, each subject (participant) receives recommendations through only one of two implemented interfaces. In a within-subject design, each user will see both of them. When the subjects have ended their interactions with the system, they are asked via a questionnaire how they liked the recommendations (and other aspects) of the system.

Based on the answers of the participants we can then check if any of the observed differences between the groups are statistically significant and support our initial hypothesis. Note that in some experiment designs the user behavior during the interaction, e.g., the listening times, can be automatically monitored or logged. In other designs, users are asked to think aloud when interacting with the system.

1.4.2 Offline Evaluation and Accuracy Metrics

Unfortunately, laboratory studies are costly, time-consuming and sometimes hard to reproduce. Since the participants of such studies are often students, they do not form a truly representative sample of a real population. Recommender systems research is therefore mostly based on offline experiments based on historical data sets. These datasets usually contain a set of user ratings for items (or purchase transactions or other forms of implicit feedback), which were collected on some real-world online platform.

The most common evaluation setting in recommender system research is based on the prediction of the relevance of a set of (hidden) items for a certain user, the application of different accuracy or ranking measures, and the repetition of the measurements using cross-validation. When the goal is to predict the value of the hidden ratings, the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE)

can be used. If the goal is to produce top-n recommendation lists, precision and recall are often applied.

While precision and recall represent the most popular evaluation metric in RS according to the study in [17], the absolute numbers reported in research papers should be considered with care as they might not reflect the "true" values very well. As mentioned for instance in the survey paper on RS evaluation by Herlocker et al. [16], in RS evaluation scenarios the so-called ground truth for most user-item ratings is not known. Considering only the known ratings of the test set for the calculation of precision and recall results in unrealistically high values. In addition, it is also not always clear how the set of relevant items is determined from the rating information.

Another shortcoming of classification accuracy metrics is that they only count the number of hits in a recommendation list but not at which position in the list the hits have been found. Intuitively, the relevant items should be placed on the top of the list as they have a higher chance to get the attention from the user. Therefore, ranking measures are often applied in the IR field that take the position of an item into account. An example for such a measure is the "discounted cumulative gain" (DCG), which is applicable mostly for non-binary notions of relevance [25]. The cumulative gain (CG) corresponds the sum of the relevance weights (ratings) of the items up to a certain list length. The idea of the DCG is to reduce the relevance value of items appearing later in the list, usually by a logarithmic factor. Let Rel_j be the relevance score for an item at position j (based on the known rating). The DCG of a ranked list of length k can be calculated as follows:

$$DCG_k = Rel_1 + \sum_{j=2}^{k} \frac{Rel_j}{log_2(j)} \tag{1.6}$$

Variations of this scheme, e.g., concerning the logarithmic base, are also common in the literature. Usually, the DCG is also normalized and divided by the score of the "optimal" ranking so that finally the values of the normalized DCG lie between zero and one.

Note that other domain-specific or problem-specific schemes are possible. In the *2011 KDD Cup*[13], the task was to separate highly rated music items from non-rated items given a test set consisting of six tracks, out of which 3 where highly rated and 3 were not rated by the user.

1.4.3 Beyond Accuracy – Additional Quality Factors

The above-mentioned accuracy metrics are relatively easy to measure given a set of historical rating data. The question however arises whether they really represent the best indicators for the quality of a recommender system. Consider, for example, a music recommender which has detected that all Rolling Stones songs have been

[13] `http://www.kdnuggets.com/2011/02/kdd-cup-2011-recommending-music.html`, accessed February 2016.

rated very highly by a user. As the system is trained to minimize the prediction error, it would then probably recommend even more Rolling Stones tracks to him. However, presenting the user a list full of Rolling Stones songs might not represent a good recommendation even though the prediction was actually good, e.g., in terms of RMSE. Thus, such a list might be boring and not at all surprising for the user. In addition, such a recommender would probably also only focus on popular items (which are often rated highly), which leads to a possibly undesired effect that the major part of songs in the music collection will be never placed in a recommendation list. Therefore, in recent years, measures other than accuracy began to gain increasing attention in the research community.

