A Comparison of Frequent Pattern Techniques and a Deep Learning Method for Session-Based Recommendation

Iman Kamehkhosh
TU Dortmund
iman.kamehkhosh@tu-dortmund.de

Dietmar Jannach
TU Dortmund
dietmar.jannach@tu-dortmund.de

Malte Ludewig
TU Dortmund
malte.ludewig@tu-dortmund.de

ABSTRACT

Making session-based recommendations, i.e., recommending items solely based on the users' last interactions without having access to their long-term preference profiles, is a challenging problem in various application fields of recommender systems. Using a coarse classification scheme, the proposed algorithmic approaches to this problem in the research literature can be categorized into *frequent pattern mining* algorithms and approaches that are based on *sequence modeling*. In the context of methods of the latter class, recent works suggest the application of recurrent neural networks (RNN) for the problem. However, the lack of established algorithmic baselines for session-based recommendation problems makes the assessment of such novel approaches difficult.

In this work, we therefore compare a state-of-the-art RNN-based approach with a number of (heuristics-based) frequent pattern mining methods both with respect to the accuracy of their recommendations and with respect to their computational complexity. The results obtained for a variety of different datasets show that in every single case a comparably simple frequent pattern method can be found that outperforms the recent RNN-based method. At the same time, the proposed much more simple methods are also computationally less expensive and can be applied within the narrow time constraints of online recommendation.

CCS CONCEPTS

•General and reference →Evaluation; •Information systems →Recommender systems; •Computing methodologies →*Neural networks; Rule learning;*

KEYWORDS

Session-Based Recommendations; Deep Learning; Frequent Pattern Mining; Benchmarking

1 INTRODUCTION

Making recommendations solely based on a user's current session and most recent interactions is a nontrivial problem for recommender systems. On an e-commerce website, for instance, when a visitor is new (or not logged in), there are no long-term user models that can be applied to determine suitable recommendations for this user. Furthermore, recent work shows that considering the user's short-term intent has often more effect on the accuracy of the recommendations than the choice of the method used to build the long-term user profiles [20]. In general, such types of problems are common on e-commerce sites, e.g., when returning users do not log in every time they use the site. The same challenges can, however, be observed also for other application domains, in particular for news and media (music and video) recommendation [21, 33].

The problem of predicting the next actions of users based solely on their sequence of actions in the current session is referred to in the literature as *session-based recommendation.* A number of algorithmic approaches have been proposed over the years to deal with the problem. Early academic approaches, for example, rely on the detection of sequential patterns in the session data of a larger user community. In principle, even simpler methods can be applied. Amazon's "Customers who bought … also bought" feature represents an example that relies on simple co-occurrence patterns to generate recommendations, in that case in the context of the very last user interaction (an item view event). A number of later works then explored the use of Markov models [30, 35, 39], and most recently, researchers explored the use of recurrent neural networks (RNN) for the session-based next-item recommendation problem [16, 17, 38, 42].

Today, RNNs can be considered one of the state-of-the-art methods for sequence learning tasks. They have been successfully explored for various sequence-based prediction problems in the past [5, 9, 11, 18] and in a recent work, Hidasi et al. [16] investigated an RNN variant based on gated recurrent units (GRU) for the session-based recommendations problem. In their work, they benchmarked their RNN-based method GRU4REC with different baseline methods on two datasets. Their results showed that GRU4REC is able to outperform the baseline approaches in terms of accuracy for top-20 recommendation lists.

While these results indicate that RNNs can be successfully applied for the given recommendation task, we argue that the experimental evaluation in [16] does not fully inform us about different aspects of the effectiveness and the practicability of the proposed method. First, regarding the effectiveness, it is unclear if the methods to which GRU4REC was compared are competitive. Second, as the evaluation was based on one single training-test split and only using accuracy measures, further investigations are necessary to assess, for example, if some algorithms exhibit certain biases, e.g., to recommend mostly popular items. Third, even if the RNN method is effective, questions regarding the scalability of the method should be discussed, in particular as hyper-parameter optimization for the complex networks can become very challenging in practice.

The goal of this work is to shed light on these questions and in the remainder of this paper we will report the detailed results of comparing a state-of-the-art RNN-based method with a number of computationally more efficient pattern mining approaches in different dimensions.

Workshop on Temporal Reasoning in Recommender Systems, collocated with ACM RecSys'17, Como, Italy.

2 PREVIOUS WORKS

In session-based recommendation problems, we are given a sequence of the most recent actions of a user and the goal is to find items that are relevant in the context of the user's specific short-term intent. One traditional way to determine recommendations given a set of recent items of interest is to apply frequent pattern mining techniques, e.g., based on association rules (AR) [1]. AR are often applied for market basket analysis with the goal to find sets of items that are bought together with some probability [14]. The order of the items or actions in a session is irrelevant for AR-based approaches. Sequential patterns mining (SP) [2] techniques, in contrast, consider the order of the elements in sessions when identifying frequent patterns. In one of the earlier works, Mobasher et al. [32] used frequent pattern mining methods to predict a user's next navigation action. In another work, Yap et al. [47] propose a sequential pattern-mining-based next-item recommendation framework, which weights the patterns according to their estimated relevance for the individual user. In the domain of music recommendation, Hariri et al. more recently [15] propose to mine sequential patterns of latent topics based on the tags attached to the tracks to predict the context of the next song.

