Evaluation of Session-based Recommendation Algorithms

MALTE LUDEWIG, TU Dortmund, Germany
DIETMAR JANNACH, AAU Klagenfurt, Austria

Recommender systems help users find relevant items of interest, for example on e-commerce or media streaming sites. Most academic research is concerned with approaches that personalize the recommendations according to long-term user profiles. In many real-world applications, however, such long-term profiles often do not exist and recommendations therefore have to be made solely based on the observed behavior of a user during an ongoing session. Given the high practical relevance of the problem, an increased interest in this problem can be observed in recent years, leading to a number of proposals for *session-based recommendation algorithms* that typically aim to predict the user's immediate next actions.

In this work, we present the results of an in-depth performance comparison of a number of such algorithms, using a variety of datasets and evaluation measures. Our comparison includes the most recent approaches based on recurrent neural networks like GRU4REC, factorized Markov model approaches such as FISM or FOSSIL, as well as more simple methods based, e.g., on nearest neighbor schemes. Our experiments reveal that algorithms of this latter class, despite their sometimes almost trivial nature, often perform equally well or significantly better than today's more complex approaches based on deep neural networks. Our results therefore suggest that there is substantial room for improvement regarding the development of more sophisticated session-based recommendation algorithms.

CCS Concepts: • **Information systems → Recommender systems**; • **General and reference → Evaluation**;

Additional Key Words and Phrases: Sequential Recommendation; Session-based Recommendation; Deep Learning; Factorized Markov Models, Nearest-Neighbors

1 INTRODUCTION

Many of today's online services use recommender systems to point their users or site visitors to additional items that might be of interest to them. In academic research, the majority of works is focusing on techniques that rely on long-term preference models to determine the items to be presented to the user. However, in many application domains of recommender systems, such long-term user models are often not available for a larger fraction of the users, e.g., because they are first-time visitors or because they are not logged in. Consequently, suitable recommendations have to be determined based on other types of information, usually the user's most recent interactions with the site or application. Recommendation techniques that rely solely on the user's actions in an ongoing session and which adapt their recommendations to the user's actions are called *session-based* recommendation approaches.

Amazon's "Customers who bought … also bought" recommendations can be considered an extreme case of such a session-based approach. In this case, the recommendations are seemingly only dependent on the item that is currently viewed by the user (and the purchasing patterns of the community). A number of other techniques were proposed in the research literature, which do not limit themselves to the very last action, but consider some or all user actions since the session started. Some of these techniques only consider which events happened; others, in contrast, in addition take the sequence of events into account in their algorithms. Besides the e-commerce domain, a number of other application fields were in the focus in the literature, among them in particular music, web page navigation, or travel and tourism.

Authors' addresses: Malte Ludewig, TU Dortmund, Otto-Hahn-Straße 12, Dortmund, 44227, Germany, malte.ludewig@tu-dortmund.de; Dietmar Jannach, AAU Klagenfurt, Universitätsstraße 65-67, Klagenfurt am Wörthersee, 9020, Austria, dietmar.jannach@aau.at.

In academia, sequential recommendation problems are typically operationalized as the task of predicting the next user action. Experimental evaluations are usually based on larger, time-ordered logs of user actions, e.g., on the users' item viewing and purchase activities on an e-commerce shop or on their listening history on a music streaming site. From an algorithmic perspective, early approaches to predict the next user actions were based, for example, on sequential pattern mining techniques. Later on, different types of more sophisticated methods based on Markov models were proposed and successfully applied to the problem. Finally, in the most recent years, the use of deep learning approaches based on artificial neural networks was explored as another solution. Recurrent Neural Networks (RNN), which are capable of learning models from sequentially ordered data, are a "natural choice" for this problem, and significant advances regarding the prediction accuracy of such algorithms were reported in the recent literature [9, 20–22, 52].

Despite the growing number of papers on the topic in recent years, no true "standard" benchmark data sets or evaluation protocols exist in the community. Therefore, it remains difficult to compare the various algorithmic proposals, in particular as often different baseline algorithms are used in the papers. And, for some of them it is also unclear if they are particularly strong. In our previous work [27, 30], we could, for example, demonstrate that a comparably simple k-nearest-neighbor method leads to similar or even better accuracy results than a modern deep learning approach.

To establish a common base for future research, we performed an in-depth performance comparison across multiple domains and datasets, which involved a number of comparably simple as well as more sophisticated algorithms from the recent literature. Our results show that computationally and conceptually simple methods often lead to predictions that are similarly accurate or even better than those of today's most recent techniques based on deep learning models. As a consequence, we argue that researchers should take these more simple methods as alternative baselines into account when developing novel session-based recommendation algorithms. Furthermore, our results suggest that there is still substantial room for improvement regarding the development of more sophisticated session-based recommendation algorithms.

The paper is organized as follows. Next, in Section 2, we discuss previous works and typical application areas of session-based recommendation approaches. In Section 3, we provide technical details about the algorithms that were compared in our work. Section 4 describes our evaluation setup and Section 5 the outcomes of our experiments. To foster reproducible research on the topic, we share the code of the used evaluation framework and the compared algorithms online.[1]

2 REVIEW OF SESSION-BASED RECOMMENDATION APPROACHES

Most of the approaches for session-based recommendation proposed in the literature implement some form of *sequence learning*, see also [41] for a recent survey on the more general class of sequence-aware recommenders. Early approaches were based on the identification of *frequent sequential patterns*, which can be used at recommendation time to predict a user's next action. These early approaches were applied, for example, in the context of predicting the online navigation behavior of users [38]. Later on, such pattern mining techniques were also used for next-item recommendation problems in e-commerce or the music domain [3, 16, 59].

While frequent pattern techniques are easy to implement and lead to interpretable models, the mining process can be computationally demanding. At the same time, finding good algorithm parameters, in particular a suitable minimum support threshold, can be challenging. Finally, in some application domains it seems that using frequent item sequences does not lead to better recommendations than when using more simple item co-occurrence patterns [3]. In the context of

[1] https://www.dropbox.com/sh/7qdquluflk032ot/AACoz2Go49q1mTpXYGe0gaANa?dl=0

this work, we investigate both sequential patterns and co-occurrence patterns in their simplest forms as baselines.

In many newer works, more sophisticated sequence learning approaches were proposed that implement some form of *sequence modeling*. Such sequence modeling approaches are usually based on Markov Chain (MC) models [14, 18, 23, 36], reinforcement learning (RL) and Markov Decision Processes (MDP) [39, 46, 53], or Recurrent Neural Networks (RNN) [11, 21, 22, 35, 47, 49, 50, 55, 60, 63]. Again, the typical application scenarios of these methods include the e-commerce and the music recommendation domain.

An early approach based on an MDP model was proposed in [46]. It demonstrated the value of using sequential data in an e-commerce scenario, but also showed that models based on Markov Chains often cannot be directly applied due to data sparsity. Therefore, Shani et al. [46] proposed different heuristics to overcome the problem. An additional challenge when using this type of models lies in the choice of the order of the models, i.e., how many of the previous interactions should be considered when predicting the next one. Some authors therefore use a mixture of Variable-order Markov Models (VMMs) or *context-trees* to consider sequences of different lengths [14, 18]. Other works like [23] rely on Hidden Markov Models (HMMs) to overcome certain limitations of plain Markov Chain models. In [39, 46], reinforcement learning was implemented based on MDPs, which made it possible to also consider the reward for the shop in the recommendation process. To deal with the problem of the explosion of the state space in such scenarios, Tavakol and Brefeld [53] proposed to model the state space based on the sequence of item attributes in order to predict the characteristics of the next item that the user will consider. In the context of the comparative analysis presented in this paper, we limit ourselves to a simple MC-based method as a baseline, in particular because some techniques like the one discussed in [53] require the existence of knowledge about certain item attributes.

The most recent works on sequence modeling are based on RNNs. Zhang et al. [63], for example, used them for the prediction of user clicks in an advertisement scenario. Hidasi et al. [21] were among the first to explore Gated Recurrent Units (GRUs) as a special form of RNNs for the prediction of the next user action in a session. Their method called GRU4REC was later on extended in different ways in [20, 22] and [42]. While Hidasi et al. in [21] reported substantial performance improvements over an *item-based* k-nearest-neighbor (kNN) method when using their first version of GRU4REC, our previous work [27] showed that a *session-based* nearest neighbor method also leads to competitive accuracy results for the same problem setting. Since GRU4REC was substantially improved since its initial version, we include the latest version of their method [20] in the performance comparison reported in this paper. Furthermore, given our observations regarding the often competitive performance of conceptually more simple methods we designed a number of variations of the basic session-based nearest neighborhood method from [27], which we also considered in the experiments.

Another family of sequence modeling approaches relies on *distributed item representations*, e.g., in the form of latent Markov embeddings [4, 5, 13, 58] or distributional embeddings [2, 10, 15, 43, 51, 56, 64]. Embeddings are dense, lower-dimensional representations that are derived from sequentially ordered data and encode transition probabilities based on the observations in the original data. They were applied, for example, in the domains of next-track music recommendation [4, 64], recommendation of learning courses [43], or next point-of-interest (POI) recommendation [13]. However, a general challenge when using item embeddings is that they can be computationally demanding and sometimes require substantial amounts of training data to be effective. In the context of our work, we experimented with item embeddings as an alternative representation of the user sessions. However, the usage of embeddings did not lead to an improvement in terms of the prediction accuracy for our problem settings, which is why we do not report the detailed outcomes of these experiments in this paper.

To overcome the limitations of pure sequence learning methods, a number of *hybrid methods* were proposed that, for instance, combine the advantages of matrix factorization techniques with sequence modeling approaches in the form of Factorized Markov Chains [6, 17, 19, 33, 45]. Rendle et al. [45] proposed the Factorized Personalized Markov Chain (FPMC) approach as an early method for next-item recommendations in e-commerce settings, where user interactions are represented as a three-dimensional tensor (user, current item, next-item). Later on, variations of FPMC were proposed and successfully applied for a variety of application problems [19, 29]. Other hybrid techniques that, for example, use some form of clustering or Latent Dirichlet Allocation in combination with a sequential recommendation method were proposed, e.g., in [16, 40, 48], for the problems of next-track or next-app recommendation. In our experimental evaluation, we include both Rendle et al.'s FPMC method [45] as well as the recent variations and improvements described in [29] (FISM) and [19] (FOSSIL).

3 DETAILS OF THE INVESTIGATED METHODS

Based on these discussion, we include the following four types of techniques in our comparison of session-based recommendation algorithms: baseline methods, nearest-neighbor techniques, recurrent neural networks, and (hybrid) factorization-based methods. The main input to all methods is a training set of past user sessions, where each session consists of a set of sequentially ordered actions of a given type, e.g., an item view event in an online shop or a consumption event on a media streaming site. The models learned by the algorithms can then be used to predict the next event in a given user session in the test set.

3.1 Baseline Methods

We include the following baseline techniques in our comparison: a method that we call Simple Association Rules (AR), first-order Markov Chains (MC), and a method that we named Sequential Rules (SR). All baselines implement very simple prediction schemes and have low computational complexity both for training and recommending. Furthermore, we include a prediction method based on Bayesian Personalized Ranking (BPR-MF) [44] as an alternative baseline.

3.1.1 Simple Association Rules (AR). Simple Association Rules (AR) are a simplified version of the association rule mining technique with a maximum rule size of two. The method is designed to capture the frequency of two co-occurring events, e.g., "Customers who bought … also bought". Algorithmically, the rules and their corresponding importance are "learned" by counting how often the items i and j occurred together in a session of any user.