1.4.3.1 Coverage, Cold Start, Popularity, and Sales Diversity

With the term coverage, either "user-space coverage" or "item-space coverage" can be meant in the literature of RS. User (-space) coverage is a measure that describes for how many of the known users a recommender system is capable of making a (useful) recommendation. When considering the basic neighborhood-based method described in Section 1.2.1, the system might be configured in a way that requires at least N neighbors whose similarity level exceeds a certain threshold. Thus, not for all users – in particular those who have only rated very few items or have a niche taste – recommendations will be calculated as the system's confidence in the recommendations might be too low. The cold start behavior of an RS is also related to coverage and can for example be measured by calculating user coverage and/or the predictive accuracy at different (artificially created) data set sparsity levels.

Item-space coverage (or catalog coverage), on the other hand, typically refers to the question of how many of the existing items can be or, more importantly, are actually ever recommended to users. Item coverage can be measured both in offline as well as in online experiments, for example by analyzing how often each catalog item actually appeared in the first n elements (top-n) of the recommendation lists presented to the users.

Item coverage is also related to sales diversity and the popularity-bias of recommender systems. In [8], Celma discusses the skewed distribution of item popularity and the corresponding music "long tail" in detail. An example of such a long tail distribution is shown in Figure 1.4.

On the x-axis, the items (songs) are sorted according to their popularity, which is measured in terms of, e.g., the playcount on an online platform such as *last.fm*, sales or download numbers, or, when the goal is to measure the diversity of the recommendations, the number of appearances of a song in a recommendation list.

The term long tail refers to the fact that in many domains – and in particular the music domain – some very few popular items account for a large amount of the sales volume. In [8], Celma cites the numbers of a report from 2007 about the state-of-the-industry in music consumption, where 1% of all available tracks were reported to be responsible for about 80% of the sales or that nearly 80% of about 570,000 tracks were purchased less than 100 times.

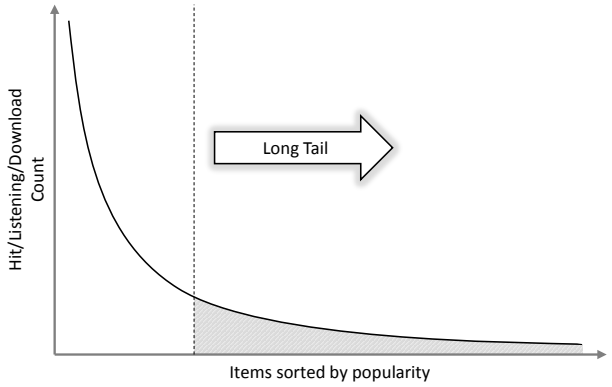

Fig. 1.4: Long tail distribution.

Given this skewed distribution toward popular songs and mainstream artists, it could therefore be – according to Marketing theory – a goal to increase sales of items in the long tail. Recommender systems are one possible method to achieve such a goal and studies such as [37] or [12] have analyzed how recommenders impact the buying behavior of customers and the overall sales diversity. On the one hand, one can observe that in some domains a recommender can help the user to better explore the item space and find new items he or she was not aware of. On the other hand, there is however a danger that depending on the underlying strategy and algorithm the usage of a recommender system can lead to the undesired effect of further boosting already very popular items as recommending blockbusters to everyone is a comparably safe strategy.

1.4.3.2 List Diversity, Novelty, Serendipity and Familiarity

Beside the global sales diversity of a platform, the diversity of recommendations for an individual user can be an important quality factor for the customer. When given a Beatles seed song on an online music platform, recommending a playlist of 10 other Beatles songs or, even worse, 10 cover versions of the same song, might be technically plausible but perhaps not what the user would enjoy. Therefore, it is often advisable to make sure that the recommendation list (playlist) is not monotonous and that the items are not too similar to each other.

