A Comparison of Playlist Generation Strategies for Music Recommendation and a New Baseline Scheme

Geoffray Bonnin and **Dietmar Jannach**
Technische Universität Dortmund
44227 Dortmund, Germany
{firstname.lastname}@tu-dortmund.de

Abstract

The digitalization of music and the instant availability of millions of tracks on the Internet require new approaches to support the user in the exploration of these huge music collections. One possible approach to address this problem, which can also be found on popular online music platforms, is the use of user-created or automatically generated playlists (mixes). The automated generation of such playlists represents a particular type of the music recommendation problem with two special characteristics. First, the tracks of the list are usually consumed immediately at recommendation time; secondly, songs are listened to mostly in consecutive order so that the sequence of the recommended tracks can be relevant.

In the past years, a number of different approaches for playlist generation have been proposed in the literature. In this paper, we review the existing core approaches to playlist generation, discuss aspects of appropriate offline evaluation designs and report the results of a comparative evaluation based on different datasets. Based on the insights from these experiments, we propose a comparably simple and computationally tractable new baseline algorithm for future comparisons, which is based on track popularity and artist information and is competitive with more sophisticated techniques in our evaluation settings.

Introduction

Internet technology and the digitalization of music almost makes us forget how we discovered music in the past. Not so long ago, if we wanted to explore the songs in the discography of some artist recommended by a friend, we had to find the corresponding CDs, vinyls or even cassettes in a shop. Today, we simply browse to our favorite online shop or music platform, where we can purchase and download the music to our computers and smart phones or listen to it online through the browser. It also seems that we do not even need friends to get recommendations. There are, for example, social web platforms on which people share their manually created playlists; furthermore, some sites are capable of automatically creating personalized playlists (mixes) containing tracks that are likely to correspond to our tastes.

Music recommendation is often considered to be a difficult task for a variety of reasons. The challenges for example include that there are often millions of available tracks, there is a lack of structured or consistent meta-data or other content information, and that we often only have limited amounts of customer feedback that can be used to personalize the recommendations[1].

Playlists, i.e., lists of sequentially ordered tracks, represent one possible approach to deal with some of these challenges and help users explore the item space. Sometimes, playlists are also considered to be a recommendation technique by themselves (Fields 2011). On popular platforms such as last.fm, for example, sequences of songs are automatically generated starting from some seed song or artist. Other sites such as 8tracks.com allow users to share their playlists with others. In either case, the playlists will usually contain at least some items which are novel for the user.

The automated creation of such playlists however comes with additional challenges that go even beyond the above-mentioned ones for music recommendation in general. In particular, since the tracks of a playlist are usually played one after the other, neighboring tracks should perhaps not be too different from each other, e.g., in terms of mood, tempo or style. When our goal is to automatically generate such playlists, one of the key questions is therefore how we can assess the quality of our playlist. In fact, there might be a number of different factors that influence the perceived value of a playlist, including, e.g., the coherence of the list, the variety of songs or the *freshness* of the songs (Fields 2011).

One typical approach to evaluate computer-generated playlists that is also independent of the specific algorithm is to compare them with playlists that are manually crafted by the user community. The assumption is therefore that these lists, which are often created by music enthusiasts, are of high quality and implicitly respect various quality criteria.

In the last few years, a number of approaches for the automated generation of playlists have been proposed in the literature, among them techniques that are based, e.g., on track co-occurrence, sequential patterns, or additional information about the tracks. In addition, different evaluation protocols for offline experimental designs have been introduced. The results reported in the literature are however often hard to compare as they sometimes focus on a particular family of techniques and use different baseline algorithms. Further-

[1]For a further discussion of typical challenges in music recommendation, see, e.g. (Lamere and Celma 2011) or (Celma 2010).

more, it often remains unclear if the proposed techniques are actually scalable when it comes to realistic and large item catalogs.

In this work, we will discuss questions regarding possible approaches to evaluating the quality of playlist generation and review existing playlist approaches. We will then report the results of a comparative evaluation of different algorithms based on different datasets and discuss our insights related to playlist quality and computational complexity. Finally, we will propose a comparably simple and computationally efficient new baseline algorithm for future comparisons which relies on track popularity and artist information. Our algorithm outperforms other approaches on two of the three datasets on which we performed experiments.

Automated Playlist Generation

A playlist is usually defined to be an ordered sequence of musical tracks. For each track, we might also have some extra information, e.g., the composer, artist, lyrics, tags, etc. The playlist generation problem typically consists in creating such a list given either some *seed* information or semantic description (Barrington, Oda, and Lanckriet 2009). The seed information can be another ordered set of tracks corresponding to the listening history so far. An example for a semantic description could be "Joyful songs."

In this work, we limit ourselves to the first case and assume that the input to the playlist generation problem consists of the listening *history* so far. Given this history, two generation strategies are possible. The first is to present recommendation lists of tracks to the user, and each time a track is selected the process is repeated (Baur, Boring, and Butz 2010). The second is to generate a whole playlist continuation with no interaction with the user as done on last.fm. In both cases the problem comes down to the computation of the score of a candidate track t given a playlist history $h = \langle t_1, t_2, ..., t_i \rangle$.