Let a session s be a chronologically ordered tuple of item click events $s = (s_1, s_2, s_3, \dots, s_m)$ and S_p the set of all past sessions. Given a user's current session s with $s_{|s|}$ being the last item in s, we can define the score for a recommendable item i as follows, where the indicator function $1_{\text{EQ}}(a, b)$ is 1 in case a and b refer to the same item and 0 otherwise.

$$score_{\text{AR}}(i, s) = \frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|} 1_{\text{EQ}}(s_{|s|}, p_x) \cdot (|p| - 1)} \sum_{p \in S_p} \sum_{x=1}^{|p|} \sum_{y=1}^{|p|} 1_{\text{EQ}}(s_{|s|}, p_x) \cdot 1_{\text{EQ}}(i, p_y) \qquad (1)$$

In Equation 1, the sums at the right-hand side represent the counting scheme. The term at the left-hand side normalizes the score by the number of total rule occurrences originating from the current item $s_{|s|}$. A list of recommendations returned by the AR method then contains the items with the highest scores in descending order. No minimum support or confidence thresholds are applied. In our implementation, as shared online, we create the rules in one iteration over the training data and store them (sorted by weight) in nested maps to support fast lookups in the recommendation phase. With this data structure, top-n recommendations can be created almost instantaneously.

3.1.2 Markov Chains (MC). The MC baseline can be seen as a variant of AR with a focus on sequences in the data. Here, the rules are extracted from a first-order Markov Chain, which describes the transition probability between two *subsequent* events in a session. In our baseline approach, we simply count how often users viewed item q immediately after viewing item p. Technically, the score for an item i given the current session s with the last event $s_{|s|}$ can be defined as a simplified version of Equation 1:

$$score_{\mathrm{MC}}(i,s) = \frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|-1} 1_{\mathrm{EQ}}(s_{|s|}, p_x)} \sum_{p \in S_p} \sum_{x=1}^{|p|-1} 1_{\mathrm{EQ}}(s_{|s|}, p_x) \cdot 1_{\mathrm{EQ}}(i, p_{x+1}) \tag{2}$$

where the function $1_{\mathrm{EQ}}(a,b)$ again indicates whether a and b refer to the same item or not. Here, with the right-hand side of the formula, we count how often item i appears immediately after $s_{|s|}$. The normalization term transforms the absolute count into a relative transition probability. In line with AR, in our implementation the rules and weights are recorded in nested maps in one single iteration over the training data to ensure short training times and to support the fast generation of the recommendations.

3.1.3 Sequential Rules (SR). Finally, the SR method is a variation of MC or AR respectively. It also takes the order of actions into account, but in a less restrictive manner. In contrast to the MC method, we create a rule when an item q appeared after an item p in a session even when other events happened between p and q.

When assigning weights to the rules, we consider the number of elements appearing between p and q in the session. Specifically, we use the weight function $w_{\mathrm{SR}}(x) = 1/(x)$, where x corresponds to the number of steps between the two items.[2] Given the current session s, the SR method calculates the score for the target item i as follows:

$$score_{\mathrm{SR}}(i,s) = \frac{1}{\sum_{p \in S_p} \sum_{x=2}^{|p|} 1_{\mathrm{EQ}}(s_{|s|}, p_x) \cdot x} \sum_{p \in S_p} \sum_{x=2}^{|p|} \sum_{y=1}^{x-1} 1_{\mathrm{EQ}}(s_{|s|}, p_y) \cdot 1_{\mathrm{EQ}}(i, p_x) \cdot w_{\mathrm{SR}}(x-y) \tag{3}$$

In contrast to Equation 1 for AR, the third inner sum only considers indices of previous item view events for each session p. In addition, the weighting function $w_{\mathrm{SR}}(x)$ is added. Again, we normalize the absolute score by the total number of rule occurrences for the current item $s_{|s|}$. As for AR and MC, the algorithm was implemented using nested sorted maps, which can be created in a single iteration over the training data.

3.1.4 Bayesian Personalized Ranking (BPR-MF). To make our results comparable with previous research, we finally include a prediction method based on BPR-MF as a baseline in our experiments.[3] BPR-MF [44] is a learning-to-rank method designed for implicit-feedback recommendation scenarios. The method is usually applied for matrix-completion problem formulations based on longer-term user-item interactions. To apply the method for the session-based recommendation scenario—where there are no long-term user profiles—we attribute each session in the training set to a different user, i.e., each session corresponds to a user in the user-item interaction matrix. At prediction time, we use the average of the latent item vectors of the current session so far as the user vector.

[2] Other weighting functions, e.g., with a logarithmic decay, are possible as well. Using the linear function however led to the best results, on average, in our experiments.

[3] The method was proposed by Hidasi et al. in the context of the GRU4REC method, see https://github.com/hidasib/GRU4Rec.

3.2 Nearest Neighbors

Despite their simplicity, nearest-neighbor-based approaches often perform surprisingly well as discussed, e.g., in [57] and our previous work [27, 30]. We, therefore, include different nearest neighbor schemes in our comparison. First, we consider a more traditional item-based variant, which was also employed as a baseline method in [21]. Furthermore, we evaluate three variations of a more recent session-based nearest neighbor technique in our experiments.

3.2.1 Item-based kNN (IKNN). The IKNN method only considers the last element in a given session and then returns those items as recommendations that are most similar to it in terms of their co-occurrence in other sessions. Technically, each item is encoded as a binary vector, where each element corresponds to a session and is set to "1" in case the item appeared in the session. The similarity of two items can then be determined, e.g., using the cosine similarity measure, and the number of neighbours k is implicitly defined by the desired recommendation list length.

Conceptually, the method implements a certain form of a "Customers who bought … also bought" scheme like the AR baseline. The use of the cosine similarity metric however makes it less susceptible to popularity biases. Although item-to-item approaches are comparably simple, they are commonly used in practice and sometimes considered a strong baselines [8, 34]. In terms of the technical implementation, all similarity values can be pre-computed and sorted in the training process to ensure fast responses at recommendation time.[4]

3.2.2 Session-based kNN (SKNN). Instead of considering only the last event in the current session, the SKNN method compares the entire current session with the past sessions in the training data to determine the items to be recommended, see also [3, 16, 32]. Technically, given a session s, we first determine the k most similar past sessions (neighbors) N_s by applying a suitable session similarity measure, e.g., the Jaccard index or cosine similarity on binary vectors over the item space [3]. In our experiments, the binary cosine similarity measure led to the best results. As in [27], using $k = 500$ as the number of neighbors to consider led to good performance results for many datasets. Next, given the current session s, its neighbors N_s, and the chosen similarity function $sim(s_1, s_2)$ for two sessions s_1 and s_2, the recommendation score for each item i can be calculated as follows [3]:

$$score_{\text{SKNN}}(i, s) = \Sigma_{n \in N_s} sim(s, n) \cdot 1_n(i) \tag{4}$$

Here, the indicator function $1_n(i)$ returns 1 if session n contains item i and 0 otherwise.

Scalability Considerations. Given a current session s, we cannot scan a potentially large set of past sessions for possible neighbors in an online recommendation scenario. Therefore, in our implementation of the algorithm, as described in [27] in more detail, we rely on pre-computed in-memory index data structures and on neighborhood sampling to enable fast recommendation responses. The index is used to quickly locate past sessions that contain a certain item, i.e., the index allows us to retrieve *possible* neighbor sessions that contain at least one element of the current session through fast lookup operations. On the other hand, sampling only a smaller fraction of all past sessions in our experiments as potential neighbors has shown to lead to comparably small accuracy compromises. In fact, in some domains like e-commerce, only looking for neighbors in the most recent sessions—thereby capturing recent trends in the community—proved to be very effective [28] and led to even better results than when all past sessions were taken into account.

Our nearest neighbor implementations, therefore, have an additional parameter m, which determines the size of the sample. In the experiments reported in [27], it was, for example, sufficient to consider only the 1,000 most recent sessions from several million existing sessions.

[4]We use the implementation published at https://github.com/hidasib/GRU4Rec.

Sequence-Aware Extensions: V-SKNN, S-SKNN, and SF-SKNN. The described SKNN method does not consider the order of the elements in a session when using the Jaccard index or cosine similarity as a distance measure. Since the order of the elements might, however, be relevant in some domains and since the user preferences might change within a single session depending on the already seen items, we propose three variations of the SKNN method.[5]

- Vector Multiplication Session-Based kNN (V-SKNN): The idea of this variant is to put more emphasis on the more recent events of a session when computing the similarities. Instead of encoding a session as a *binary* vector as described above, we use real-valued vectors to encode the current session. Only the very last element of the session obtains a value of "1"; the weights of the other elements are determined using a linear decay function that depends on the position of the element within the session, where elements appearing earlier in the session obtain a lower weight. As a result, when using the *dot product* as a similarity function between the current weight-encoded session and a binary-encoded past session, more emphasis is given to elements that appear later in the sessions.
- Sequential Session-based kNN (S-SKNN): This variant also puts more weight on elements that appear later in the session. This time, however, we achieve the effect with the following scoring function:
$$score_{\text{S-SKNN}}(i,s) = \Sigma_{n \in N_s} sim(s,n) \cdot w_n(i,s) \tag{5}$$
Here, the indicator function $1_n(i)$ is replaced by a weighting function $w_n(i,s)$, which takes the order of the events in the current session s into account. The weight $w_n(i,s)$ increases when the more recent items of the current session s also appeared in a neighboring session n. If an item s_x is the most recent item of the current session s that also appears in the neighbor session n, then the weight will be defined as $w_n(i,s) = x/|s|$, where the index x indicates the position of s_x within the session. If, for example, the second-to-last item of the current session with a length of 5 is the most recent item also included in the neighbor session n, the weight would be $w_n(i,s) = 4/5$. Items from this neighbor can, therefore, potentially obtain a higher score than, e.g., items from neighbor sessions that only include the third from last item of the current session, which are assigned a weight of $3/5$.
- Sequential Filter Session-based kNN (SF-SKNN): This method also uses a modified scoring function, but in a more restrictive way. The basic idea is that given the last event (and related item $s_{|s|}$) of the current session s, we only consider items for recommendation that appeared directly after $s_{|s|}$ in the training data at least once.
$$score_{\text{SF-SKNN}}(i,s) = \Sigma_{n \in N_s} sim(s,n) \cdot 1_n(s_{|s|}, i) \tag{6}$$
While the general scoring function is identical to the one of SKNN (Equation 4), we use a different implementation of the indicator function $1_n(s_{|s|}, i)$. Here, 1 is only returned if there exists any past session which contains the sequence $(s_{|s|}, i)$, given $s_{|s|}$ is the item currently viewed in the user's current session s. Though the sequence $(s_{|s|}, i)$ can be part of any past session, the item i obviously still has to be a part of the neighbor session n for the indicator function to return 1.

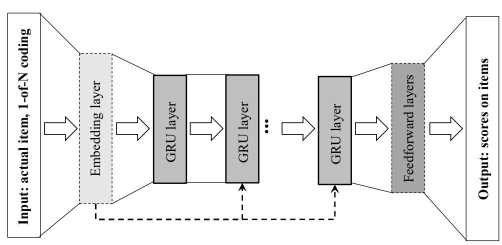
[5] We made additional experiments using other ways of encoding sequential information, e.g., by using embeddings of sessions and items with the popular *Word2Vec* and *Doc2Vec* approaches. However, none of these variations led to better accuracy results than the SKNN method in our experiments. We therefore omit these results from our later discussions.

3.3 Neural Networks – GRU4REC

Approaches based on Recurrent Neural Networks (RNNs), as discussed in Section 2, represent the most recently explored family of techniques for session-based recommendation problems. Among these methods, GRU4REC is one of the latest deep learning approaches that was specifically designed for session-based recommendation scenarios [20, 21].