A different way of finding item-to-item correlations is to look for sessions that are similar to the current one (*neighbors*), and to determine frequent item co-occurrence patterns that can be used in the prediction phase. Such neighborhood-based approaches were for example applied in the domains of e-commerce and music in [4] or [26]. In some cases and application domains, simple co-occurrence patterns are despite their simplicity quite effective, see, e.g., [20, 40] or [44].

Differently from such pattern- and co-occurrence-based techniques, a number of recent approaches are based on *sequence modeling* using, e.g., Markov models. The main assumption of Markov-model-based approaches in the context of session-based recommendation is that the selection of the next item in a session is dependent on a limited number of previous actions. Shani et al. [35] were among the first who applied first-order Markov chains (MC) for session-based recommendation and showed the superiority of sequential models over non-sequential ones. In the music domain, McFee and Lanckriet [30] proposed a music playlist generation algorithm based on MCs that – given a seed song – selects the next track from uniform and weighted distributions as well as from k-nearest neighbor graphs. Generally, a main issue when applying Markov chains in session-based recommendation is that the state space quickly becomes unmanageable when all possible sequences of user selections should be considered [12, 16].

More recent approaches to sequence modeling for session-based recommendation utilize recurrent neural networks (RNN). RNNs process sequential data one element at a time and are able to selectively pass information across sequence steps [28]. Zhang et al. [49], for example, successfully applied RNNs to predict advertisement clicks based on the users' browsing behavior in a sponsored search scenario. For session-based recommendations, Hidasi et al. [16] investigated a customized RNN variant based on gated recurrent units (GRU) [5] to model the users' transactions within sessions. They also tested several ranking loss functions in their solutions. Later on, in [17] and [42] RNN-based approaches were proposed which leverage additional item features to achieve higher accuracy. For the problem of news recommendation, Song et al. [36] proposed a temporal deep semantic structured model for the combination of long-term static and short-term temporal user preferences. They considered different levels of granularity in their model to process both fast and slow temporal changes in the users' preferences. In general, neural networks have been used for a number of recommendation-related tasks in recent years. Often, such networks are used to learn embeddings of content features in compact fixed-size latent vectors, e.g., for music, for images, for video data, for documents, or to represent the user [3, 6–8, 13, 25, 29, 46]. These representations are then integrated, e.g., in content-based approaches, in variations of latent factor models, or are part of new methods for computing recommendations [7, 8, 13, 27, 37, 43, 45].

In the work presented in this paper, we will compare different existing and novel pattern-mining-based approaches with a state-of-the-art RNN-based algorithm.

3 EXPERIMENT CONFIGURATIONS

3.1 Algorithms

3.1.1 RNN Baseline. GRU4REC is an RNN-based algorithm that uses Gated Recurrent Units to deal with the vanishing or exploding gradient problem proposed in [16]. In our experiments, we used the Python implementation that is shared by the authors online.[1]

3.1.2 Session-based kNN – KNN. The KNN method searches the k most similar past sessions ("neighbors") in the training data based on the set of items in the current session. Since the process of determining the neighbor sessions becomes very time-consuming as the number of sessions increases, we use an special in-memory index data structure (cache) in our implementation. Technically, in the training phase, we create a data structure that maps the training sessions to their set of items and one structure that maps the items to the sessions in which they appear. To make recommendations for the current session s, we first create a union of the sessions in which the items of s appear. This union will be the set of *possible* neighbors of the current session. This is a fast operation as it only involves a cache lookup and set operations. To further reduce the computational complexity of the prediction process, we select a subsample of these possible neighbors using a heuristic. In this work, we took the m most *recent* sessions as focusing on recent trends has shown to be effective for recommendations in e-commerce [23]. We then compute the similarity of these m most recent possible neighbors and the current session and select the k most similar sessions as the neighbor sessions of the current session. Again through lookup and set union operations, we create the set of *recommendable* items R that contains items that appear in one of the k sessions. For each recommendable item i in R, we then compute the KNN score as the sum of the similarity values of s and its neighbor sessions $n \in N_s$ which contains i (Equation 1). The indicator function $1_n(i)$ returns 1 if n contains i and 0 otherwise, see also [4].

$$score_{\text{KNN}}(i, s) = \Sigma_{n \in N_s} sim(s, n) \times 1_n(i) \quad (1)$$

In our experiments, we tested different distance measures to determine the similarity of sessions. The best results were achieved when the sessions were encoded as binary vectors of the item space

[1] https://github.com/hidasib/GRU4Rec

and when using cosine similarity. In our implementation, the set operations, similarity computations, and the final predictions can be done very efficiently as will be discussed later in Section 4.2.2. Our algorithm has only two parameters, the number of neighbors k and the number of sampled sessions m. For the large e-commerce dataset used in [16], the best parameters were, for example, achieved with $k = 500$ and $m = 1000$. Note that the kNN method used in [16] is based on item-to-item similarities while our kNN method aims to identify similar sessions.