Evaluation of playlist recommendations

Before we review different playlist generation approaches, we will briefly discuss how we can compare the quality of such playlists. In general, the purpose of a recommender system (RS) is to present resources that correspond to the needs and tastes of a user. The best related criterion to determine the quality of an automatically generated playlist is probably the satisfaction of the user. On a real-world music platform, this could be measured for example based on listening times, customer return rates, track downloads or with the help of surveys. In principle, one could also try to use more objective measures such as the coherence, similarity or diversity to assess and compare playlists. In this work, however, we focus on the *accuracy* of the recommendations. As usual in RS research, we try to compare the lists generated through different approaches based on a historical dataset, in this case a given set of real-world playlists.

In most RS evaluation settings, a user-item rating matrix is used as the basis for the evaluation, where a fraction of the data is hidden during the training phase and the hidden ratings, which should be predicted by the RS and are used to measure the accuracy. When evaluating playlist recommendations, the setting is slightly different and not based on rating information. Usually, the idea is however again to "hide" some information, and let the recommendation algorithm make guesses about the rest.

Measuring hit rates Classical information retrieval measures can also be applied in standard RS evaluation settings. In particular, if recommendation lists are automatically generated, we can then determine measures like precision and recall. This scheme can be applied for playlists by considering each playlist as being a user and the playlist elements as relevant items, from which we hide individual elements in the training phase[2].

An evaluation method of this type is used, e.g., by (Hariri, Mobasher, and Burke 2012), who hide the last element of each given playlist, which has then to be recommended by the algorithm. The corresponding metric is the *hit rate*. In general, any subset of playlist elements could be hidden in such a protocol. Removing the last one however is based on the assumption that the sequential history of a playlist can be relevant.

$$HitRate(Train, Test) = \frac{1}{\|Test\|} \sum_{(h,t) \in Test} \delta_{t, R_{Train}(h)}$$

where $R_{Train}(h)$ is a recommendation list of a defined length computed by an algorithm based on $Train$ and the playlist beginning h and $\delta_{t,R_{Train}(h)} = 1$ if $R_{Train}(h)$ contains t and 0 otherwise.

The limitation of this evaluation metric is that it corresponds to the assumption that the actual next tracks in the playlist are the only relevant tracks that can be recommended, although some other tracks may be relevant. In other words, it is possible that hundreds of tracks are relevant, but as the recommender has to select a subset of them, the actual next tracks of the test playlists might not be recommended. As it is impossible to know how many tracks are relevant for each situation, it is reasonable to analyze the accuracy of a system using longer recommendation lists. The assumption is then that there is a correspondency between the size of the recommendation lists and the average number of relevant tracks. Still, the hit rate can only be considered to be a lower bound for the accuracy.

Measuring the average log-likelihood Another way to measure accuracy is to use the *average log-likelihood*. The average log-likelihood can be used to measure how likely a system is to recommend the tracks of a given set of playlists through a weighted random process. More precisely, given a test set of playlists, the average log-likelihood $\mathcal{ALL}$ can be determined by computing the probability of observing each next track according to the corresponding playlist history and some model learned on the training data as follows:

[2]Notice that even in cases where information about the playlist creator is available, it will not be used in such a protocol and the generated playlists are not personalized. The incorporation of this type of information could however further help to improve the playlist quality.

$$\mathcal{ALL}(Train, Test) = \frac{1}{\|Test\|} \sum_{(h,t) \in Test} \log P_{Train}(t \mid h)$$

where $P_{Train}(t \mid h)$ corresponds to the probability of observing t given h according to a model learned based on $Train$. Research on music recommendation using playlists that use this metric includes (McFee and Lanckriet 2011) and (Chen et al. 2012). Obviously, the application of this measure requires that the output of a playlist recommender can be expressed as probability values for each song which is between 0 and 1.

In contrast to the hit rate, which provides a realistic lower bound on the accuracy that is directly interpretable, this metric is not interpretable on an absolute scale: the possible values vary between $-\infty$ (at least one track in the test set has a corresponding 0 probability in the model) and 0 (all probabilities in the model for all tracks in the test set are 1). This means that 0 probabilities must be avoided, thus requiring an additional smoothing step and the impossibility of a 0 average log-likelihood result. Thus, this metric only allows us to compare the results of different smoothed generative approaches without knowing if the best one is actually good.

Another limitation is that given the number of tracks that can be recommended, a generative process may not be suited for music recommendation. Even when weighted according to the distribution of the model, a random generation may present irrelevant tracks too often.