GRU4REC models user sessions with the help of an RNN with Gated Recurrent Units [7] in order to predict the probability of the subsequent events (e.g., item clicks) given a session beginning. Figure 1 shows the general architecture of the network, in which the embedding, the feedforward, and additional GRU layers are optional. In fact, the authors of the method found that a single GRU layer of varying depth led to the best performance in their experiments. While the usage of RNNs for session-based, or more generally, sequential prediction problems is a natural choice, the particular network architecture, the choice of the loss functions, and the use of session-parallel mini-batches to speed up the training phase are key innovative elements of the approach.

Fig. 1. Architecture of the GRU4REC neural network, adapted from [21].



The model can be trained with stochastic gradient descent (SGD) using established optimizations like *ADAM*, *ADADELTA*, *RMSProp*, or *ADAGRAD* [12, 31, 62]. Furthermore, a number of hyper-parameters can be tuned, including, the learning rate, the layer sizes, a momentum factor, and a drop-out factor to stabilize the network.

The choice of the loss function is key to the quality of the recommendations of GRU4REC. The following loss functions were designed or applied by the authors. In particular the latest of their functions (*MAX*) [20] led to a significant performance improvement over the previous ones.

- *BPR*: Bayesian Personalized Ranking (BPR) [44] is a learning-to-rank algorithm, which uses a pairwise ranking loss function for the task of creating top-n recommendations. In GRU4REC, a generalized version of this function is applied using the following formula:

$$L_s = -\frac{1}{|S_N|} \cdot \sum_{j \in S_N} log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j})) \tag{7}$$

  In the loss function, the predicted rating $\hat{r}_{s,i}$ for the actual next item i given the current session s is compared to a set of negative samples S_N with the goal of maximizing the difference between them. Here, the sigmoid and logarithm functions are applied to represent the proportion.
- *TOP1*: This loss function was introduced by the authors of GRU4REC and can be seen as a regularized approximation of the relative rank of a positive sample $\hat{r}_{s,i}$ and the negative samples S_N:

$$L_s = \frac{1}{|S_N|} \cdot \sum_{j \in S_N} \sigma(\hat{r}_{s,j} - \hat{r}_{s,i}) + \sigma(\hat{r}^2_{s,j}) \tag{8}$$

Here, the proportion is approximated with the sigmoid function, and the regularization term $\sigma(\hat{r}^2_{s,j})$ is added so that the score of the negative samples is directed to zero.

- *MAX*: In continuation of their work, the authors proposed a generic extension to theses two loss functions.

$$L_{max}(\hat{r}_{s,i}, \{\hat{r}_{s,j}\}_{j \in S_N}) = L_s(\hat{r}_{s,i}, \max_{j \in S_N} \hat{r}_{s,j}) \tag{9}$$

Instead of using a sum of differences between the positive item's rating $\hat{r}_{s,i}$ and the negative samples S_N, only the highest rated negative sample $\max_{j \in S_N} \hat{r}_{s,j}$ from S_N is used to calculate the loss. As this function has to be differentiable for SGD training, $\max_{j \in S_N}$ is approximated with the *softmax* function. The resulting functions BPR_{max} and $TOP1_{max}$ showed superior performance when compared to the *BPR* and *TOP1* functions [20].

In our experiments, we used the GRU4REC (v2.0) implementation that the authors shared online. The code is regularly maintained by the authors and includes the implementation of the GRU4REC method, the code of their baseline algorithms, as well as the code for the evaluation procedure proposed in [21].

3.4 Factorization-based Methods

As described in Section 2, a number of (hybrid) factorization-based methods were proposed in recent years for *sequential* recommendation problems. We include three existing methods from the literature in our experiments, Factorized Personalized Markov Chains (FPMC) [45], FISM [29], and FOSSIL [19]. Generally, these methods aim at predicting the next actions of users, but were not designed for session-based recommendation scenarios with anonymous users. We therefore describe for each method how we applied them to our problem setting. In addition, we propose a novel factorization method called Session-based Matrix Factorization (SMF), which relies on the BPR_{max} and $TOP1_{max}$ loss functions as described above.

3.4.1 Factorized Personalized Markov Chains (FPMC). The FPMC method was designed for the specific problem of next-basket recommendation. The problem consists of predicting the contents of the next basket of a user, given his or her history of past shopping baskets. By limiting the basket size to one item and by considering the current session as the history of baskets, the method can be directly applied for session-based recommendation problems.

Technically, FPMC combines MC and traditional user-item matrix factorization in a three dimensional tensor factorization approach. As illustrated in Figure 2, the third dimension captures the transition probabilities from one item to another.

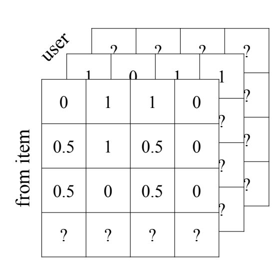


Fig. 2. Personalized transition cube, adapted from [44].

Internally, a special form of the Canonical Tensor Decomposition is used to factor the cube into latent matrices, which can then be used to predict a ranking in the following way:

$$\hat{r}_{u,l,i} = \langle v_u^{U,I}, v_i^{I,U} \rangle + \langle v_i^{I,L}, v_l^{L,I} \rangle + \langle v_u^{U,L}, v_l^{L,U} \rangle \tag{10}$$

where $\hat{r}_{u,l,i}$ is a score for item i with the preferences of user u when he previously examined item l. The three dimensional decomposition results in six latent matrices $v^{X,Y}$ representing the latent factor for dimension X regarding dimension Y, e.g., $v^{U,L}$ are the user latent factors in terms of the previously examined item and $v^{I,L}$ the item latent factors regarding the previously examined item. Those latent factors are learned using SGD with the pairwise ranking loss function BPR.

In our problem setting, where we have no long-term user histories, each session in the training data corresponds to a user. Once the model is trained, each new session therefore represents a user cold-start situation. To apply the model to our setting, we estimate the session latent vectors as the average of the latent factors of the individual items in the session. This approach was adopted also by [21] to apply BPR-MF to session-based recommendation scenarios.

3.4.2 Factored Item Similarity Models (FISM). This method is based on an item-item factorization, which has the advantage of being directly applicable to our session-based cold-start scenario, where no explicit user representation can be learned. However, FISM does not incorporate sequential item-to-item transitions like FPMC does. Equation 11 shows the prediction function, which in [29] is trained using SGD to predict ratings, e.g., for the movie domain.

$$\hat{r}_{u,i} = b_u + b_i + (n_u^+)^{-\alpha} \sum_{j \in R_u^+} p_j q_i^T \tag{11}$$

Technically, for user u and item i, a score $\hat{r}_{u,i}$ is calculated as the sum of latent vector products $p_j q_i^T$ between item i and the items R_u^+ already rated by the user u. In our scenario, R_u^+ corresponds to the previously inspected items in a session. The terms b_u and b_i are bias terms and n_u^+ specifies the number of ratings by user u, which is combined with a parameter α to normalize the sum of vector products to a certain degree. Instead of using the *RMSE* as an error metric, we use *BPR*'s pairwise loss function when optimizing the top-n recommendations for the given implicit feedback scenario.

3.4.3 Factorized Sequential Prediction with Item Similarity Models (FOSSIL). In this approach, FISM is combined with factorized Markov chains to incorporate sequential information into the model. The model can be described as shown in Equation 12 (from [19]):

$$\hat{r}_{u,l,i} = \underbrace{\sum_{j \in R_u^+ \setminus \{i\}} p_j q_i^T}_{\text{long-term preferences}} + \overbrace{(w + w_u)}^{\text{personalized weighting}} \cdot \underbrace{n_l m_i^T}_{\text{sequential dynamics}} \tag{12}$$

Again, $\hat{r}_{u,l,i}$ represents a rating for item i given a user u and his or her previously inspected item l. The first term represents the long-term user preferences and corresponds to the FISM model in Equation 11. Using a weighted sum with a global factor w and a personalized factor w_u, the model is extended by a factorized Markov chain to capture the sequential dynamics. In the last term of Equation 12, a latent vector n_l for item l is multiplied with a latent vector m_i for item i to factor in the user-independent probability of item l being followed by item i.

In our scenario, again, the sessions represent the users, R_u corresponds to the current session and *BPR* is used as the loss function to rank suitable items over negative examples.

3.4.4 Session-based Matrix Factorization (SMF). Finally, SMF is a novel factorization-based model that we designed for the specific task of session-based recommendation. Similar to FOSSIL it combines factorized Markov chains with classic matrix factorization. In addition, our method considers the cold-start situation of session-based recommendation scenarios as follows.

In contrast to the traditional factorization-based prediction model $r_{u,i} = p_u q_i^T$, in the SMF method, we replace the latent user vector p_u with a session preference vector s_e, which is computed as an embedding of the current session s:

$$s_e = M_{ST} \cdot s^T \tag{13}$$

Here, the session s is as a binary vector similar to the representation in SKNN (see Section 3.2.2) and M_{ST} is a transformation matrix of size $|I| \cdot |u_s|$, which reduces the size of the binary session vector (number of unique items $|I|$) to a specific latent vector size $|s_e|$.

Based on the embedded session representation s_e, the prediction function is defined as shown in Equation 14.

$$\hat{r}_{s,l,i} = w_i \cdot (\underbrace{s_e q_i^T + b_{1,i}}_{\text{session preferences}}) + (1 - w_i) \cdot (\overbrace{n_l m_i^T + b_{2,i}}^{\text{sequential dynamics}}) \tag{14}$$

The score $\hat{r}_{s,l,i}$ for a session s with the most recent item l and an item i is computed as a weighted combination of session preferences and sequential dynamics. Here, the session preferences correspond to the long-term user preferences in the traditional matrix factorization model, i.e., the embedded session latent vector s_e for the current session s is multiplied with an item latent vector q_i for item i to compute a relevance score i regarding s. The sequential dynamics are captured exactly as in Equation 12 for FOSSIL using latent representations for the currently inspected item l and item i. Both partial scores are adjusted with a separate bias term $b_{x,i}$ and combined in a weighted sum with the factor w_i dependent on item i.

To train this model, we incorporated some of the concepts from GRU4REC (see Section 3.3). Specifically, we adopted *ADAGRAD* for SGD-based optimization, and used BPR_{max} and $TOP1_{max}$ as loss functions. Furthermore, we integrated two additional concepts (and corresponding hyper-parameters) in the training phase to avoid model over-fitting: a session drop-out factor and a skip-rate. For a drop-out factor of 0.1, for example, each positive entry of the binary session input vector is set to 0 with a probability of 10%. The skip-rate, in contrast, refers to the positive sample that should be used in a training step. For example, with a skip-rate of 0.1, in 10% of the cases not the next event represents the positive sample but the subsequent one.

4 EXPERIMENT SETUP

In this section, we describe the details of our algorithm comparison in terms of to the used evaluation protocol, the performance measures, and the evaluation datasets. All source code and pointers to the public datasets are provided online to ensure reproducibility of our research.[6]

4.1 Evaluation Protocol and Performance Measures

The general computational task in session-based recommendation problems is to generate a ranked list of objects that in some form "matches" a given session beginning. What represents a good match, depends on the specific application scenario. It could be a set of alternative shopping items in an e-commerce scenario or a continuation of given music listening session.