3.1.3 kNN Temporal Extension – TKNN. The KNN method, when using cosine similarity as a distance measure, does not consider the temporal sequence of the events in a session. To be able to leverage the temporal information within the KNN technique, we designed an additional temporal-filtering heuristic for it. The proposed TKNN method uses the same scoring scheme as the KNN method (Equation 1). The only difference is that, given the current session s, we consider item i as being recommendable only if it appears in the neighbor session n directly after a certain item. In our implementation, that certain item is the last item of the current session s. Technically, we therefore use a slightly different implementation of the indicator function of Equation 1: $1_n(i) = 1$ if neighbor session n contains i and (j, i) is a subsequence of n, where j is the last item of the current session and thus the basis to predict the next item.

3.1.4 Simple Association Rules – AR. To assess the strength of simple two-element co-occurrence patterns, we included a method named AR which can be considered as an association rule technique with a maximum rule size of two. Technically, we create a rule $r_{p,q}$ for every two items p and q that appear together in the training sessions. We determine the *weight*, $w_{p,q}$, of each rule simply as the number of times p and q appear together in past sessions. Given the current session s, the AR score of a target item i will be then computed as

$$score_{\text{AR}}(i, s) = w_{i,j} \times 1_{AR}(r_{i,j}) \tag{2}$$

where j is the last item of the current session s for which we want to predict the successor and AR is the set of rules and their weights as determined based on the training data. The indicator function $1_{AR}(r_{i,j}) = 1$ when AR contains $r_{i,j}$ and 0 otherwise.

3.1.5 Simple Sequential Rules – SR. The SR method is a variant of AR, which aims to take the order of the events into account. Similar to the AR method, we create a sequential rule for the co-occurrence of every two items p and q as $r_{p,q}$ in the training data. This time, however, we consider the distance between p and q in the session when computing the weight of the rules. In our implementation, we use the multiplicative inverse as a weight function and set $w_{p,q} = 1/x$, where x is the number of items that appear between p and q in a session. Other heuristics such as a linear or a logarithmic function can also be used. In case that those two items appear together in another session in the training data, the weight of the rule in that session will be added to the current weight. We finally normalize the weight and divide it by the total number of sessions that contributed to the weight. Given the current session s, the SR score of a target item i is then computed as

$$score_{\text{SR}}(i, s) = w_{j,i} \times 1_{SR}(r_{j,i}) \tag{3}$$

Table 1: Dataset characteristics.

	RSC	TMall	#nowplaying	30Music	AotM	8tracks
Sessions	8M	4.6M	95K	170K	83K	500K
Events	32M	46M	1M	2.9M	1.2M	5.8M
Items	38K	620K	115K	450K	140K	600K
Avg. E/S	3.97	9.77	10.37	17.03	14.12	11.40
Avg. I/S	3.17	6.92	9.62	14.20	14.11	11.38

where j is the last item of session s and SR is the set of sequential rules. The indicator function $1_{SR}(r_{j,i}) = 1$ when SR contains $r_{j,i}$ and 0 otherwise.

3.1.6 Hybrid Approaches. We made additional experiments with several hybrids that combine different algorithms. At the end, a weighted combination of the two normalized prediction scores of the algorithms led to the best results in our experiments.

3.2 Datasets and Evaluation Protocol

We performed experiments using datasets from two different domains in which session-based recommendation is relevant, namely e-commerce and next-track music recommendation. The source code and the public datasets can be found online.[2]

3.2.1 E-commerce Datasets. For the e-commerce domain, we chose the *ACM RecSys 2015 Challenge* dataset (RSC) as used in [16]. The RSC dataset is a collection of sequences of click events in shopping sessions. The second e-commerce dataset is a public dataset published for the *TMall* competition. This dataset contains shopping logs of the users on the Tmall.com website.

3.2.2 Music Datasets. We used (a) two datasets that contain *listening logs* of several thousand users and (b) two datasets that comprise thousands of manually created *playlists*.

Listening logs: These used datasets are (almost one-year-long) sub-samples of two public datasets. First, we created a subset of the *#nowplaying* dataset [48], which contains music-related tweets on Twitter. Second, we used the recent *30Music* dataset [41], which contains listening sessions retrieved from Internet radio stations through the Last.fm API.