Compared to the hit rate, the only advantage of using the log-likelihood would be its better accuracy in the comparison of the smoothed generative version of the approaches. It should thus only be used as a second step, after having made sure that the evaluated predictive version of the approaches have a satisfying lower bound of accuracy on an absolute scale. As this paper is aimed at being a first step towards the analysis of the different approaches for playlist generation, we limit ourselves to the hit rate in the evaluations reported in this paper.

Computational complexity Another important fact that should be taken into account in the context of music recommendation is the computational complexity. Indeed, as opposed to, for instance, movie recommendation, for which recommendations can be computed offline and updated regularly, music recommendation can be highly dynamic and contextual. Users usually listen to tracks in sequence, where each track lasts a few minutes. Therefore, a music recommender should be able to provide fast contextual recommendations. Moreover, as the number of tracks that can be recommended is usually very high, the efficiency of the training phase can become crucial. In the subsequent analysis of algorithms, we will thus also briefly discuss aspects of computational complexity.

Playlist Generation Approaches

In the following, we review existing approaches to playlist generation. The task of each technique is to calculate a score for each next possible track t given a playlist beginning $h = \langle t_1, t_2, ..., t_i \rangle$. The resulting scores – which in some cases correspond to probability estimates – can then be used to filter and rank the remaining tracks.

Markov chains Given that the recommendation scenario usually is to generate a good continuation of a playlist, attempting to recommend tracks that represent a smooth transition with the previous track is an obvious approach. This corresponds to the Markov property and leads to a first-order Markov model in which states can correspond to track IDs or any other representation of the tracks. Given a history h of a playlist and a candidate track t, the probability of t in such a model thus only depends on t_i, the last element of h.

$$P_{\text{Markov}}(t \mid h) = P(t \mid t_i) \qquad (1)$$

Examples of playlist modeling approaches based on this strategy include (McFee and Lanckriet 2011) and (Chen et al. 2012). (McFee and Lanckriet 2011) compare three simple approaches to assign transition probabilities to a Markov model: a uniform distribution, a distribution based on artist popularity and a set of k-Nearest-Neighbors models (kNN) that use the similarity between tracks based on tags and the audio signal with different parameters. They also experiment with a mixture of these models where weights are computed according to a state-of-the-art convex optimization algorithm. Experimental evaluations based on the average log-likelihood show that the only models to outperform the uniform model are the artist popularity model and the mixture model.

In (Chen et al. 2012), tracks are represented by vectors in the Euclidean space. The corresponding coordinates are then learned through a regularized maximum-likelihood embedding of Markov chains where the transition probabilities are a function of the Euclidean distance between the tracks. Experimental evaluations based on the average log-likelihood on a radio station track-list dataset show that the proposed Latent Markov Embedding (LME) model outperforms a Markov model whose transition probabilities correspond to the track frequency (which is also called a bigram model) and the uniform model.

The major limitation of these models is that the assumptions on which they are based may be too strong: the choice of the next track by a user may or may not depend only on the previous track. Although tracks are usually being listened one after the other and transitions between tracks surely have some importance, in practice, the rules users follow to build playlists can be quite different and often contradict this assumption, see (Cunningham, Bainbridge, and Falconer 2006).

Frequent patterns Another possibility to recommend tracks for playlist generation is to extract frequent patterns from playlists. The common techniques are association rule and sequential pattern mining. An association rule (Agrawal, Imieliński, and Swami 1993) has the form $A \rightarrow C$, where A and C are two itemsets. A is called the *antecedent* and C the *consequent*. The relevance of patterns and association rules is usually measured in terms of *support* and *confidence*. The support of a pattern corresponds to the probability of observing all of its elements at the same time. The confidence of a rule corresponds to the conditional probability of finding the elements of the consequent in presence of the elements of the antecedent.

The major difficulty of association rules is to extract such relevant rules efficiently and find appropriate threshold values. Once the relevant patterns have been extracted, the score of the next track t can be computed according to the following formula:

$$\text{score}_{pattern}(t,h) = \frac{1}{\|\Omega\|} \sum_{\omega \in \Omega} \text{confidence}(\omega \rightarrow t) \quad (2)$$

with Ω the set of all possible antecedents that can be considered in h.

The resulting recommenders can be optimized by varying different parameters, including the maximum size of the patterns, the minimum support and confidence thresholds and the size of the window in which the patterns are being extracted and retrieved.

Sequential patterns are a sequential version of association rules (Agrawal and Srikant 1995) in which the order of the elements in the pattern is also taken into account in the mining process. The additional constraints of sequential patterns over association rules can in general lead to more accurate recommendations, but the approach has a higher computational complexity and requires a larger amount of training data. Another possible limitation of this approach might be the comparably small confidence values for the extracted patterns given the usually high sparsity of musical datasets.

The overall effectiveness of sequential patterns however depends on the type of data. If the data is not ruled by some sort of sequentiality, then association rules provide the same accuracy and should be preferred. Our hypothesis in this context is that playlists are at least partly governed by sequential constraints and that sequential patterns should lead to a higher accuracy. Since we are not aware of previous works who have evaluated sequential track patterns for playlist generation, we performed a comparative evaluation of association rules and sequential patterns which is discussed later in this paper.