In offline evaluations for session-based recommendations, researchers often abstract from the underlying purpose of the system [24], e.g., if the recommender should help discover something

[6] https://www.dropbox.com/sh/7qdquluflk032ot/AACoz2Go49q1mTpXYGe0gaANa?dl=0

new or find alternatives to a currently inspected item. Instead, the recorded user sessions are typically considered as a "gold standard" for the evaluation. To measure the performance of an algorithm, researchers resort to assessing the capability of an algorithm to predict the withheld entries of a session.

Different approaches are found in the literature to withhold certain entries of a session. In some works, only the last element is hidden [3, 16], some propose to "reveal" the first n elements of a session [25], while others, finally, evaluate their approaches by iteratively revealing one entry of a session after the other [22].

Selection of the Target Item and Accuracy Measures. In order to establish comparability with existing research, we report the results of two ways of measuring prediction accuracy.

- We use an evaluation scheme in which the task is to predict the *immediate next item* given the first n elements. For each session, we iteratively increment n, measure the hit rate (HR) and the Mean Reciprocal Rank (MRR), and finally determine the average HR and MRR for all sessions for the different list lengths, as done in [22].
- Alternatively, instead of focusing only on the next item, we made a measurement where we considered *all subsequent* elements for the given session beginning, because all of them might be relevant to the user. In this scheme, we used the standard information retrieval measures Precision and Recall at defined list lengths. The number of given elements of the session is also iteratively incremented as in the previously described evaluation scheme.

Training and Test Splits, Repeated Subsampling. Hidasi et al. [22] used one single training-test split. In the case of an e-commerce dataset, the data was split in a way that the sessions of all six months expect those of the very last day of the entire dataset were placed in the training set. The last day was used for testing. We report the results of applying this evaluation scheme to ensure comparability, e.g., with respect to the e-commerce dataset that was used in their experiments.

Since such single-split setups have their limitations, we focus our discussion on the results that were obtained when applying a *sliding-window* protocol, where we split the data into several slices of equal size. For the e-commerce data, for example, we can use the data of one month for training and the subsequent data (e.g., of one day) for testing. We then evaluate the performance for each of these data samples and report the average of the performance results for all slices. This latter protocol helps us reduce the danger that the observed outcomes are the results of one particular train-test configuration.[7]

Additional Quality Factors. Since accuracy is not the only relevant quality factor in practice, we made the following additional measurements, as was done in [27].

- *Coverage:* We report how many different items ever appear in the top-k recommendations. This measure represents a form of catalog coverage, which is sometimes referred to as *aggregate diversity* [1].
- *Popularity bias:* High accuracy values can, depending on the specific measurement method, correlate with the tendency of an algorithm to recommend mostly popular items [26]. To assess the popularity tendencies of the tested algorithms, we report the average popularity score for the elements of the top-k recommendations of each algorithm. The individual item popularity is determined as the min-max normalized count of item occurrences.

[7] To ensure that the smaller size of those splits does not negatively affect the performance of the model-based approaches, we tested the single-split configurations as well on all datasets. The obtained results are mostly in line with those obtained with the sliding-window protocol and shown in Appendix C.

- *Cold start:* Some methods might only be effective when a significant amount of training data is available. We, therefore, report the results of measurements where we artificially removed parts of the (older) training data to simulate such situations.
- *Scalability:* Training modern machine learning methods can be computationally challenging, and obtaining good results may furthermore require extensive parameter tuning. We, therefore, report the times that the algorithms needed to train the models and to make predictions at runtime. In addition, we report the memory requirements of the algorithms.

Parameter Optimization. Some of the algorithms that we tested require extensive (hyper-)parameter tuning including SMF and GRU4REC. Thus, we systematically optimized the parameters for those algorithms for each dataset. Due to the computational complexity of the methods, we restricted the layer size for GRU4REC as well as the number of latent factors for SMF to 100 and used a randomized search method with 100 iterations for the remaining parameters as described in [20]. In each iteration, the learning rate, the drop-out factor, the momentum, and the loss function were determined randomly in order to find the maximum hit rate for a list length of 20. All optimizations were performed on special validation splits, which were created by splitting a training set into a validation training and test set. For the more simple S-KNN-based approaches, we used the same validation sets to manually adjust the number of neighbors and samples when applying cosine similarity as the distance measure (except for V-SKNN). The final parameters for each method and dataset are provided in Appendix A.

4.2 Datasets

We made measurements for datasets from three different domains: e-commerce, music, and news.

E-Commerce Datasets. We used the following four e-commerce datasets.

- *RSC15.* This is one of the datasets which was used by Hidasi et al. in [21] and their later works. It was published in the context of the ACM RecSys 2015 Challenge and contains recorded click sequences (item views, purchases) for a period of six months. We use the label *RSC15-S* to denote the dataset and measurement where only one single train-test split is used.
- *TMALL.* This dataset was published in the context of the TMall competition and contains interaction logs of the tmall.com website for one year.
- *RETAILROCKET.* The e-commerce personalization company *retailrocket* published this dataset covering six month of user browsing activities, also in the context of a competition.
- *ZALANDO.* The final dataset is non-public and was shared with us by the fashion retailer Zalando. It contains user logs of their shopping platform for a period of one year. In our evaluation, we only considered the item *view* events as was done for the other e-commerce datasets.

Table 1 shows an overview of the characteristics of the e-commerce datasets. Except for the *RSC15-S* dataset, which we include to make our evaluation comparable with previous works [20, 27], we report the average values after creating five data splits as described above.

Media Datasets: Music and News. As in [27], we use the music domain as an alternative area to evaluate session-based recommendation algorithms, because music is commonly consumed within listening sessions in sequential order. We use the same datasets that were used in [27], which consist of two sets of *listening logs* and two datasets of user-created *playlists*. In addition, we made measurements using a dataset from the news domain.

- *8TRACKS and AOTM:* These dataset include playlists created by music enthusiasts. The *AOTM* dataset was collected from the Art-of-the-Mix platform and is publicly available [37]. The non-public *8TRACKS)* dataset was shared with us by the 8tracks.com music platform.

Table 1. Characteristics of the e-commerce datasets. The values are averaged over all five non-overlapping splits for each dataset, except for *RSC15-S*, where we only use one train-test split.

Dataset	RSC15-S	RSC15	TMALL	RETAILROCKET	ZALANDO
Actions	31,708,461	5,426,961	13,418,695	212,182	4,536,950
Sessions	7,981,581	1,375,128	1,774,729	59,962	365,126
Items	37,483	28,582	425,348	31,968	189,328
Timespan in Days	182	31	90	27	90
Actions per Session	3.97	3.95	7.56	3.54	12.43
Unique Items per Session	3.17	3.17	5.56	2.56	8.39
Actions per Day	174,222.31	175,063.26	149,096.61	7,858.59	50,410.56
Sessions per Day	43,854.84	44,358.97	19,719.22	2220.84	4056.96

- *30MUSIC and NOWPLAYING:* The *30MUSIC* dataset contains listening histories of the last.fm music platform and was published in [54]. The *NOWPLAYING* dataset was created from music-related tweets, where users posted which tracks they were currently listening [61].
- *CLEF:* The dataset was made available to participants of the 2017 CLEF NewsREEL challenge.[8] It consists of a stream of user actions (e.g., article reads) and article publication events, which were collected by the company *plista* for several publishers. In our evaluation we only considered the article read events. We used the data of the publisher with the largest amount of recorded interactions (the popular German sports news portal Sport1[9]).

The statistics for the datasets from the media (music and news) domain are given in Table 2.

Table 2. Characteristics of the music and news datasets. The values are again averaged over all five non-overlapping splits.

	8TRACKS	30MUSIC	AOTM	NOWPLAYING	CLEF
Actions	1,499,645	638,933	306,830	271,177	5,540,486
Sessions	132,453	37,333	21,888	27,005	1,644,442
Items	376,422	210,633	91,166	75,169	742
Timespan in Days	90	90	90	90	6
Actions per Session	11.32	17.11	14.02	10.04	3.37
Items per Session	11.31	14.47	14.01	9.38	3.17
Actions per Day	16,662.72	7099,26	3,409.22	3,013.08	923,414
Sessions per Day	1,471.70	414.81	243.20	300.06	274,074

5 RESULTS

5.1 E-Commerce Datasets

Table 3 shows the MRR and Hit Rate results at list length 20 for the four tested e-commerce datasets. In addition, we report the results when applying the standard measures Precision and Recall when considering *all* hidden elements in the rest of the session as described above (see Table 4). Finally, we also report coverage and popularity statistics for each algorithm.

[8]http://www.clef-newsreel.org/
[9]https://www.sport1.de/

Table 3. Hit rate (HR), Mean reciprocal rank (MRR), catalog coverage (COV), and the average popularity (POP) for a list length of 20 obtained for the e-commerce datasets (Table 3a to 3e). The table rows are ordered by MRR@20. The highest values are highlighted in each column and, in case of accuracy measures, marked with a star when the difference w.r.t. to the second-best performing method was statistically significant.

(a) RSC15

Metrics	MRR@20	HR@20	COV@20	POP@20
GRU4REC	**0.3078**	***0.6827**	0.5040	0.0541
SR	0.3037	0.6530	0.6676	0.0722
SMF	0.3016	0.6658	0.5652	0.0545
MC	0.3004	0.6417	0.6454	0.0700
AR	0.2894	0.6360	0.6296	**0.0926**
V-SKNN	0.2828	0.6530	0.6186	0.0825
S-SKNN	0.2716	0.6019	0.6548	0.0715
SF-SKNN	0.2704	0.5890	0.6185	0.0656
S-KNN	0.2658	0.6213	0.6341	0.0846
IKNN	0.2081	0.4864	0.7553	0.0408
FPMC	0.2011	0.3626	**0.9754**	0.0545
BPR-MF	0.1761	0.2347	0.9107	0.0880
FISM	0.1145	0.1619	0.9740	0.0082
FOSSIL	0.0616	0.1896	0.9167	0.0482

(b) TMALL

Metrics	MRR@20	HR@20	COV@20	POP@20
S-SKNN	***0.1852**	0.3865	0.4667	0.0245
S-KNN	0.1815	***0.4038**	0.3807	0.0290
V-SKNN	0.1789	0.3732	0.4644	0.0248
BPR-MF	0.1590	0.2040	0.7226	**0.0572**
SF-SKNN	0.1357	0.2163	0.4358	0.0181
GRU4REC	0.1290	0.2770	0.1512	0.0354
AR	0.1289	0.2621	0.5091	0.0213
SR	0.1276	0.2418	0.5688	0.0214
SMF	0.1207	0.2605	0.2611	0.0359
MC	0.1156	0.2004	0.4980	0.0193
FPMC	0.1007	0.1191	**0.8803**	0.0051
IKNN	0.0510	0.1500	0.7280	0.0071
FISM	0.0237	0.0367	0.7522	0.0026
FOSSIL	0.0014	0.0036	0.5981	0.0158

(c) RETAILROCKET

Metrics	MRR@20	HR@20	COV@20	POP@20
S-SKNN	***0.3450**	***0.5906**	0.5957	0.0561
V-SKNN	0.3378	0.5726	0.5754	0.0608
S-KNN	0.3371	0.5830	0.5656	0.0608
BPR-MF	0.3029	0.3569	0.8241	0.0602
FPMC	0.2729	0.3200	**0.9291**	0.0224
SF-SKNN	0.2599	0.3576	0.4032	0.0345
SR	0.2454	0.4186	0.5240	0.0430
GRU4REC	0.2427	0.4796	0.6022	0.0597
AR	0.2411	0.4391	0.5443	0.0527
MC	0.2295	0.3585	0.4106	0.0350
SMF	0.2254	0.4593	0.4490	0.0845
IKNN	0.1072	0.2401	0.5844	0.0332
FISM	0.0750	0.1316	0.8482	0.0176
FOSSIL	0.0215	0.0580	0.7531	**0.1269**

(d) ZALANDO

Metrics	MRR@20	HR@20	COV@20	POP@20
SR	**0.3038**	0.4829	0.5860	0.0705
MC	0.3026	0.4553	0.5128	0.0599
IKNN	0.2745	0.4047	0.7141	0.0369
GRU4REC	0.2671	0.4678	0.3038	0.1006
SMF	0.2668	0.4470	0.3616	**0.1072**
AR	0.2578	0.4665	0.4672	0.0886
SF-SKNN	0.2494	0.4377	0.4324	0.0568
V-SKNN	0.2326	***0.5205**	0.4322	0.0993
S-SKNN	0.2193	0.4986	0.4350	0.0874
S-KNN	0.1715	0.4555	0.3085	0.1024
BPR-MF	0.1037	0.1617	0.6089	0.0575
FPMC	0.0508	0.0749	**0.8115**	0.0208
FISM	0.0044	0.0105	0.6241	0.0197
FOSSIL	0.0019	0.0047	0.6711	0.0342

(e) RSC15-S

Dataset	RSC15	
Metric	MRR@20	HR@20
GRU4REC	**0.3119**	**0.7190**
SMF	0.3086	0.7126
SR	0.3075	0.6903
V-SKNN	0.2735	0.6752
SKNN	0.2503	0.6410

Table 4. Precision (P@20) and Recall (R@20) for the e-commerce datasets. The table rows are ordered according to the P@20 values for the *TMALL* data set, which led to a relatively consistent ranking of the algorithms across the datasets.