Playlists: Generally, music playlists are different in nature from listening logs and e-commerce user logs in various ways. Nonetheless, they are designed to be consumed in a listening session and the tracks are often arranged in a specific sequence. The used playlist datasets come from two different music platforms. The *Art-of-the-Mix* dataset (AotM) was published by [31] and contains playlists by music enthusiasts. The *8tracks* dataset was shared with us by the 8tracks platform. A particularity of the 8tracks dataset is that each public playlist can only contain two tracks per artist.

The dataset statistics are shown in Table 1. The total number of sessions is larger for the e-commerce datasets. However, the number of unique items in the music datasets, which corresponds to the number of tracks included in the playlists or the number of played tracks in the listening sessions, is higher than the number of items in e-commerce datasets.

[2] http://ls13-www.cs.tu-dortmund.de/homepage/rectemp2017

The music sessions are on average longer than the e-commerce sessions.[3] The last row of Table 1 shows the average number of unique items in each session ("Avg. I/S"). Comparing this value with the average session length ("Avg. E/S") indicates what we call the item *repetition rate* in each dataset. Including the same track more than once in a playlist is comparably uncommon. Listening to a track more than once during a listening session is, however, common. The difference between the average session length and the average number of items in each session for the e-commerce dataset indicates that re-occurring of the same item in a session is common in the e-commerce domain.

3.2.3 Evaluation Protocol. The general task of the session-based recommendation techniques in our experiment is to predict the next-item view event in a shopping session or to predict the next track that is played in a listening session or is included in a playlist. To evaluate the session-based algorithms, we use the same evaluation scheme as in [16]. We incrementally add events to the sessions in the test set and report the average hit rate (HR), which corresponds to recall in this evaluation setting, and the mean reciprocal rank (MRR), which takes the position of the hit into account. We tested list lengths of 1, 2, 3, 5, 7, 10, 15, and 20. While the experiments in [16] are done without cross-validation, we additionally apply a fivefold sliding-window validation protocol as in [24] to minimize the risk that the obtained results are specific to the single train-test split. We, therefore, created five train-test splits for each dataset. For the listening logs, we used 3 months of training data and the next 5 days as the test data and randomized splits for the playlists as they have no timestamps assigned.

4 RESULTS

4.1 Accuracy Results

Our first experiment used the exact same setup as described in [16], i.e., we use only one training-test split when comparing GRU4REC with our methods. As done in [16], we trained the algorithms using 6 months of data containing 7,966,257 sessions of 31,637,239 clicks on 37,483 items and tested them on the sessions of the next day.

In the subsequent sections, we then report the results of our comparison using the sliding-window validation scheme described above with recommendation list lengths varying from 1 to 20. In all experiments, we tuned the parameters for the different algorithms using grid search and optimized for HR@20 on validation sets (subsets of the training sets). GRU4REC was only optimized with 100 layers as done in [16] due to the computational complexity of the method. To test for statistical significance, we use Wilcoxon signed-rank test with $\alpha = 0.05$.

4.1.1 Results Using the Original Evaluation Setup. Table 2 shows the results ordered by the hit rate (HR@20) when using the original setup. We could reproduce the hit rate and MRR results from [16] (using their optimal parameters) for GRU4REC(1000,BPR) and GRU4REC(1000,TOP1), which use 1000 hidden units and the TOP1 and BPR's pairwise ranking loss functions, respectively. In Table 2, we additionally report the results for recommendation list length ten, which might be more important for different application domains.

[3]Note that each session in the TMall dataset is defined as the sequence of actions of a user during one day which results in relatively larger average session length.

Table 2: Results when using the evaluation scheme of [16].

Method	HR@10	MRR@10	HR@20	MRR@20
SR	**0.568**	**0.290**	**0.672**	**0.297**
TKNN	0.545	0.251	0.670	0.260
AR	0.543	0.273	0.655	0.280
KNN	0.521	0.242	0.641	0.250
GRU4REC(1000,BPR)	0.517	0.235	0.636	0.243
GRU4REC(1000,TOP1)	0.517	0.261	0.623	0.268

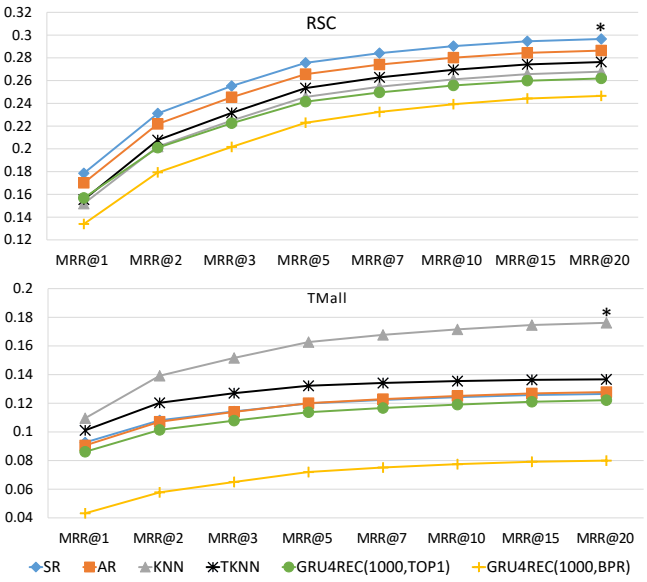

Figure 1: MRR results for the e-commerce datasets (* indicates statistical significance).