Neighborhood recommenders Another way to exploit co-occurrences of tracks is to use a k-nearest-neighbors (kNN) recommender which is based on the similarity between playlists (not between tracks, as in (McFee and Lanckriet 2011)) and which can be calculated using a binary cosine similarity measure:

$$\text{sim}_{\text{p}}(p,q) = \frac{\|p \cap q\|}{\sqrt{\|p\|\|q\|}}$$

Given N_h nearest neighbors for a playlist, the score of a track t could then be defined as:

$$\text{score}_{k\text{NN}}(t,h) = \sum_{n \in N_h} \text{sim}_{\text{p}}(h,n) \cdot \delta_{t,n} \quad (3)$$

A similar kNN approach was proposed in (Hariri, Mobasher, and Burke 2012) and used as a basis for a more sophisticated recommender which uses sequential patterns of latent topics based on tags. The authors compare the topic-aware recommender they propose to the kNN approach used alone on a small dataset (about 20,000 tracks), and obtain a relatively slight improvement in terms of the hit rate. They also compare these approaches to a Bayesian Personalized Ranking (BPR) recommender (Rendle et al. 2009) (a recent learning-to-rank technique), and a content-based recommender. The content-based recommender represents tracks as vectors whose dimensions correspond to artist names, genre, era and album title and uses the kNN approach to compute recommendations. The BPR algorithm slightly outperforms the content-based recommender when many tracks are recommended and is slightly outperformed by the two other approaches.

Similar to association rules, such a kNN approach not only exploits information about the collocation of items in playlists but also takes the number of shared items in each playlist into account when estimating the probability. However, association rule mining is based on counting the frequency of patterns for all users in an offline process. The described kNN approach, in contrast, dynamically computes a "local" probability using the k most similar playlists. In other words, the above-mentioned limitation with respect to low confidence values is reduced. The computation of neighborhoods and playlist similarities is however computationally complex both in terms of time and space, making the approach intractable when recommendations have to be made in real time.

Playlists as users In principle, if we interpret the playlist generation problem to be similar to the item prediction problem in typical RS settings by considering playlists to be users, existing RS algorithms for item recommendations based on implicit feedback can be applied including recent learning-to-rank techniques. In particular, the BPR-approach from (Rendle et al. 2009) has been included in previous comparative evaluations for playlist or music recommendation, see e.g. (Hariri, Mobasher, and Burke 2012) and (McFee et al. 2012). The experiments in the last two papers however show that the plain BPR method can be easily outperformed by other methods in particular problem settings.

Content-based approaches So far, kNN approaches with binary cosine similarities between playlists seem the most likely to provide sufficient accuracy but can be computationally complex. Pattern-based approaches can overcome this drawback when using appropriate parameters but may not be very accurate given the sparsity of musical datasets. Using additional information, one can try to avoid the complexity of the kNN approach or enhance the confidence of pattern-based approaches. Such additional information can be the content of the tracks (lyrics, spectrum, etc.), the similarity of musical features (Flexer et al. 2008), user tags, or more simple elements such as artist names.

Some of the aforementioned approaches use some forms of content and meta-data. For instance, the topic-aware, hybrid recommender of (Hariri, Mobasher, and Burke 2012) uses tags to determine topics, but does not solve the scalability problem of the underlying kNN approach. As well, (McFee and Lanckriet 2011) experiment with some Markov models that use tags, the audio signal and artist names. However, their approach does not solve the problem of the too strong assumption of the Markov property.

Regarding the incorporation of additional information into the recommendation process, we hypothesize that the use of artist names is particularly promising as this type of data is objective, easy to obtain and to process (as opposed to, for instance, information about the playlist topic, genre or style). Moreover, as will be shown, users often put several tracks from one artist in their playlists.

Popularity-based approaches In many application domains for RS and in particular in the music domain (Celma 2010), we can observe a so called "long tail" distribution of items, meaning that a small subset of the items accounts for the majority of transactions or interactions. This popularity bias results in the fact that simple popularity-based approaches, which present the same set of popular items to everyone, can represent a comparably hard baseline (Cremonesi, Koren, and Turrin 2010).

Given these observations we have included two approaches which are based on popularity combined with artist information in our experiments.

"Same artists - greatest hits" (SAGH): In (McFee et al. 2012), the authors propose a baseline algorithm for music recommendation – not in the context of playlists – called "Same artists - greatest hits", which simply recommends the most popular songs of the artists appearing in the user's listening history. Their experiments on the Million Song dataset shows that higher prediction accuracy can be obtained with such an approach than when using, e.g., the above-mentioned BPR method. In our classification scheme, this method would be a hybrid that uses both additional information as well as popularity information.

"Collocated artists - greatest hits" (CAGH): In this paper, we do not only apply the previous scheme for the playlist generation problem, but propose an extension to it. Our assumption is that the different artists that are included in playlists by the users are not too different from each other. We thus propose to recommend tracks based on the frequency of the collocation of artists.