Dataset	RSC15		TMALL		ROCKET		ZALANDO	
Metric	P@20	R@20	P@20	R@20	P@20	R@20	P@20	R@20
SKNN	0.0856	0.4638	**0.0945**	**0.3122**	**0.0562**	**0.4779**	0.0742	0.2018
V-SKNN	**0.0920**	0.4935	0.0876	0.2907	0.0553	0.4620	**0.0757**	**0.2070**
SMF	0.0916	**0.5005**	0.0679	0.2304	0.0471	0.3971	0.0616	0.1752
GRU4REC	0.0853	0.4698	0.0677	0.2331	0.0458	0.3997	0.0646	0.1814
SR	0.0889	0.4876	0.0519	0.1928	0.0380	0.3417	0.0595	0.1744

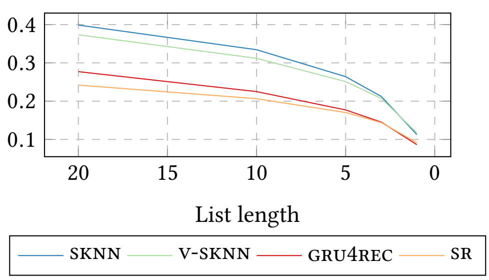
5.1.1 Accuracy Measures. The results when the task is to predict the *immediate next* element in a session (as done in [20, 27]) are shown in Tables 3a to 3d. The following observations in terms of the hit rate and the MRR can be made.[10]

- The lowest accuracy values are consistently achieved across all datasets by the family of Factorized Markov Chain approaches (FISM, FPMC and FOSSIL) and the session-aware BPR-MF variant. BPR-MF in fact often exhibits the best performance among these methods even thought it was not designed for sequential recommendation problems. These results show that the methods that were designed under the assumption of longer-term and richer user profiles are generally not particularly well suited for the specifics of session-based recommendation problems.
- The simple pairwise association methods (AR and SR) mostly occupy the middle places in our comparison. In most cases, it is preferable to consider the available sequentiality information (SR). Only for the *TMALL* dataset, where the transactions of an entire day are considered as a session[11], and for *RETAILROCKET*, the sequence-agnostic AR method is slightly better in terms of the hit rate. In terms of the overall ranking, the trivial SR method is, to some surprise, among the *top-performing* methods for two of the datasets in terms of the MRR, with good results also for the hit rate. The MC method finally, is usually placed somewhere in the middle of the ranking. Similar to the SR method, it is very strong in terms of the MRR for two of the datasets.
- The performance of the newly-proposed SMF method is very strong for the *RSC15* and *RSC15-S* dataset and in the middle ranges for the other datasets. The SMF method consistently outperforms the factorization-based methods from the literature, apparently due to the embedding of the current user session.
- GRU4REC is consistently among the top five algorithms in this comparison in terms of the hit rate and exhibits competitive performance results also with respect to the MRR. The method is outperforming all other methods on the *RSC15(-S)* datasets in terms of the hitrate and is competitive w.r.t. the MRR, where the differences between the top-performing methods are tiny. On the other datasets, the accuracy results of GRU4REC are, however, often significantly lower than those of the best-performing methods.[12]
- For each of the datasets, one of the proposed neighborhood-based methods was usually the winner in terms of the hit rate and the MRR (except for *RSC15(-S)* and the MRR on *ZALANDO*). Using one of the variants that considers sequentiality information is usually favorable, except for the case of the *TMALL* dataset. The most consistent performance of the neighborhood-based methods is achieved with the V-SKNN method which uses a specific sequence-aware similarity measure that gives more weight to the most recent interactions. And there is even room for further improvement in the context of the neighborhood-based methods. In the experiments reported in this work, we could, for example, observe that using a slightly different similarity measure already led to substantial performance improvements for some of the datasets.

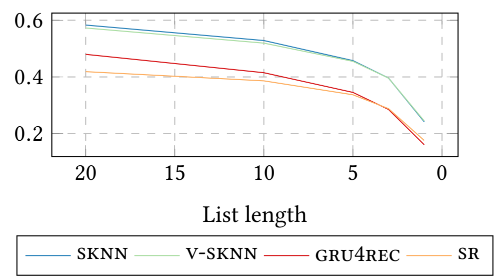
Precision and Recall for the Remaining Session. The ranking of the best-performing algorithms when considering *all* subsequent elements of a session and evaluating Precision and Recall is given in Table 4. The obtained results are mostly in line with the previously reported observations. The best performance is achieved by the neighborhood-based methods, with V-SKNN working very well

[10] We provide additional results that were obtained for measurements taken at multiple list lengths in Appendix B.
[11] In the dataset, timestamps are only available at the granularity of days.
[12] We applied the Wilcoxon signed-rank test ($\alpha = 0.05$) to determine the significance of differences between the two best performing approaches for each dataset.

across all datasets. Differently from the previous measurement, GRU4REC shows a lower performance for the *RSC-15* dataset than the other methods. This is probably due to the fact that GRU4REC is optimized to predict the *immediate* next action. Generally, which type of accuracy measurement—focusing on the prediction of the immediate next element or considering the prediction of any item that is relevant in the session as a success—is more appropriate, depends on the application domain. Our results show that the kNN-based methods are successful in both forms, i.e., they are often good at predicting the next element while, at the same time, they many times include *more* items that are relevant for the given session than, e.g., GRU4REC.



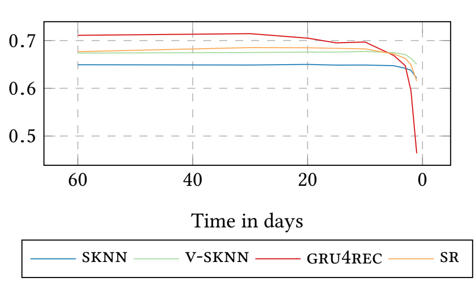
(a) TMALL



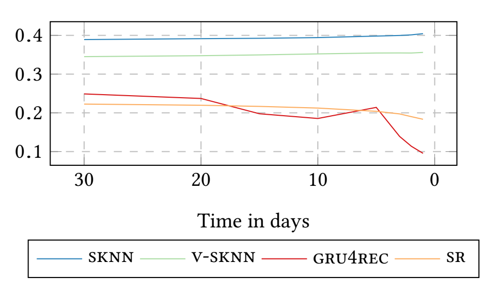
(b) RETAILROCKET

Fig. 3. Hit rate (HR) for two of the e-commerce datasets when reducing the result list length from 20 to 1.

Impact of Different List Lengths. To see if the list length at which the measurement is taken has an influence on the algorithm ranking, we varied the length from 20 to 1. Figure 3a and Figure 3b show how the best algorithms perform for the *TMALL* and *RETAILROCKET* datasets when different list lengths are used in the evaluation. The results show that the ranking of the algorithms is unaffected by the change of the list length. At shorter list lengths, the differences, however, become smaller. The trend is the same for the other e-commerce datasets.



(a) RSC15-S



(b) TMALL

Fig. 4. HR@20 for two e-commerce datasets when artificially reducing the size of the training set from 60 days to 1 day.

5.1.2 Cold-Start / Sparsity Effects. Previous experiments on the *RSC15* dataset revealed that discarding major parts of the older data has no strong impact on the prediction accuracy, at least in the e-commerce domain [27]. We therefore made additional experiments to analyze the effects in more depth. Figure 4a and Figure 4b show the results of this simulation for two of the e-commerce datasets.

The results for the *RSC15-S* (single-split) dataset (Figure 4a) are in line with what was previously reported in [27]. In the e-commerce domain, the user behavior seems to be strongly influenced by recent sales trends, an effect that was also reported in [28]. Discarding most of the historical data has almost no influence on the resulting hit rates. This behavior is similar for all compared algorithms. Only in the extreme case when only the data of the last few days is considered, the performance of the algorithms degrades. A similar observation can be made for the *TMALL* dataset. Generally, the observations also explain why the recency-based neighborhood sampling approach implemented in the kNN methods does not have a strong negative effect on the accuracy. In fact, focusing on the most recent sessions when looking for similar neighbors has shown to have a positive effect in [27], when compared to a random neighborhood selection scheme.

5.1.3 Coverage and Popularity Bias. In terms of the coverage (or: aggregate diversity), the factorization-based methods consistently lead to the highest values, i.e., the place the largest number of different items into the top-n lists of the users. GRU4REC represents in all datasets, except *RETAILROCKET*, the other extreme and seems to focus its recommendations on a comparably narrow range of items. In particular in the case of the *TMALL* dataset, the coverage of the item space of GRU4REC is as low as 0.15, i.e., the top-20 recommendations for all given sessions in the test set cover only 15% of the available items. To what extent low coverage is undesired, again depends on the specific application domain.

Not many consistent patterns can be identified with regard to the popularity biases of the different algorithms. BPR-MF, as was previously discussed in [26], has a comparably strong tendency to focus on generally popular items. Our newly proposed SMF method exhibits a similar tendency across all datasets. The FPMC method usually represents the other end of the spectrum. The tendency of the many of the other algorithms to recommend popular items seems to strongly depend on the dataset characteristics. According to our previous work [27], the basic SKNN method tends to recommend slightly more popular items than GRU4REC. In this new series of measurements, this is, however, not consistently the case across the datasets.

5.2 Media Datasets

Table 5, Table 6, and Table 7 show the results for the music and news domains, respectively.

Accuracy. The accuracy results generally exhibit similar patterns as the results obtained for the e-commerce datasets. For these datasets, however, the winning strategy more strongly depends on the chosen measure. When the MRR is used as a performance measure, often the trivial baselines SR or AR lead to the best results. In terms of the hitrate, in contrast, usually one of the nearest neighbor methods again performs best.

With respect to the MRR measure also GRU4REC exhibited very competitive performance, except for the *8TRACKS* and *AOTM* datasets, where the highest MRR values were achieved with the AR and the SKNN method. Looking at the playlist datasets (*8TRACKS* and *AOTM*), the comparably good results of the sequence-agnostic AR and SKNN strategy indicate that the ordering of the tracks is not too important for the playlist creators. Among the neighborhood-based methods, V-SKNN was again consistently among the top-performing methods. When looking at the standard Precision and Recall measurements for the five best-performing approaches in Table 6, we can see that V-SKNN is the winning strategy across all datasets and that GRU4REC is again less effective for this particular measurement.