The best accuracy results were achieved by the SR method both for the hit rate and MRR and for both list lengths. In terms of the hit rate, every single frequent pattern method used in the experiment was better than the GRU4REC methods. A similar observation can be made also for the MRR, with the exception that the KNN-based methods consistently performed worse than the GRU4REC(1000,TOP1) method on this measure.

4.1.2 E-commerce Datasets. Figure 1 shows the MRR results for the algorithms on the two e-commerce datasets, RSC and TMall. For both datasets, we can observe that most of the frequent pattern methods lead to higher or at least similar MRR values as GRU4REC. There is, however, no clear "winner" across both datasets. The SR method works best for the RSC dataset. On the TMALL dataset, the KNN method outperforms the others, an effect which might be caused by the longer list session lengths for this dataset.[4] In both cases, however, the difference between the winning method and the best-performing GRU4REC configuration is statistically significant. This is indicated by a star symbol in Figure 1.

4.1.3 Listening Logs Datasets. Figure 2 shows the accuracy performance of the algorithms on two selected listening logs datasets.

[4]Remember that the sessions of the TMALL dataset cover the events of one day, as the time stamps in this dataset are given only in the granularity of days.

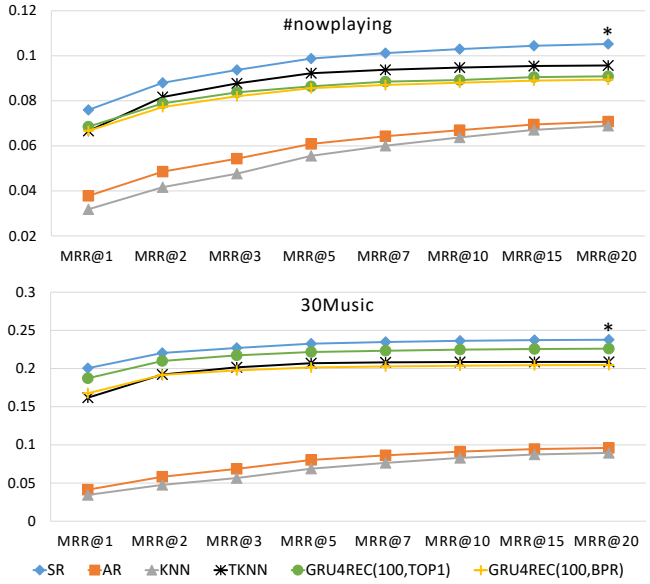

Figure 2: MRR results for the listening log datasets.

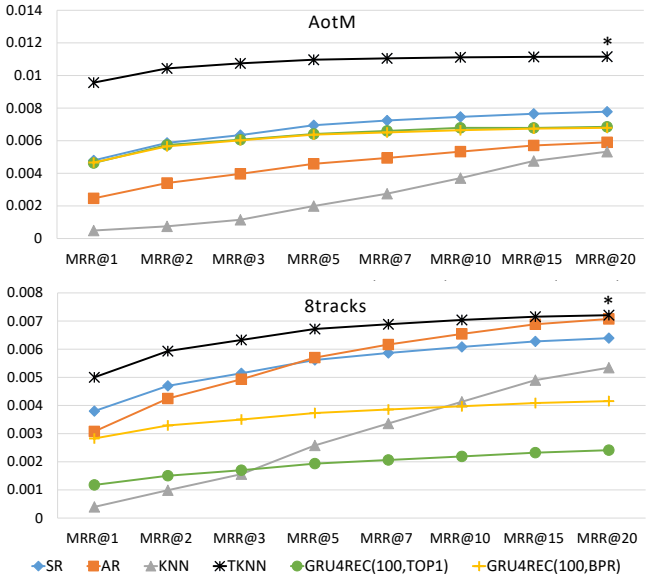

Figure 3: MRR results for the playlist datasets.

Similar to the e-commerce datasets, in all measurements, a frequent pattern approach, namely the SR method, outperforms GRU4REC. Here again, for MRR@20, the recommendations of SR are significantly more accurate than the recommendations of GRU4REC. Note that on the music datasets, we apply GRU4REC(100,TOP1) and GRU4REC(100,BPR), which use 100 hidden units and the TOP1 and BPR's pairwise ranking loss function, respectively.[5]

The TKNN method – the time-aware extension of KNN– works always significantly better than the KNN method on the listening logs datasets. TKNN also outperforms both GRU4REC configurations on the #nowplaying dataset for list lengths larger than 1.