A possible strategy could therefore be to try to include items in the recommendation list that increase the diversity of the list, items that are supposedly novel for the user, as well as items which are to some extent surprising (but relevant) for the user. In order to control diversity, one needs a measure of item similarity. When following a content-based recommendation approach as described in Section 1.2.2, the underlying similarity metric can be used to compare two tracks. Alternatively, one

could use metadata such as genre, artist, etc. Based on such a measure, an overall diversity metric for a list can be derived, e.g., by computing and aggregating the pairwise similarities.

Another related criterion is novelty. Novelty can be measured in user studies via a questionnaire. Some researchers also propose offline evaluation approaches which use the general popularity of an item to estimate the novelty of a recommendation list, assuming that highly popular items are not novel to the user. Alternatively, schemes that use the time stamps of the ratings to assess the novelty of an item are possible, see [32].

Serendipity is also often mentioned as a desirable playlist characteristics. This concept is often referred to as a measure of how surprising and unexpected, but accurate, the recommendations are. When a user is pointed through a recommendation list to a track of a genre, style or artist she or he usually does not particularly like, and the user finds that he likes the track, then the recommendation can be considered as being serendipitous. This also corresponds to valuable recommendations given by friends or music enthusiasts. Serendipity can also be measured in a user study and approximated in offline experiments based on the similarity of items and the deviation of recommendations from obvious recommendations. Providing only serendipitous recommendations can however be dangerous, and it is also important that the recommendations include a set of items the user is familiar with as these items can help to increase the user's trust in the system.

An example of a user study on novelty and familiarity that includes 288 participants can be found in [8]. In that study the users were asked to rate recommended songs based on excerpts of 30 seconds of duration. The recommendations included both tracks the users already knew and tracks which were novel to them. One of the observed results was that users rated those songs much higher which they already knew and the perceived quality of the system increased when familiar songs were recommended. One of the insights and conclusions of the study therefore was that a recommender should provide additional contextual information such as an explanation as to why a certain song had been recommended.

1.4.3.3 Adaptivity, Scalability, and Robustness

The quality measures discussed so far focus mainly on the utility of the recommendations for an individual user or a service provider. Beside that, other aspects can be relevant for the practical success of a recommender system.

Adaptivity refers to the capability of an RS to quickly adapt the recommendations based on very recent events. Such an event could for instance be that a track becomes popular over night, e.g., due to its usage in a popular TV advertisement or its relation to some other event. In some domains such as news recommendation, items can become outdated very quickly as well. Another perspective on adaptivity is the system's rate of taking changes and additions in the user profile into account. When users rate tracks, they might expect that their preferences are immediately taken into account, which might however not be the case if the underlying algo-

rithm is based on a computationally expensive training phase and models are only updated, e.g., once a day.

Scalability is a characteristic of recommender systems which is particularly relevant for situations where we have to deal with millions of items and several millions of users as it is the case in music recommendation. Techniques such as nearest-neighborhood algorithms do not scale even to problems of modest size. Therefore, only techniques which rely on offline pre-computation and model-building work in practice.

Robustness typically refers to the resistance of a system against attacks by malevolent users who want to push or "nuke" certain artists. Recent works such as [27] have shown that for example nearest-neighbors algorithms can be vulnerable to various types of attacks whereas model-based approaches are often more robust in that respect.

1.5 Current Topics and Outlook

In this chapter we have discussed the basic techniques for building and evaluating music recommender systems, which have already been successfully implemented in various domains. In this final section we will shortly discuss three topics which have attracted increased interest in the recommender systems research community also outside the music recommendation field: context-awareness, the incorporation of social web information in the recommendation process, and sequential recommendation.

1.5.1 Context-Aware Recommendation

When the recommended music is "consumed" immediately, the current situation or context of the listener can be of extreme importance. You might, for example, be interested in different types of music depending on the time of the day or depending on what you are currently doing. In the morning, on the way to work, you might enjoy a different type of music than when doing sports. But the term context in music recommendation can have even more facets. It can be the social environment (e.g., being part of a group or alone, one's geographical location, and even the current weather and your current emotional state). All these aspects may influence what type of music will be most appropriate as a recommendation. The context may finally refer to general and non-personal characteristics such as the time of the year (think of recommending Christmas songs in May) or very specific ones like the set of tracks which you have been listening to during the current session.