Finally, looking at the *news* domain, the average results shown in Table 7 in general confirm the trends observed for the other datasets. The V-SKNN method is top-performing on almost all measures. GRU4REC also works comparably well on this dataset, especially on the precision and recall measures. Again, however, we can also observe a comparably low level of coverage and

Table 5. Hit rate (HR), Mean reciprocal rank (MRR), catalog coverage (COV), and the average popularity (POP) for a list length of 20 tested on the music datasets (Table 5a to 5d). The tables show the top ten algorithms ordered by MRR@20. Again, the best results are highlighted and significant differences are marked with a star.

(a) NOWPLAYING

Metrics	MRR@20	HR@20	COV@20	POP@20
SR	**0.1052**	0.2033	0.4655	0.0397
GRU4REC	0.1018	0.1969	0.4331	0.0515
MC	0.0971	0.1581	0.2935	0.0283
SF-SKNN	0.0954	0.1647	0.2772	0.0311
SMF	0.0881	0.1825	0.2416	**0.0915**
V-SKNN	0.0784	0.2551	0.4282	0.0662
S-SKNN	0.0776	*__0.2621__	0.4148	0.0623
AR	0.0710	0.2076	0.4530	0.0511
S-KNN	0.0689	0.2429	0.3006	0.0748
IKNN	0.0569	0.1821	**0.5798**	0.0293

(b) 8TRACKS

Metrics	MRR@20	HR@20	COV@20	POP@20
AR	*__0.0071__	0.0255	0.4529	0.0912
SMF	0.0064	0.0230	0.1527	0.0864
SR	0.0063	0.0170	0.4967	0.0531
SF-SKNN	0.0063	0.0118	0.3049	0.0362
V-SKNN	0.0057	0.0352	0.4080	0.1356
S-KNN	0.0053	*__0.0375__	0.2430	**0.1527**
IKNN	0.0050	0.0176	**0.6956**	0.0245
GRU4REC	0.0050	0.0189	0.0692	0.1222
S-SKNN	0.0047	0.0293	0.4509	0.0806
MC	0.0046	0.0098	0.3496	0.0320

(c) 30MUSIC

Metrics	MRR@20	HR@20	COV@20	POP@20
SR	*__0.2377__	0.3323	0.3892	0.0311
MC	0.2318	0.2843	0.2038	0.0205
GRU4REC	0.2263	0.3256	0.3447	0.0556
SF-SKNN	0.2079	0.2856	0.1854	0.0219
SMF	0.1776	0.2842	0.1508	**0.1047**
V-SKNN	0.1098	0.3818	0.3169	0.0554
IKNN	0.1086	0.2971	**0.4595**	0.0225
S-SKNN	0.1076	*__0.3855__	0.2931	0.0515
AR	0.0960	0.3087	0.3524	0.0393
S-KNN	0.0897	0.3443	0.1912	0.0626

(d) AOTM

Metrics	MRR@20	HR@20	COV@20	POP@20
SMF	**0.0111**	0.0297	0.2456	**0.1998**
SF-SKNN	0.0110	0.0144	0.3558	0.0508
SR	0.0076	0.0195	0.5864	0.0742
GRU4REC	0.0071	0.0156	0.4652	0.1151
MC	0.0063	0.0132	0.3803	0.0497
AR	0.0059	0.0233	0.5531	0.1049
V-SKNN	0.0054	0.0377	0.5362	0.1454
S-SKNN	0.0054	0.0397	0.5356	0.1289
S-KNN	0.0053	*__0.0429__	0.2802	0.1856
IKNN	0.0049	0.0186	**0.7879**	0.0472

Table 6. Precision (P@20) and Recall (R@20) for the music datasets. The results are ordered by P@20 for *8TRACKS*, which represents the largest music dataset in our evaluation.

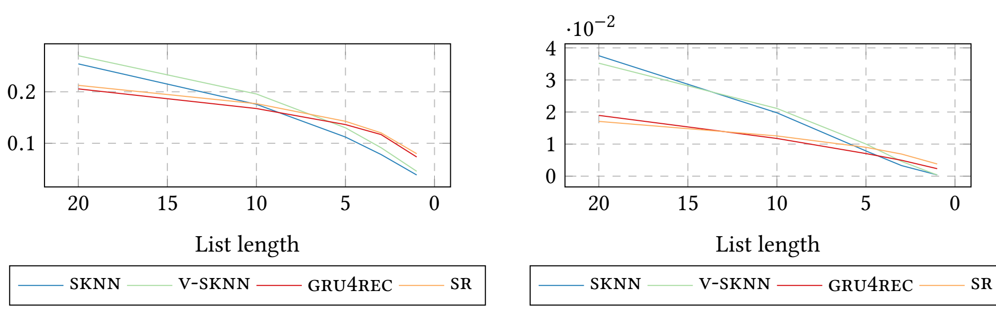
Dataset	LFM		8TRACKS		30MUSIC		AOTM	
Metric	P@20	R@20	P@20	R@20	P@20	R@20	P@20	R@20
V-SKNN	**0.0717**	**0.1909**	**0.0122**	0.0308	**0.1094**	**0.2321**	0.0133	0.0361
SKNN	0.0680	0.1824	0.0117	**0.0313**	0.1035	0.2140	**0.0155**	**0.0440**
SMF	0.0499	0.1453	0.0086	0.0218	0.0746	0.1655	0.0084	0.0259
SR	0.0501	0.1465	0.0055	0.0140	0.0878	0.2010	0.0053	0.0146
GRU4REC	0.0272	0.0810	0.0037	0.0095	0.0404	0.0988	0.0010	0.0027

a comparably high tendency to recommend popular items. In contrast to all other domains and datasets however, when looking at the results of the individual splits for the *CLEF* dataset, we could observe that those are subject to large fluctuations. Depending on the day that was chosen for testing, the ranking of the algorithms in terms of the accuracy measures changes drastically, which we could not observe for any other dataset.

Table 7. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the *CLEF* dataset (ordered by MRR@20).

Metrics	MRR@20	HR@20	COV@20	POP@20	P@20	R@20
SMF	**0.2344**	0.7061	0.6495	0.0828	0.0616	0.5268
V-SKNN	0.2235	0.7755	0.6210	0.0827	0.0679	0.5841
SR	0.2229	0.6718	0.6553	0.0932	0.0582	0.5019
GRU4REC	0.2199	0.5684	0.1744	**0.0942**	**0.0724**	**0.6260**
S-KNN	0.2185	**0.7780**	0.6125	0.0842	0.0656	0.5771
FPMC	0.1711	0.5976	0.8467	0.0823	0.0601	0.5145
FOSSIL	0.1661	0.5706	0.9627	0.0786	0.0580	0.5036
FISM	0.1293	0.4031	**0.9973**	0.0796	0.0581	0.5045

As mentioned in Section 4, we conducted additional single split experiments to ensure that the reduced amount of training data in the sliding window protocol does not affect the performance of the model-based approaches. The single-split results in Appendix C reveal GRU4REC as the best-performing approach for this particular dataset, which was also the case for two of the individual splits. Thus, even though such large fluctuations did only occur in the news domain, this is an indicator that applying a single-split evaluation protocol can easily lead to “random” and misleading results.



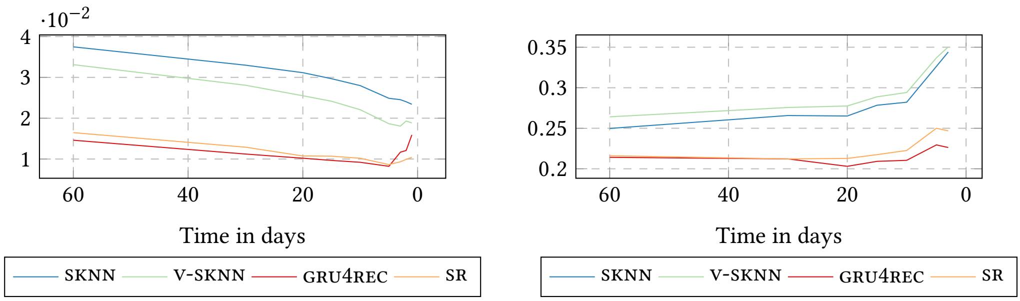
(a) NOWPLAYING (b) 8TRACKS

Fig. 5. Hit rate (HR) for two of the music datasets when reducing the result list length from 20 to 1.

The effects when considering different list lengths for two of the datasets is shown in Figure 5a (*NOWPLAYING*) and Figure 5b (*8TRACKS*). Differently from the e-commerce datasets, the relative ranking of the algorithms changes when the list lengths become shorter. For both datasets, the GRU4REC method and the very simple AR and SR methods, respectively, are slightly better in terms of the hit rate when it comes to very short list lengths. Considering the good results for the MRR for these methods (Table 5a and Table 5b), this was expected.

Cold-Start / Sparsity Effects. An interesting effect can be observed when older data is discarded to simulate sparsity effects. Figure 6a and Figure 6b show the results for the *8TRACKS* and *NOWPLAYING* datasets, respectively.[13] While for the *8TRACKS* playlist dataset the accuracy values more or less consistently decrease when older data is discarded, we can observe an *increase* in accuracy for the *NOWPLAYING* dataset. Remember that this dataset is based on the analysis of user posts on Twitter about their current listening behavior. Obtaining the highest accuracy values when

[13] The other media datasets did not exhibit any notable particularities.



(a) 8TRACKS (b) NOWPLAYING

Fig. 6. HR@20 for two music datasets when incrementally reducing the size of the training set to 1 day.

only considering the very last days means that this dataset is strongly dominated by short-term popularity trends and that the recommendation of older, non-trending tracks is detrimental to the accuracy results.

Coverage and Popularity Bias. In terms of coverage (see Table 5), the findings for datasets from the media domain are also mostly in line with those for the e-commerce datasets. The ranking of the algorithms varies largely across the datasets. The differences are, however, often less pronounced. Regarding the popularity tendency of the algorithms, methods that are based on pairwise sequences (SR and MC) in most cases lead to the recommendation of lesser known items, while nearest-neighbor-based techniques quite often focus on the recommendation of comparably popular objects.

5.3 Computational Complexity & Memory Usage

The methods included in our comparison vary largely in terms of the computational complexity and their memory requirements. Since neighborhood-based methods do not scale well when applied in a naive manner, we used implementation variants that rely on neighborhood sampling and specific in-memory data structures. The comparison of SKNN method and GRU4REC in [27] showed that, with such an implementation, recommendations can be quickly computed at prediction time with nearest neighbor methods, even though the prediction performance of model-based techniques like GRU4REC could not be achieved.

To enable comparability with previous research [27], we report the running times and memory demands for the single-split *RSC15-S* dataset, which is also the largest one in terms of the recorded user actions. Additionally, we include the *8TRACKS* dataset, which is rather small compared to *RSC15-S* in terms of the number of events, but has the largest product catalog of all datasets. Table 8 shows the times required for training the model (if applicable), the time needed to compute a recommendation at prediction time, and the memory requirements for the internal data structures. The reported results were obtained when using an Intel Core i7 4790K processor with 32GB of DDR3-1600 memory and a Nvidia GeForce GTX 960 graphics card with 2GB of memory. The following observations can be made.