Another observation on the listening logs datasets is that the sequence-based approaches (SR, TKNN and GRU4REC) work significantly better than methods that do not consider the temporal information in data (KNN and AR).

4.1.4 Playlists Datasets. Figure 3 shows the MRR results of the algorithms on the playlists datasets. On both datasets, the temporal extension of kNN, TKNN, leads to the best results across all recommendation list sizes and significantly outperforms both variants of GRU4REC. The performance of all other methods, however, seems to depend on the specifics of the datset. The SR method works well on both datasets. The relative performance of the AR method, however, depends on the dataset and the list length at which the measurement is made.

One interesting observation that we made for the music datasets is that the relative performance of KNN strongly improves in terms of the hit rate[6] when the recommendation list length is increased. This can, for example, be seen in Figure 4, which shows the hit rate results for the #nowplaying dataset. The hit rate of KNN on the #nowplaying dataset that is about 3% for list length one increases

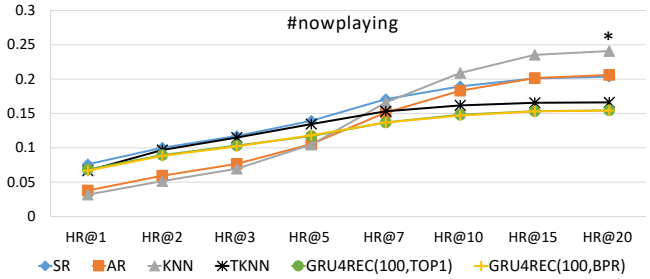

Figure 4: HR results for the #nowplaying dataset.

to 24% for list length 20. At the same time, the hit rate of some of the other methods only slightly increases, e.g., from 6% to 15%. As a result, across all four investigated music datasets, KNN outperforms all other algorithms in terms of HR@20. A similar trend can also be seen for AR, the other non-sequential approach.

4.1.5 Aggregated Ranking of Algorithms. To determine the ranking of different algorithms based on their accuracy results (MRR@20) across all six datasets, we applied the Borda Count (BC) rank aggregation strategy [10]. The results show that SR and TKNN are both ranked first (30 points), followed by AR as the second best algorithm (20 points). The GRU4REC method with TOP1 ranking loss is ranked third (18 points). Finally, KNN and GRU4REC with BPR ranking loss are ranked fourth (15 points) and fifth (13 points), respectively.

4.1.6 Hybrid Approaches. We conducted a variety of additional experiments with different hybridization methods as described in Section 3.1.6 to analyze the effect of combining the algorithms. In general, a weighted combination of the two normalized prediction scores of a neighborhood-based and a sequence-based method led to the best results in our experiments. For instance, the combination of KNN and SR with a weight ratio of 3 to 7, WH(KNN,SR:0.3,0.7), outperformed all other individual algorithms on the 30Music dataset.

[5] Repeating the experiments with 1000 hidden layers for the GRU4REC methods did not lead to any better results on the music datasets.

[6] Generally, the hit rate results for the experiments, which we do not include here for space reasons, are similar to the MRR results.

Table 3: Results of the hybrid methods for 30Music.

Method	HR@5	MRR@5	HR@20	MRR@20
SR	0.285	0.233	0.332	0.238
KNN	0.142	0.069	0.342	0.089
GRU	0.275	0.222	0.315	0.226
WH(KNN,SR:0.3,0.7)	**0.298**	**0.243**	0.386	**0.252**
WH(KNN,GRU:0.6,0.4)	0.261	0.144	**0.396**	0.159

Another example is combining the normalized score of KNN and GRU4REC(100,TOP1), which can outperform other algorithms in terms of HR@20. The differences between the winning hybrid approaches (printed in bold face in Table 3) and the best performing individual methods in each measurement were statistically significant. Similar results were also achieved for the other datasets, which we do not include here for space reasons.

4.2 Additional Analyses

Since prediction accuracy might not be the only possible relevant quality criterion in a domain [19], we made a number of additional analyses as shown in Figure 5.

4.2.1 Popularity Bias and Catalog Coverage. As in [22], we first measured the average popularity of the top-20 recommendations of the algorithms to assess possible recommendation biases. The popularity of an item is computed based on its number of occurrences in the training dataset. Overall, the recommendations of non-sequential approaches (KNN and AR) shows the highest bias towards popular items. The sequence-based approaches (SR and GRU4REC), in contrast, recommend comparably less popular items.

Additionally, we analyzed the catalog coverage of each algorithm by counting the number of different items that appear in the top-20 recommendation lists of all sessions in the test set. Overall, the recommendation lists of GRU4REC and SR include more different items than the other algorithms. The recommendations of neighborhood methods, KNN and TKNN, on the other hand, focus on smaller sets of items and show a higher concentration bias. This can be explained by considering the sampling strategy of KNN which focuses on a smaller subset of the sessions, e.g., those of the last few days.