More precisely, we compute the similarity between two artists a and b according to the following formula:

$$\text{sim}_\text{a}(a,b) = \frac{\sum_p (\delta_{a,p} \cdot \delta_{b,p})}{\sqrt{\sum_p \delta_{a,p} \cdot \sum_p \delta_{b,p}}}$$

with $\delta_{a,p} = 1$ if playlist p contains a and 0 otherwise. The similarity thus depends on the collocations of artists within playlists, which can be computed offline.

Our proposed formula for the computation of the score of a next track t with artist a given a playlist beginning h is as follows:

$$\text{score}_\text{CAGH}(t,h) = \sum_{b \in A_h} \text{sim}_\text{a}(a,b) \cdot \text{counts}(t) \tag{4}$$

where A_h is the set of artist names of the tracks in h and $\text{counts}(t)$ is the number of occurrences of t in the dataset.

Evaluation Design and Experiments

In the following, we will describe our evaluation design for the playlist generation problem, describe the experimental design and present the results of the experiments.

Data Sets

We used three datasets in our experiments. The first one is the one provided by Artofthemix, the most commonly used dataset for related research (McFee and Lanckriet 2011; Hariri, Mobasher, and Burke 2012). The second was retrieved using the web service of last.fm[3]. The third was provided to us by 8tracks[4]. In order to reduce the sparsity of the data, we used the high-quality web service of Musicbrainz[5] to correct artist and track misspellings in the datasets. We also removed playlists of size 1. For the last.fm data, in order to reduce the long-tail we decided to select playlists such that long-tail tracks are used at least twice. The different properties of the resulting datasets are shown in Table 1.

	`last.fm`	`Aotm`	`8tracks`
Playlists	50,000	28,636	99,542
Users	47,603	–	51,743
Tracks	69,022	214,769	179,779
Avg. tracks/playlist	7.6	20.1	9.7
Avg. track usage count	5.5	2.7	5.3
Head	4.8%	1.6%	4.5%
Middle	35.0%	18.5%	25.7%
Tail	60.1%	79.9%	69.8%
Artists	11,788	47,473	29,352
Avg. artists/playlist	4.5	17.3	8.9
Avg. artist usage count	32.2	12.1	32.7

Table 1: Properties of the datasets.

Notice that the Artofthemix data does not contain user IDs. As implicitly done also by (Hariri, Mobasher, and Burke 2012), we consider users as being equivalent to playlists, as they usually do not create large numbers of playlists. This assumption is validated by the proportion of playlist per user on both other datasets (1.05 playlists per user on last.fm data and 1.9 on 8tracks data). Regarding track occurrences, the last.fm and 8tracks datasets have a similar average track usage count (5.5 and 5.3)[6]. This usage count is significantly smaller for Artofthemix (2.7). Another related characteristic is the long tail distribution of track usages. Table 1 divides the corresponding distribution into three parts: head, middle and tail. The "head" contains tracks which appeared more than 20 times in playlists, tracks in the "middle" were included in playlist between 2 and

[3]`http://www.lastfm.de/api`

[4]`http://8track.com`

[5]`http://musicbrainz.org/ws/2/`

[6]The track/artist usage count means how often a track/artist was used in all playlists

20 times, and songs from the "tail" were only used once or twice. These values are admittedly somewhat arbitrary but allow us to roughly compare the respective distributions. The resulting proportions reveal another difference between the last.fm and 8tracks datasets: although they have a similar average track usage count, the size of the long tail of 8tracks is much larger.

Regarding artist-based recommendation approaches, Table 1 shows that playlists usually contain fewer artists than tracks. A further analysis revealed that the artist of the last track of a playlist already appeared in the same playlist before in 31.1% of the cases for the last.fm dataset, in 21.8% of the cases for the Artofthemix dataset and 13.8% of the cases for the 8tracks dataset. This represents another difference between the last.fm and 8tracks datasets: although they have a similar average artist usage count, the artists seem to be better distributed across the playlists in the 8tracks data. Overall, these values represent a strong argument to emphasize on artist names as an additional information when recommending tracks.

Using three datasets with quite different characteristics should allow us to analyze how the different algorithms perform in different situations. In general, generating recommendations based on the Artofthemix dataset should be much more difficult than with the last.fm and 8tracks datasets, as it is smaller and the individual tracks are less often used. Other factors may however play a major role as well, in particular the size of the long tail.

Experiments

In our evaluation, we followed the evaluation design of (Hariri, Mobasher, and Burke 2012) and measured the accuracy of recommending the last track of each playlist in terms of hit rate. A 10-fold cross-validation procedure was applied on the three datasets. Recall that the total number of tracks of the dataset highly influences the hit rate values. (Hariri, Mobasher, and Burke 2012) provide the results for prediction lists of size varying between 1 and 300 given 21,783 tracks. This corresponds to the selection of about 1.5% of the tracks. We used a similar proportion in our experiments and set the maximum size of the prediction lists to 1,000 for last.fm, 3,000 for Artofthemix and 2,500 for 8tracks.