Running Times. The simple methods in our comparative evaluation need from less than one to about three minutes of "training" (e.g., co-occurrence counting or in-memory data structure setup) for the *RSC15-S* dataset. The factorization-based methods and the deep learning based method, on the other hand, need about 6 to 8 hours to learn a model for the single data split. Note that while the deep learning method GRU4REC and the factorization-based approach SMF do not take the longest absolute time in this comparison, they are the only method for which the computations

Table 8. Overview of Computation Times and Memory Requirements for the *RSC15-S* dataset and the first split of the *8TRACKS* dataset, ordered in terms of required training times for the *RSC15-S* dataset.

Dataset	RSC15-S			8TRACKS		
Technique/Metric	Train. (min)	Pred. (ms)	Mem. (MB)	Train. (min)	Pred. (ms)	Mem. (MB)
MC	0.77	3.34	38.45	0.05	14.71	143.58
S-KNN	1.24	33.05	6051.30	0.04	52.96	352.83
S-SKNN	1.26	30.26	6051.30	0.05	51.01	352.83
V-SKNN	1.30	32.67	6051.30	0.04	52.43	352.83
SF-SKNN	1.72	29.82	6253.90	0.06	18.82	493.11
SR	2.41	3.14	54.32	0.18	15.88	283.67
AR	3.00	3.36	40.06	0.28	15.79	257.24
FISM	356.84	8.40	4936.60	35.07	60.72	386.77
GRU4REC (on a GPU)	385.35	7.43	59.17	19.08	58.08	587.57
BPR-MF	392.60	8.37	8009.30	42.69	65.15	771.48
SMF	446.66	14.02	1639.90	77.50	43.36	1805.30
FPMC	469.39	9.08	6786.30	60.92	71.58	1301.50
FOSSIL	499.19	10.56	4986.50	50.99	64.91	581.54

are done on the GPU. Running GRU4REC, for example, on a CPU tripled the computation times according to the measurements in [27].

Looking at the times needed to compute a single recommendation list, given a session beginning, we can observe that the simple rule-based methods AR, MC, and SR are among the fastest ones with prediction times at about 3 ms for the *RSC15-S* dataset. The factorization-based methods and GRU4REC are also very efficient, with prediction times mostly below 10 ms on average. The nearest-neighbor methods are slower for this task as they have to consider the neighbors in the prediction process. Since the neighbors can be determined through fast lookup operations, the overall prediction time even for the more elaborate S-SKNN and V-SKNN similarity schemes never exceeds 33 ms for creating a recommendation list.

Looking at the *8TRACKS* dataset with its large number of items, we can, however, see that the prediction times for many algorithms, including GRU4REC and several of the factorization-based ones significantly increase, while the prediction time for the neighborhood models only doubles. In the end, making the neighborhood-based computations is at least as fast as computing the predictions based on the offline-trained models. Overall, due to the used in-memory data structures and through the neighborhood sampling approach, such neighborhood models are also suited under the narrow time constraints of real-time recommendations. Differently from other methods, newly arriving interaction signals can be easily included in the underlying model without re-training.

Memory Requirements. In terms of the memory requirements, the rule-based methods AR, MC, and SR that basically record item co-occurrences of size two require the least memory, i.e. below 100 megabytes. Also the memory demands of GRU4REC are very low in this comparison, and GRU4REC occupies only about 60 MB of memory on the graphics card for the *RSC15-S* dataset. The factorization-based methods and the neighborhood methods, in contrast, have substantially higher memory requirements. The lookup data structures of the neighborhood-based methods, for example, in our implementation occupy about 6 GB of memory. When additional recency-based sampling is applied, which according to the analyses above does not hurt accuracy, these demands could, however, be substantially lowered.

For some algorithms, the memory requirements largely depend on the characteristics of the datasets. Looking at the numbers for the *8TRACKS* dataset, which covers over 300,000 different items (in contrast to the about 30,000 of the *RSC15* dataset), we see that in particular the memory demand of GRU4REC substantially increases with the number of items. As a result, GRU4REC's network even needs more memory than neighborhood-based methods for this dataset. Given these observations it seems promising to implement additional data sampling strategies within the more complex methods—as we did for the nearest neighbor methods—to decrease their computational demands.

6 CONCLUSION

Being able to predict the user's short-term interest in an online session is a highly relevant problem in practice, which has raised increased interest also in the academic field in recent years. Even though a number of different algorithmic approaches were proposed over the years, no standard benchmark datasets and baseline algorithms exist today. In this work, we have compared a number of very recent and computationally complex algorithms for session-based recommendation with more light-weight approaches based, e.g., on session neighborhoods. The experimental analyses on a number of different datasets show that in many cases one of the more simple methods is able to outperform even the most recent methods based on recurrent neural networks in terms of the prediction accuracy. At the same time, the computational demands of these methods can be kept comparably low when using in-memory cache data structures and data sampling.

Overall, the results, therefore, indicate that more research is required with respect to the development of more sophisticated models that are able to even better leverage the sequential information in the data. This is in particular the case, as several improvements for the nearest-neighbor methods can be imagined as well. In this work, we could for example observe that already using a different similarity measure, as done in the V-SKNN method, can lead to substantial performance improvements for different datasets.

As a side result, we could observe that using the latent feature vectors of the items of the current session for sequential factorization-based methods does not lead to high accuracy values. Instead, algorithms have to be used as baselines in performance comparisons that were specifically designed for the session-based recommendation task.

A number of aspects are still open to be explored in future works, in particular with respect to the integration of additional information like different types of user actions (e.g., add-to-cart, add-to-wishlist, etc.), item metadata, external context information, trends in the community, or long-term user preference data when users are logged in. Finally, since the relative performance of the different algorithms tested in our work sometimes varies across different datasets, more research is required to understand in which situations certain algorithms are better suited than others. These insights can then be further used to inform the design of hybrid recommendation approaches, which have shown to lead to the highest recommendation accuracy for session-based recommendation in [27].

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2012. Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques. *IEEE Trans. on Knowl. and Data Eng.* 24, 5 (May 2012), 896–911.

[2] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting The Next App That You Are Going To Use. In *WSDM '15*. 285–294.

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

[4] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist Prediction via Metric Embedding. In *KDD '12*. 714–722.

[5] Shuo Chen, Jiexun Xu, and Thorsten Joachims. 2013. Multi-space Probabilistic Sequence Modeling. In *KDD '13*. 865–873.

[6] Chen Cheng, Haiqin Yang, Michael R. Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-interest Recommendation. In *IJCAI '13*. 2605–2611.

[7] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP '14*. 1724–1734.

[8] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube Video Recommendation System. In *RecSys '10*. 293–296.

[9] Robin Devooght and Hugues Bersini. 2017. Long and Short-Term Recommendations with Recurrent Neural Networks. In *UMAP '17*. 13–21.

[10] Nemanja Djuric, Vladan Radosavljevic, Mihajlo Grbovic, and Narayan Bhamidipati. 2014. Hidden Conditional Random Fields with Deep User Embeddings for Ad Targeting. In *ICDM '14*. 779–784.

[11] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent Marked Temporal Point Processes: Embedding Event History to Vector. In *KDD '16*. 1555–1564.

[12] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159.

[13] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation. In *IJCAI '15*. 2069–2075.

[14] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized News Recommendation with Context Trees. In *RecSys '13*. 105–112.

[15] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in Your Inbox: Product Recommendations at Scale. In *KDD '15*. 1809–1818.

[16] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latent topic sequential patterns. In *RecSys '12*. 131–131.

[17] Jing He, Xin Li, Lejian Liao, Dandan Song, and William Cheung. 2016. Inferring a Personalized Next Point-of-Interest Recommendation Model with Latent Behavior Patterns. In *AAAI '16*.

[18] Qi He, Daxin Jiang, Zhen Liao, Steven C. H. Hoi, Kuiyu Chang, Ee-Peng Lim, and Hang Li. 2009. Web Query Recommendation via Sequential Query Prediction. In *ICDE '09*. 1443–1454.

[19] Ruining He and Julian McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. *CoRR* abs/1609.09152 (2016). https://arxiv.org/abs/1609.09152

[20] Balázs Hidasi and Alexandros Karatzoglou. 2017. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. *CoRR* abs/1706.03847 (2017). arXiv:1706.03847 http://arxiv.org/abs/1706.03847

[21] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR '16*.

[22] 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.

[23] Mehdi Hosseinzadeh Aghdam, Negar Hariri, Bamshad Mobasher, and Robin Burke. 2015. Adapting Recommendations to Contextual Changes Using Hierarchical Hidden Markov Models. In *RecSys '15*. 241–244.

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

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

[26] 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* 25, 5 (2015), 427–491.

[27] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *RecSys '17*. 306–310.

[28] Dietmar Jannach, Malte Ludewig, and Lukas Lerche. 2017. Session-based Item Recommendation in E-Commerce: On Short-Term Intents, Reminders, Trends, and Discounts. *User-Modeling and User-Adapted Interaction* 27, 3–5 (2017), 351–392.
[29] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored Item Similarity Models for top-N Recommender Systems. In *KDD '13*. 659–667.
[30] Iman Kamehkhosh, Dietmar Jannach, and Malte Ludewig. 2017. A Comparison of Frequent Pattern Techniques and a Deep Learning Method for Session-Based Recommendation. In *TempRec Workshop at ACM RecSys '17*. Como, Italy.
[31] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980
[32] Lukas Lerche, Dietmar Jannach, and Malte Ludewig. 2016. On the Value of Reminders Within E-Commerce Recommendations. In *UMAP '16*. 27–35.
[33] Defu Lian, Vincent W. Zheng, and Xing Xie. 2013. Collaborative Filtering Meets Next Check-in Location Prediction. In *WWW '13*. 231–232.
[34] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (Jan. 2003), 76–80.
[35] Yanchi Liu, Chuanren Liu, Bin Liu, Meng Qu, and Hui Xiong. 2016. Unified Point-of-Interest Recommendation with Temporal Interval Assessment. In *KDD '16*. 1015–1024.
[36] Brian Mcfee and Gert Lanckriet. 2011. The Natural Language of Playlists. In *ISMIR '11*. 537–541.
[37] Brian McFee and Gert R. G. Lanckriet. 2012. Hypergraph Models of Playlist Dialects. In *ISMIR '12*. 343–348.
[38] 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.
[39] Omar Moling, Linas Baltrunas, and Francesco Ricci. 2012. Optimal Radio Channel Recommendations with Explicit and Implicit Feedback. In *RecSys '12*. 75–82.
[40] Nagarajan Natarajan, Donghyuk Shin, and Inderjit S. Dhillon. 2013. Which App Will You Use Next?: Collaborative Filtering with Interactional Context. In *RecSys '13*. 201–208.
[41] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *Comput. Surveys* (2018).
[42] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In *RecSys '17*.
[43] Siddharth Reddy, Igor Labutov, and Thorsten Joachims. 2016. Learning Student and Content Embeddings for Personalized Lesson Sequence Recommendation. In *ACM Learning @ Scale '16*. 93–96.
[44] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI '09*. 452–461.
[45] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing Personalized Markov Chains for Next-basket Recommendation. In *WWW '10*. 811–820.
[46] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-Based Recommender System. *J. Mach. Learn. Res.* 6 (2005), 1265–1295.
[47] Harold Soh, Scott Sanner, Madeleine White, and Greg Jamieson. 2017. Deep Sequential Recommendation for Personalized Adaptive User Interfaces. In *IUI '17*. 589–593.
[48] Qiang Song, Jian Cheng, Ting Yuan, and Hanqing Lu. 2015. Personalized Recommendation Meets Your Next Favorite. In *CIKM '15*. 1775–1778.
[49] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-Rate Deep Learning for Temporal Recommendation. In *SIGIR '16*. 909–912.
[50] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A Hierarchical Recurrent Encoder-Decoder for Generative Context-Aware Query Suggestion. In *CIKM '15*. 553–562.
[51] Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, and Akira Tajima. 2015. Modeling User Activities on the Web Using Paragraph Vector. In *WWW '15*. 125–126.
[52] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. In *DLRS '16 Workshop at ACM RecSys*. 17–22.
[53] Maryam Tavakol and Ulf Brefeld. 2014. Factored MDPs for Detecting Topics of User Sessions. In *RecSys '14*. 33–40.
[54] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music Listening and Playlists Dataset. In *Poster Proceedings of RecSys '15*.
[55] Bartlomiej Twardowski. 2016. Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks. In *RecSys '16*. 273–276.
[56] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation. In *RecSys '16*. 225–232.
[57] Koen Verstrepen and Bart Goethals. 2014. Unifying Nearest Neighbors Collaborative Filtering. In *RecSys '14*. 177–184.