When thinking about our main types of recommendation approaches (collaborative filtering and content-based filtering), the intuition is that CF methods are more suited of taking context into account. For instance, a CF-based recommender that

takes into account what your friends have been recently listening to can help to alleviate at least some of the problems. On the contrary, traditional content-based methods are for instance not always capable to take the cultural background of a track into account, except for cases in which cultural information can be extracted from tags or metadata annotations.

Kaminskas and Ricci classify the possible contextual factors in three major groups [22]:

- Environment-related context, e.g., location, time or weather.
- User-related context, e.g., the current activity, demographical information (even though this can be considered part of the user profile, and provides information about the environment), and the emotional state
- Multimedia context, which relates to the idea of combining music with other corresponding resources such as images or stories.

In the same work, the authors then review a set of context-aware prototypical music recommenders and experimental studies in this area. They conclude that research so far is "data-driven" and that researchers often tend to fuse given contextual information into their machine learning techniques. Instead, the authors advocate a "knowledge-based" approach, where expertise, e.g., from the field of psychology, about the relationship between individual contextual factors and musical perception is integrated into the recommendation systems.

1.5.2 Incorporating Social Web information

The emergence of what is called "Web 2.0" has dramatically changed how we behave in the online world. We are no longer pure consumers of edited content but we actively annotate, "like" and comment items on resource-sharing platforms, we voluntarily post information about our current situation, mood or interests on the Social Web, review items on e-commerce sites or even write our own (micro-)blogs.

Given the discussions above, e.g., on contextual parameters that influence what music should be recommended, it is obvious that the Participatory Web opens new opportunities for music recommendation. On the one hand, more information about the users, in particular their preferences, tastes and current state, can be found online; on the other hand – through social annotations and tags – more information about the tracks in the catalog becomes available.

In the last years, the exploitation of social tagging data in the recommendation process was one of the key topics in RS research. In the context of music recommendation – but also in other domains – user-contributed tags can be simply seen as additional content information and, on principle, standard CB methods could be applied. However, in contrast to automatically extracted or manually annotated metadata, tagging data often contain lots of noise. While some tags for a certain track such as "classic jazz" or "dance music" carry potentially valuable information about the song, users may also tag an album in their personal and subjective view,

e.g., with tags like "own it". Other problems with tags include that there might be malicious users who add various types of unusable or noisy tags to the data.

In [35], various methods for acquiring high-quality tags and addressing the problem of the lacking common vocabulary are proposed, see also [22]. The methods include small-scale data collection within a defined user group and vocabulary, harvesting social tags from online music sites, and using tagging games or different strategies to automatically mine tags from other web sources. A complementary approach to harmonize the vocabulary on tagging-enabled platforms is the use of "tag-recommenders". Such recommenders can already be found on today's resource sharing platforms such as *delicious.com* and make tagging suggestions to the users based on RS technology.

As discussed in [24], user-provided tags may carry different types of valuable information about tracks such as genre, mood or instrumentation that can help to address some challenging tasks in Music Information Retrieval such as similarity calculation, clustering, (faceted) search, music discovery and, of course, music recommendation. The major research challenges include however the detection and removal of noise and, as usual, cold start problems and the issue of lacking data for niche items.

1.5.3 Playlist Generation

As mentioned in Section 1.3, music is typically played in a sequential manner, according to playlists. This means that the relationships between the successive tracks is important. For instance, the transitions between the tracks may be important, as well as the overall diversity of the recommended tracks, the general topics or themes, the musical path from the first track to the last track, etc. For that reason, we can often consider the music recommendation problem as a playlist generation problem, i.e., the recommendation of an ordered collection of tracks.