In the following sections, we will first present an evaluation of the frequent-pattern based approaches previously described using different configurations. The "winner" of this comparison is later used in a comparative evaluation of further techniques, including our new baseline scheme CAGH which is based on popularity and artist information.

Evaluation of Frequent Patterns Figure 1, 2 and 3 show the hit rates of association rules and sequential patterns on last.fm, Artofthemix and 8tracks data. The support and confidence threshold values correspond to the ones that led to the highest accuracy[7].

As expected, only modest accuracy values can be achieved for short recommendation lists. Recall however

[7]AR = Association Rules, SP = Sequential Patterns, n = max. size of patterns, w = window size for SP.

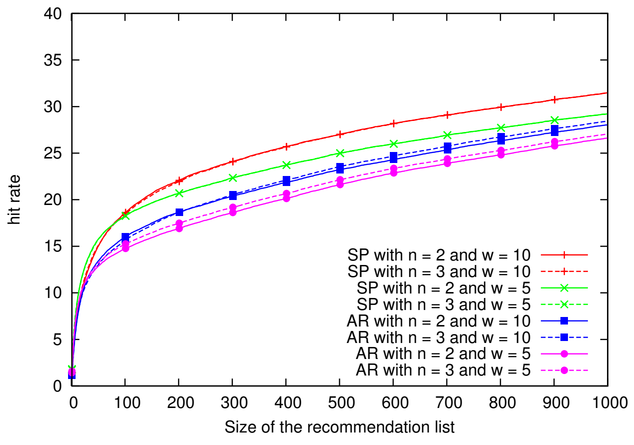

Figure 1: Hit rate for the last.fm dataset (69,022 tracks).

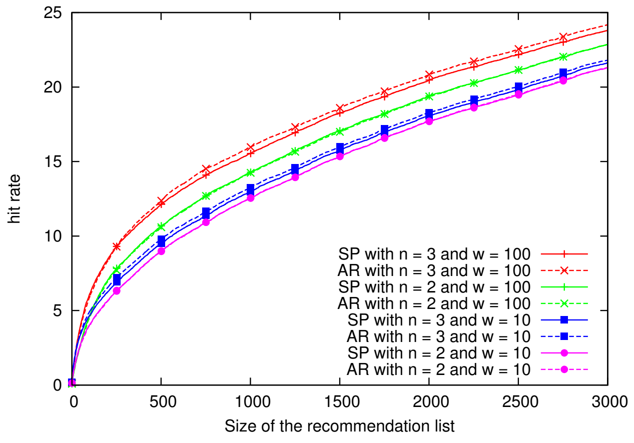

Figure 2: Hit rate for Artofthemix (214,769 tracks).

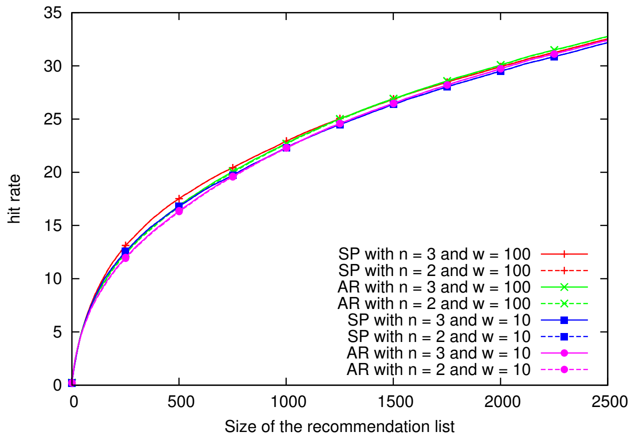

Figure 3: Hit rate for 8tracks (179,779 tracks).

that the hit rate only provides a lower bound on the accuracy. It is in general not possible to determine how many tracks are actually relevant, and as the datasets contain tens of thousands of tracks, it is possible that, for instance, the first 100 tracks presented are all relevant but do not contain the one the user actually chooses.

Moving to larger recommendation lists, we can identify four groups that lead to similar results for the last.fm and Artofthemix datasets. This phenomenon can also be observed on the 8tracks dataset, but the difference between the results are very small. The results for the last.fm dataset suggest that using longer patterns does not help to improve the accuracy as the results are the same for patterns of size 2 and 3, independent of the other parameters. Other aspects, however, have some influence: taking into account sequentiality information (SP) leads to better results, as well as using a sliding window of size 10 instead of a 5. A size of 100 however leads to lower accuracy values.

The results for the Artofthemix dataset however suggest a different conclusion. Taking into account sequentiality information or not does not make much difference. Using longer patterns and larger window sizes, however, helps to improve the accuracy on this dataset. Thus, only the influence of the size of the sliding window is consistent on the two datasets. This influence is also corroborated on the 8tracks dataset, although the difference is small. Our assumption however is that the size of the long tails of the Artofthemix and 8tracks datasets prevented us from successfully extracting representative sequential patterns which in turn led to the observed results. Under this assumption, sequentiality may in general have some importance and should be taken into account when recommending tracks for playlist generation.