4.2.2 Computational Complexity and Memory Usage. We measured the training time as well as the needed memory and time to generate recommendations for each algorithm. On a desktop computer with an Intel i7-4790k processor, training GRU4REC on one split of the RSC dataset with almost 4 million sessions and in its best configuration takes more than 12 hours. This time can be reduced to 4 hours when calculations are performed by the GPU (Nvidia GeForce GTX 960).[7] The KNN method needs about 27 seconds to build the needed in-memory maps, see Section 3.1.2. The well-performing SR method needs about 48 seconds to determine the rule weights. A specific advantage of the latter two methods is that they support incremental updates, i.e., new events can be immediately incorporated into the algorithms. Creating one recommendation list with GRU4REC needed, on average, about 12 ms. KNN needs about 26 ms for this task and SR only 3 ms.

[7] Training the model for 6 month of data using the GPU lasts about 8 hours.

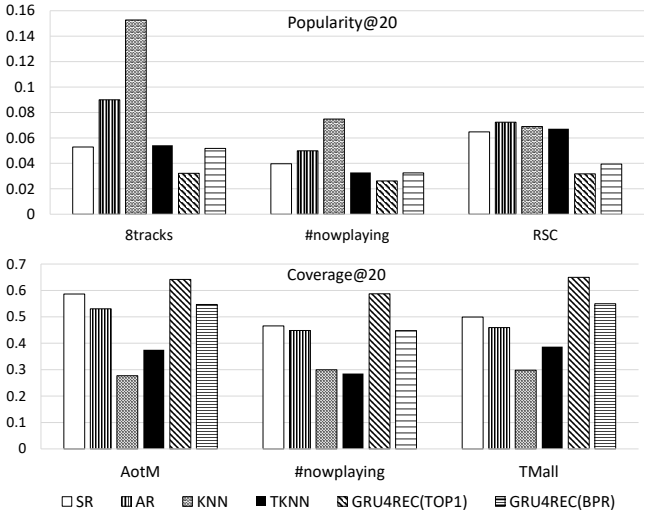

Figure 5: Popularity biases and catalog coverages of the algorithms on three sample datasets.

The raw data used for training the algorithms in this specific experiment (one split of the RSC dataset) occupies about 540 MB of main memory. The data structures used for training SR and KNN occupy about 50 MB and 3.2 GB, respectively. The model created by GRU4REC needs about 510 MB. Note that memory demand of GRU4REC depends on the algorithm parameters and significantly increases with the number of items. For the music and Tmall datasets, the memory demand of GRU4REC exceeded the capacity of our graphics card. Running GRU4REC using the CPU is multiple times slower than when a graphics card is used.

5 CONCLUSION AND FUTURE WORKS

Our work indicates that comparably simple frequent-pattern-based approaches can represent a comparably strong baseline when evaluating session-based recommendation problems. At the end, we could find at least one pattern-based approach that was significantly better than a recent RNN-based method. In particular the SR method was surprisingly effective, despite the fact that both learning and applying the rules is very fast.

Our results also indicates that the "winning" strategy seems to strongly depend on the characteristics of the data sets like average session lengths or repetition rates. Further research is still required to understand this relationship. In our future work, we will investigate the performance of additional session-based algorithms. These algorithms include both ones that are based on Markov models, e.g., Rendle et al.'s factorized Markov chains [34], as well as recently proposed improvements to GRU4REC, e.g., by Tan et al. [38]. We expect that continuously improved RNN-based methods will be able to outperform the frequent pattern based baselines used in the evaluation reported in this paper. These methods can, however, be computationally quite expensive. From a practical perspective, one has therefore to assess depending on the application domain if the obtained gains in accuracy justify the usage of these complex models, which cannot be easily updated online and whose predictions can be difficult to explain.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD '93*. 207–216.

[2] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining Sequential Patterns. In *ICDE '95*. 3–14.

[3] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the GRU: Multi-task Learning for Deep Text Recommendations. In *RecSys '16*. 107–114.

[4] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *ACM Computing Surveys* 47, 2 (2014), 26:1–26:35.

[5] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014).

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys '16*. 191–198.

[7] Sander Dieleman. 2016. Deep Learning for Audio-Based Music Recommendation. In *DLRS '16 Workshop*. 1–1.

[8] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. In *WWW '15*. 278–288.

[9] Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science* 14, 2 (1990), 179 – 211.

[10] Peter Emerson. 2013. The Original Borda Count and Partial Voting. *Social Choice and Welfare* 40, 2 (2013), 353–358.

[11] Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *CoRR* abs/1308.0850 (2013). http://arxiv.org/abs/1308.0850

[12] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *CoRR* abs/1410.5401 (2014).

[13] Yupeng Gu, Bo Zhao, David Hardtke, and Yizhou Sun. 2016. Learning Global Term Weights for Content-based Recommender Systems. In *WWW '16*. 391–400.