Although playlist generation can be considered as a special case of track recommendation, it goes slightly beyond, as a playlist itself can be considered as an artistic resource. Moreover, as users like to listen to tracks they already know and rarely want to discover one single track at a time, providing static recommendation lists as done for instance for movies is not often relevant. For that reason, several track recommendation scenarios actually correspond to a form of playlist generation:

1. Simple playlist generation: a seed track is selected by the user, or some set of desired characteristics such as a minimum tempo, a set of genres, etc. and a whole playlist is generated. This scenario is for instance used by iTunes Genius and the playlist generation service of The Echo Nest.
2. Repeated track recommendation for playlist construction: each time the user adds a track, a new list of recommended tracks is provided, and the user can use this list to add a new track to the playlist. This scenario was for instance used in the Rush application [3].

3. Radios: recommended tracks are automatically played one after the other and the user can only skip them if he or she does not want to listen to them. This scenario requires few user effort and is the most frequent in commercial platforms (Spotify, Pandora, last.fm, etc.).

Playlist generation is a part of the research literature since the early 2000's [28] but did not attract much attention until recently. Among the recent work, [15] proposed a frequent pattern mining approach where patterns of latent topics are extracted from user playlists and then used to compute recommendations. [14] exploited long-term preferences including artists liked on Facebook and usage data on Spotify to build playlists which consist in the most popular tracks of the artists who are the most similar to the artists the user likes. A statistical approach was presented in [26], where random walks on a hypergraph are used to iteratively select similar tracks. In the same spirit, [11] proposed a sophisticated Markov model in which tracks are represented as points in the Euclidean space and transition probabilities are derived from the corresponding Euclidean distances. The coordinates of the tracks are learned using a likelihood maximization heuristic. [19] went further by taking into account the characteristics of the tracks that are already in the playlist, and proposed an heuristic which tries to mimic these characteristics in the generation process.

1.6 Concluding Remarks

The way we consume music has dramatically changed during the last decade. Today, millions of tracks are instantly available through an ever-increasing number of online music services. Finding suitable music for a certain listening situation or discovering new music becomes more and more challenging given the millions of songs which are available for instant download. Music recommenders shall help users in different ways, e.g., discovering new songs or artists, exploring the catalog, creating personalized playlists or recommending music that corresponds to the situation of the user.

1.7 Further Reading

In this chapter we have introduced the basic techniques for building such systems, which have been successfully applied in industry in various domains over the last decade. We provided short explanations of several such techniques. For a comprehensive overview of other methods, see for instance [1, 23, 32, 20].

We have shown that music recommendation has some particularities which have to be taken into account. While some of them – like certain types of context – are already addressed in current research, we believe that in particular psychological aspects of music perception have to be taken into account better in future research on

music recommendation. For an overview on context-aware music recommendation, see [22]. For an overview on context-aware recommendation in general, see also [2].

We have also introduced some of the basic techniques for evaluating the recommendations and pointed out some limitations of the current evaluation strategies. More details about state-of-the-art user-centric evaluation procedures can be found in [30]. Further questions of experimental design, measurement and analysis, which are common in social sciences, are covered in detail in [29], and comprehensive overviews of the common offline evaluation schemes for recommender systems can be found in [16, 33].