In the next set of experiments we therefore used the following configurations: SP with $n = 2$ and $w = 10$ for the last.fm dataset, AR with $n = 3$ and $w = 100$ for the Artofthemix dataset and SP with $n = 3$ and $w = 100$ for the 8tracks dataset.

Comparing kNN, frequent patterns and other baselines Figure 4, 5 and 6 show the results of comparing five different recommendation approaches on the three datasets. The approaches include the three above-mentioned frequent-pattern approaches, a kNN recommender using 50 and 100 neighbors, the SAGH recommender ("Greatest hits of artists in playlist") and our new baseline recommender CAGH ("Greatest hits of collocated artists")[8].

As in the previous experiments, all approaches lead to comparably low accuracy values for short recommendation lists. For longer recommendation lists, our new CAGH recommender clearly outperforms the other approaches on last.fm and Artofthemix data, except for the frequent-patterns approach on the Artofthemix data for recommendation lists longer than $2,800$. On the data from 8tracks, the frequent pattern approach clearly outperforms all other approaches, followed by the kNN approach with 100 neighbors, and the CAGH recommender. One reason for the

[8]The method of (Hariri, Mobasher, and Burke 2012) is not included here but is comparable to the kNN method according to their measurements.

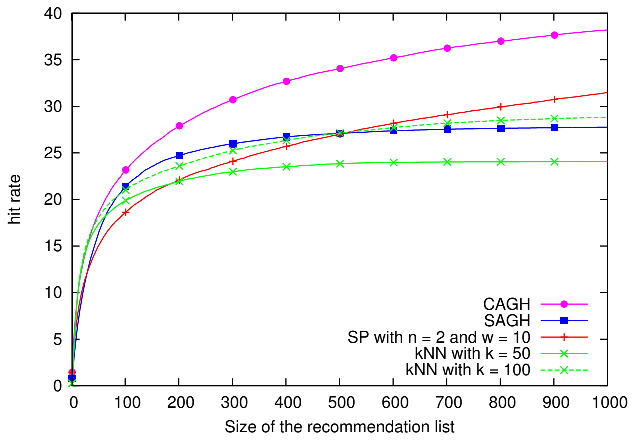

Figure 4: Hit rates on last.fm data ($69,022$ tracks).

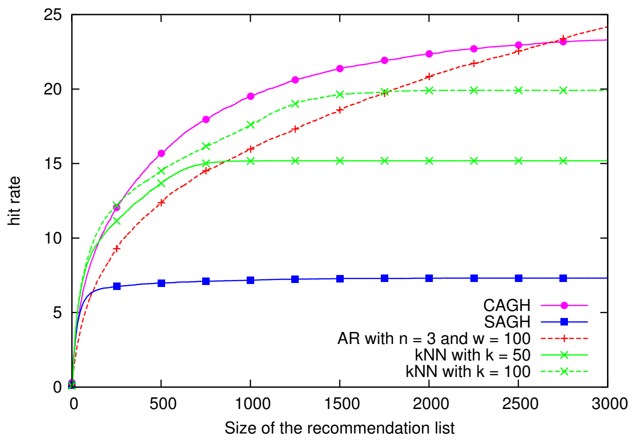

Figure 5: Hit rates on Artofthemix data ($214,769$ tracks).

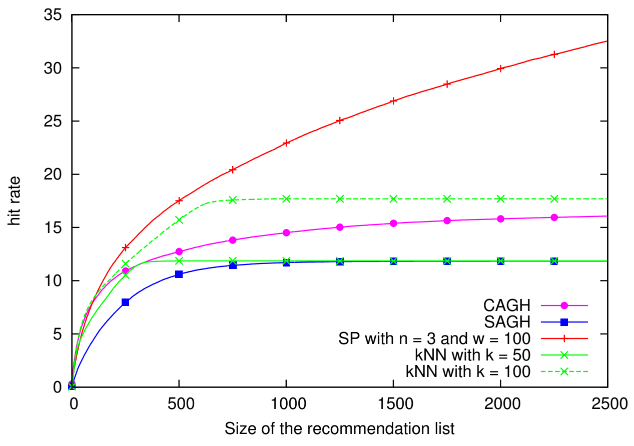

Figure 6: Hit rates on 8tracks data ($179,779$ tracks).

lower performance of the CAGH recommender could be that artists are more distributed across playlists on this particular dataset.