[14] Jiawei Han and Micheline Kamber. 2006. *Data Mining: Concepts and Techniques (Second Edition)*. Morgan Kaufmann.

[15] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-Aware Music Recommendation Based on Latent Topic Sequential Patterns. In *RecSys '12*. 131–138.

[16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based Recommendations with Recurrent Neural Networks. *CoRR* abs/1511.06939 (2015).

[17] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In *RecSys '16*. 241–248.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.

[19] Dietmar Jannach and Gedas Adomavicius. 2016. Recommendations with a Purpose. In *RecSys '16*. 7–10.

[20] Dietmar Jannach, Lukas Lerche, and Michael Jugovac. 2015. Adaptation and Evaluation of Recommendations for Short-term Shopping Goals. In *RecSys '15*. 211–218.

[21] Dietmar Jannach, Lukas Lerche, and Iman Kamehkhosh. 2015. Beyond "Hitting the Hits": Generating Coherent Music Playlist Continuations with the Right Tracks. In *RecSys '15*. 187–194.

[22] Dietmar Jannach, Lukas Lerche, Iman Kamehkhosh, and Michael Jugovac. 2015. What recommenders recommend: an analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction* (2015), 1–65.

[23] Dietmar Jannach and Malte Ludewig. 2017. Determining Characteristics of Successful Recommendations from Log Data – A Case Study. In *SAC '17*.

[24] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *RecSys 2017*. (forthcoming).

[25] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. 2016. Convolutional Matrix Factorization for Document Context-Aware Recommendation. In *RecSys '16*. 233–240.

[26] Lukas Lerche, Dietmar Jannach, and Malte Ludewig. 2016. On the Value of Reminders within E-Commerce Recommendations. In *UMAP '16*. 27–25.

[27] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep Collaborative Filtering via Marginalized Denoising Auto-encoder. In *CIKM '15*. 811–820.

[28] Zachary Chase Lipton. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *CoRR* abs/1506.00019 (2015).

[29] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *SIGIR '15*. 43–52.

[30] Brian McFee and Gert R. G. Lanckriet. 2011. The Natural Language of Playlists. In *ISMIR '11*. 537–542.

[31] Brian McFee and Gert R. G. Lanckriet. 2012. Hypergraph Models of Playlist Dialects. In *ISMIR '12*. 343–348.

[32] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. 2002. Using Sequential and Non-Sequential Patterns in Predictive Web Usage Mining Tasks. In *ICDM '02*. 669–672.

[33] Ozlem Ozgobek, Jon A. Gulla, and Riza C. Erdur. 2014. A Survey on Challenges and Methods in News Recommendation. In *WEBIST '14*. 278–285.

[34] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing Personalized Markov Chains for Next-basket Recommendation. In *WWW '10*. 811–820.

[35] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-Based Recommender System. *The Journal of Machine Learning Research* 6 (Dec. 2005), 1265–1295.

[36] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-Rate Deep Learning for Temporal Recommendation. In *SIGIR '16*. 909–912.

[37] Florian Strub, Romaric Gaudel, and Jérémie Mary. 2016. Hybrid Recommender System based on Autoencoders. In *DLRS '16 Workshop*. 11–16.

[38] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS '16)*. ACM, 17–22.

[39] Maryam Tavakol and Ulf Brefeld. 2014. Factored MDPs for Detecting Topics of User Sessions. In *RecSys '14*. 33–40.

[40] Roberto Turrin, Andrea Condorelli, Paolo Cremonesi, Roberto Pagano, and Massimo Quadrana. 2015. Large Scale Music Recommendation. In *LSRS 2015 Workshop at ACM RecSys*.

[41] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music Listening and Playlists Dataset. In *Poster Proceedings RecSys '15*.

[42] Bart lomiej Twardowski. 2016. Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks. In *RecSys '16*. 273–276.

[43] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation. In *RecSys '16*. 225–232.

[44] Koen Verstrepen and Bart Goethals. 2014. Unifying Nearest Neighbors Collaborative Filtering. In *RecSys '14*. 177–184.

[45] Jeroen B. P. Vuurens, Martha Larson, and Arjen P. de Vries. 2016. Exploring Deep Space: Learning Personalized Ranking in a Semantic Space. In *DLRS '16 Workshop*. 23–28.

[46] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *KDD '15*. 1235–1244.

[47] Ghim-Eng Yap, Xiao-Li Li, and Philip S. Yu. 2012. Effective Next-items Recommendation via Personalized Sequential Pattern Mining. In *DASFAA'12*. Berlin, Heidelberg, 48–64.

[48] Eva Zangerle, Martin Pichl, Wolfgang Gassler, and Günther Specht. 2014. #Nowplaying Music Dataset: Extracting Listening Behavior from Twitter. In *WISMM '14 Workshop at MM '14*. 21–26.

[49] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. 2014. Sequential Click Prediction for Sponsored Search with Recurrent Neural Networks. In *AAAI '14*. 1369–1375.