References

1. Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
2. Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 217–253. Springer, 2011.
3. Dominikus Baur, Sebastian Boring, and Andreas Butz. Rush: Repeated Recommendations on Mobile Devices. In *Proc. IUI*, pages 91–100, New York, NY, USA, 2010. ACM.
4. Erik Bernhardsson. Systems and methods of selecting content items using latent vectors, August 18 2015. US Patent 9,110,955.
5. Geoffray Bonnin and Dietmar Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys*, 47(2):1–35, 2014.
6. Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.
7. Matt Burns. The Pandora One subscription service to cost $5 a month, `http://techcrunch.com/2014/03/18/the-pandora-one-subscription-service-to-cost-5-a-month/`, 2014, accessed February 2016.
8. Òscar Celma. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.
9. Òscar Celma and Paul B. Lamere. Music recommendation tutorial. ISMIR'07, `http://ocelma.net/MusicRecommendationTutorial-ISMIR2007/slides/music-rec-ismir2007-low.pdf`, September 2007, accessed February 2016.
10. Òscar Celma and Xavier Serra. FOAFing the music: Bridging the semantic gap in music recommendation. *Journal of Web Semantics*, 6(4):250–256, 2008.
11. S. Chen, J. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *Proc. KDD*, pages 714–722, New York, NY, USA, 2012. ACM.
12. Daniel M. Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM Conference on Electronic Commerce (EC'07)*, pages 192–199, New York, NY, USA, 2007. ACM.
13. Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Transactions On Information Systems (TOIS)*, 23(2):147–168, 2005.
14. Arthur Germain and Jacob Chakareski. Spotify me: Facebook-assisted automatic playlist generation. In *Proc. MMSP*, pages 25–28, Piscataway, NJ, USA, 2013. IEEE.
15. N. Hariri, B. Mobasher, and R. Burke. Context-aware music recommendation based on latent topic sequential patterns. In *Proc. RecSys*, pages 131–138, New York, NY, USA, 2012. ACM.

16. Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.
17. D. Jannach, M. Zanker, M. Ge, and M. Gröning. Recommender systems in computer science and information systems: A landscape of research. In *Proc. EC-WEB 2012*, Heidelberg / Berlin, 2012. Springer Verlag.
18. Dietmar Jannach and Kolja Hegelich. A case study on the effectiveness of recommendations in the mobile Internet. In *Proceedings of the 2009 ACM Conference on Recommender Systems (RecSys'09)*, pages 41–50, New York, NY, USA, 2009. ACM.
19. Dietmar Jannach, Lukas Lerche, and Iman Kamehkhosh. Beyond hitting the hits: Generating coherent music playlist continuations with the right tracks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 187–194, New York, NY, USA, 2015. ACM.
20. Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerahrd Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2011.
21. Niclas Jones, Pearl Pu, and Li Chen. How Users Perceive and Appraise Personalized Recommendations. *User Modeling, Adaptation, and Personalization (UMAP 2009)*, pages 461–466, 2009.
22. Marius Kaminskas and Francesco Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, Vol. 6(23):89–119, 2012.
23. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
24. Paul B. Lamere. Social tagging and music information retrieval. *Journal of New Music Research*, 37(2):101–114, 2008.
25. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
26. B. McFee and G. Lanckriet. Hypergraph models of playlist dialects. In *Proc. ISMIR*, pages 343–348, 2012.
27. Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, 7(4):23, 2007.
28. F. Pachet, P. Roy, and D. Cazaly. A combinatorial approach to content-based music selection. *Multimedia*, 7(1):44–51, 2000.
29. E.J. Pedhazur and L.P. Schmelkin. *Measurement, Design, and Analysis: An Integrated Approach*. Lawrence Erlbaum Associates, 1991.
30. Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: Survey of the state of the art. *User Model. User-Adapt. Interact.*, 22(4-5):317–355, 2012.
31. P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW'94)*, pages 175–186, New York, NY, USA, 1994. ACM.
32. Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
33. Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender Systems Handbook*, pages 257–297. Springer, 2011.
34. Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating word of mouth. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM.
35. Douglas Turnbull, Luke Barrington, and Gert R. G. Lanckriet. Five approaches to collecting tags for music. In *Proc. ISMIR 2008*, pages 225–230, Philadelphia, 2008.
36. Martin Vacher. Introducing Discover Weekly: Your ultimate personalised playlist, `https://press.spotify.com/it/2015/07/20/introducing-discover-weekly-your-ultimate-personalised-playlist`, accessed February 2016.

37. Markus Zanker, Marcel Bricman, Sergiu Gordea, Dietmar Jannach, and Markus Jessenitschnig. Persuasive online-selling in quality & taste domains. In *Proceedings of the 7th International Conference on Electronic Commerce and Web Technologies (EC-Web'06)*, pages 51–60, Heidelberg / Berlin, 2006. Springer Verlag.