Overall, using more neighbors enhances the accuracy of the kNN approach on the three datasets. The kNN approach may even outperform all the other appraoches using more than 100 neighbors. However, both neighborhood sizes used in these experiments are already high and make the recommendation algorithm not only intractable in terms of space requirements, but also in terms of running time. Still, kNN approaches lead to lower accuracy values than both our new baseline approach and the frequent patterns method. More precisely, on the three data sets the accuracy of the kNN approach seems to be limited by the size of the recommendation lists it is able to build. This is probably the reason why the frequent patterns outperform this approach on the 8tracks dataset, as it is close to a kNN approach that uses all the neighbors.

Other observations depend on the used dataset. In particular, for the last.fm dataset, the SAGH recommender leads to results that are similar to those of the kNN recommender with 100 neighbors. For the Artofthemix dataset, the SAGH recommender is clearly outperformed by all other approaches. For the 8tracks dataset, it leads to results that are similar to those of the kNN recommender with 50 neighbors for recommendation lists longer than 750.

Beside the results shown in Figure 4, 5 and 6, we also experimented with models based on the Markov property, among them the simple bigram model and the recent Latent Markov Embedding (LME) model of (Chen et al. 2012). Despite the long time that can be required to train these models – e.g., several weeks for the LME model – these methods led to particularly low accuracy values which were consistently below 10% for recommendation lists of size $1,000$ for the last.fm dataset and 5% for recommendation lists of size $3,000$ for the Artofthemix dataset. We therefore omit these results in this paper. In general, given these comparably strong differences, assuming the Markov property might be too strong for this problem setting. Furthermore, our results indicate that emphasizing on artist names can be particularly promising for accurate track recommendation in the context of playlist generation.

Conclusion

This paper proposes a classification of existing approaches for playlist generation and discusses limitations of typical experimental designs, which for example do not take scalability aspects into account or are based on comparably strong assumptions such as the Markov property. Based on this discussion, we propose a new computationally efficient recommendation scheme based on popularity and artist information. An experimental comparative evaluation showed that our algorithm outperforms the other approaches in terms of accuracy on two of three different datasets. On the remaining dataset, our recommender is on a par with neighborhood-based approaches and was outperformed by a frequent pattern technique. Our hypothesis is that this difference is caused by the high dispersion of artists on this particular dataset. However, this difference in accuracy might be caused by other factors, which we are investigating in our current work.

Another set of experiments in this paper suggests that playlists seem to be at least partially governed by sequential constraints. In our future research we plan to investigate techniques that are able to identify criteria which can be used as indicators of the relevance of sequentiality information. Another perspective is the incorporation of other usually available additional information such as the playlist creator or the creation time. Some music platforms also provide detailed information about their users, independently of playlists. This information could be used to efficiently personalize the generation of playlists even for users who never created any before, which might be a very frequent situation.

Acknowledgments

We thank 8tracks for providing us their valuable data.

References

Agrawal, R., and Srikant, R. 1995. Mining Sequential Patterns. In *Proc. ICDE 1995*, 3–14.

Agrawal, R.; Imieliński, T.; and Swami, A. 1993. Mining Association Rules between Sets of Items in Large Databases. In *Proc. SIGMOD 1993*, 207–216.

Barrington, L.; Oda, R.; and Lanckriet, G. 2009. Smarter than Genius? Human Evaluation of Music Recommender Systems. In *Proc. ISMIR 2009*, 357–362.

Baur, D.; Boring, S.; and Butz, A. 2010. Rush: Repeated Recommendations on Mobile Devices. In *Proc. IUI 2010*, 91–100.

Celma, Ò. 2010. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer.

Chen, S.; Moore, J.; Turnbull, D.; and Joachims, T. 2012. Playlist Prediction via Metric Embedding. In *Proc. KDD 2012*, 714–722.

Cremonesi, P.; Koren, Y.; and Turrin, R. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. ACM RecSys 2010*, 39–46.

Cunningham, S.; Bainbridge, D.; and Falconer, A. 2006. 'More of an Art than a Science': Supporting the Creation of Playlists and Mixes. In *Proc. ISMIR 2006*, 240–245.

Fields, B. 2011. *"Contextualize Your Listening: The Playlist as Recommendation Engine"*. PhD thesis, Goldsmiths, University of London, London, UK.

Flexer, A.; Schnitzer, D.; Gasser, M.; and Widmer, G. 2008. Playlist Generation Using Start and End Songs. In *Proc. ISMIR 2008*, 173–178.

Hariri, N.; Mobasher, B.; and Burke, R. 2012. Context-Aware Music Recommendation Based on Latent Topic Sequential Patterns. In *Proc. ACM RecSys 2012*, 131–138.

Lamere, P., and Celma, Ò. 2011. Music Recommendation and Discovery Remastered, Tutorial at ACM RecSys 2011. Online at `http://www.slideshare.net/slideshow/embed_code/9860137`.

McFee, B., and Lanckriet, G. 2011. The Natural Language of Playlists. In *Proc. ISMIR 2011*.

McFee, B.; Bertin-Mahieux, T.; Ellis, D.; and Lanckriet, G. 2012. The million song dataset challenge. In *Proc. AdMIRe'12*.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. UAI*, 452–461.