[58] Xiang Wu, Qi Liu, Enhong Chen, Liang He, Jingsong Lv, Can Cao, and Guoping Hu. 2013. Personalized Next-song Recommendation in Online Karaokes. In *RecSys '13*. 137–140.

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

[60] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A Dynamic Recurrent Model for Next Basket Recommendation. In *SIGIR '16*. 729–732.

[61] 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.

[62] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR* abs/1212.5701 (2012). arXiv:1212.5701 http://arxiv.org/abs/1212.5701

[63] 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.

[64] Elena Zheleva, John Guiver, Eduarda Mendes Rodrigues, and Nataša Milić-Frayling. 2010. Statistical Models of Music-listening Sessions in Social Media. In *WWW '10*. 1019–1028.

A PARAMETER CONFIGURATIONS

Table 9. Parameters for algorithm GRU4REC for all datasets.

Dataset	layer_size	objective	lr	momentum	drop_out
RSC15	100	BPR_{max}	0.20	0.5	0.0
TMALL	100	$TOP1_{max}$	0.05	0.0	0.3
RETAILROCKET	100	$TOP1_{max}$	0.15	0.3	0.0
8TRACKS	100	$TOP1_{max}$	0.10	0.0	0.7
AOTM	100	BPR_{max}	0.10	0.3	0.1
NOWPLAYING	100	BPR_{max}	0.10	0.5	0.1
30MUSIC	100	$TOP1_{max}$	0.10	0.1	0.1
ZALANDO	100	BPR_{max}	0.20	0.1	0.1
CLEF	100	$TOP1_{max}$	0.20	0.2	0.5
LASTFM	100	BPR_{max}	0.15	0.4	0.3

Table 10. Parameters used for the SMF algorithm for all datasets.

Dataset	layer_size	objective	lr	momentum	drop_out	skip
RSC15	100	$TOP1_{max}$	0.085	0.2	0.30	0.00
TMALL	100	BPR_{max}	0.015	0.6	0.00	0.00
RETAILROCKET	100	BPR_{max}	0.045	0.1	0.40	0.20
8TRACKS	100	$TOP1_{max}$	0.010	0.5	0.30	0.35
AOTM	100	BPR_{max}	0.090	0.8	0.40	0.20
NOWPLAYING	100	$TOP1_{max}$	0.055	0.2	0.40	0.20
30MUSIC	100	$TOP1_{max}$	0.100	0.1	0.20	0.20
ZALANDO	100	$TOP1_{max}$	0.030	0.3	0.25	0.00
CLEF	100	BPR_{max}	0.050	0.0	0.40	0.25
LASTFM	100	$TOP1_{max}$	0.015	0.3	0.15	0.45

Table 11. Parameters used for the V-SKNN algorithm for all datasets.

Dataset	K	samples
RSC15	200	2000
TMALL	200	2000
RETAILROCKET	200	2000
ZALANDO	200	2000
8TRACKS	200	2000
AOTM	200	2000
NOWPLAYING	100	1000
30MUSIC	100	1000
CLEF	100	1000
LASTFM	200	2000

Table 12. Parameters used for the SKNN, S-SKNN, and SF-SKNN algorithm for all datasets.

Dataset	K	samples
RSC15	500	1000
TMALL	100	500
RETAILROCKET	100	500
ZALANDO	100	500
8TRACKS	100	500
AOTM	100	500
NOWPLAYING	100	500
30MUSIC	100	500
CLEF	100	500
LASTFM	100	500

B FULL RESULT TABLES

Table 13. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the RSC15 dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
GRU4REC	0.308	0.683	0.504	0.054	0.301	0.591	0.431	0.058	0.285	0.470	0.355	0.062
SR	0.304	0.653	0.668	0.072	0.298	0.569	0.592	0.073	0.283	0.457	0.503	0.075
SMF	0.302	0.666	0.565	0.055	0.295	0.575	0.486	0.058	0.280	0.459	0.406	0.061
MC	0.300	0.642	0.645	0.070	0.295	0.562	0.584	0.071	0.280	0.452	0.506	0.073
AR	0.289	0.636	0.630	0.093	0.283	0.550	0.548	0.091	0.268	0.438	0.455	0.092
V-SKNN	0.283	0.653	0.619	0.079	0.277	0.563	0.534	0.081	0.261	0.446	0.451	0.084
S-SKNN	0.272	0.602	0.655	0.072	0.267	0.531	0.543	0.077	0.252	0.424	0.437	0.082
SF-SKNN	0.270	0.589	0.619	0.066	0.266	0.524	0.545	0.074	0.252	0.421	0.457	0.081
S-KNN	0.266	0.621	0.634	0.073	0.259	0.526	0.520	0.078	0.244	0.411	0.417	0.084
IKNN	0.208	0.486	0.755	0.041	0.203	0.408	0.671	0.046	0.190	0.315	0.566	0.050
fpMC	0.201	0.363	0.975	0.055	0.198	0.311	0.908	0.056	0.191	0.261	0.774	0.058
BPR-MF	0.176	0.235	0.911	0.088	0.175	0.223	0.793	0.079	0.174	0.214	0.630	0.070
FISM	0.115	0.162	0.974	0.008	0.114	0.149	0.917	0.012	0.112	0.135	0.810	0.019
FOSSIL	0.062	0.190	0.917	0.048	0.058	0.135	0.806	0.047	0.052	0.092	0.653	0.046

Table 14. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the RSC15S dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SMF	0.309	0.713	0.512	0.052	0.301	0.606	0.414	0.054	0.284	0.476	0.320	0.056
GRU4REC	0.308	0.719	0.350	0.033	0.301	0.609	0.277	0.034	0.283	0.473	0.207	0.037
SR	0.308	0.690	0.512	0.038	0.301	0.591	0.407	0.038	0.284	0.468	0.308	0.040
MC	0.296	0.667	0.518	0.039	0.289	0.567	0.413	0.039	0.273	0.444	0.314	0.041
AR	0.281	0.655	0.473	0.045	0.273	0.543	0.374	0.043	0.257	0.422	0.283	0.046
V-SKNN	0.274	0.675	0.427	0.037	0.266	0.562	0.328	0.039	0.248	0.429	0.242	0.042
S-SKNN	0.266	0.667	0.417	0.035	0.258	0.548	0.309	0.038	0.240	0.414	0.221	0.041
SF-SKNN	0.260	0.670	0.446	0.037	0.251	0.545	0.339	0.039	0.233	0.406	0.246	0.042
S-KNN	0.250	0.641	0.398	0.036	0.242	0.521	0.293	0.038	0.224	0.390	0.207	0.041

Table 15. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the TMALL dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
S-SKNN	0.185	0.387	0.467	0.025	0.181	0.330	0.309	0.027	0.173	0.267	0.196	0.031
S-KNN	0.182	0.404	0.381	0.026	0.177	0.334	0.249	0.029	0.168	0.264	0.161	0.032
V-SKNN	0.179	0.373	0.464	0.024	0.175	0.312	0.320	0.026	0.167	0.251	0.218	0.029
BPR-MF	0.159	0.204	0.723	0.057	0.159	0.197	0.534	0.076	0.157	0.189	0.343	0.097
SF-SKNN	0.136	0.216	0.436	0.018	0.135	0.203	0.338	0.022	0.132	0.182	0.243	0.026
GRU4REC	0.129	0.277	0.151	0.035	0.125	0.225	0.109	0.039	0.119	0.177	0.078	0.043
AR	0.129	0.262	0.509	0.021	0.126	0.217	0.358	0.024	0.120	0.175	0.235	0.027
SR	0.128	0.242	0.569	0.021	0.125	0.206	0.421	0.023	0.120	0.170	0.286	0.026
SMF	0.121	0.261	0.261	0.036	0.118	0.213	0.193	0.039	0.112	0.168	0.140	0.041
MC	0.116	0.200	0.498	0.019	0.114	0.178	0.391	0.022	0.111	0.151	0.284	0.025
fpMC	0.101	0.119	0.880	0.005	0.100	0.114	0.730	0.007	0.100	0.109	0.540	0.010
IKNN	0.051	0.150	0.728	0.007	0.048	0.112	0.575	0.008	0.044	0.079	0.403	0.009
FISM	0.024	0.037	0.752	0.003	0.023	0.032	0.586	0.003	0.023	0.028	0.419	0.004
FOSSIL	0.001	0.004	0.598	0.016	0.001	0.003	0.457	0.021	0.001	0.002	0.310	0.028

Table 16. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the RETAILROCKET dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
S-SKNN	0.345	0.591	0.596	0.056	0.341	0.537	0.480	0.066	0.332	0.470	0.344	0.076
V-SKNN	0.338	0.573	0.575	0.060	0.334	0.519	0.474	0.069	0.326	0.455	0.348	0.078
S-KNN	0.337	0.583	0.566	0.058	0.333	0.528	0.445	0.068	0.324	0.458	0.316	0.080
BPR-MF	0.303	0.357	0.824	0.060	0.303	0.352	0.627	0.072	0.302	0.345	0.417	0.083
fpMC	0.273	0.320	0.929	0.022	0.272	0.309	0.777	0.026	0.271	0.298	0.560	0.032
SF-SKNN	0.260	0.358	0.403	0.035	0.259	0.350	0.373	0.046	0.257	0.331	0.311	0.058
SR	0.245	0.419	0.524	0.042	0.243	0.386	0.458	0.050	0.236	0.337	0.354	0.059
GRU4REC	0.243	0.480	0.602	0.060	0.238	0.415	0.478	0.066	0.229	0.345	0.350	0.072
AR	0.241	0.439	0.544	0.053	0.238	0.390	0.449	0.061	0.230	0.331	0.333	0.071
MC	0.230	0.359	0.411	0.035	0.228	0.343	0.383	0.045	0.224	0.308	0.322	0.056
SMF	0.225	0.459	0.449	0.085	0.221	0.393	0.360	0.092	0.211	0.322	0.270	0.099
IKNN	0.107	0.240	0.584	0.033	0.105	0.202	0.505	0.038	0.099	0.159	0.388	0.042
FISM	0.075	0.132	0.848	0.018	0.074	0.112	0.672	0.019	0.071	0.094	0.474	0.023
FOSSIL	0.022	0.058	0.753	0.127	0.020	0.043	0.560	0.150	0.019	0.032	0.377	0.171

Table 17. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the ZALANDO dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.304	0.483	0.586	0.061	0.302	0.462	0.433	0.066	0.298	0.429	0.290	0.069
MC	0.303	0.455	0.513	0.060	0.302	0.441	0.412	0.066	0.298	0.415	0.292	0.069
IKNN	0.275	0.405	0.714	0.037	0.273	0.385	0.532	0.041	0.270	0.362	0.349	0.047
GRU4REC	0.267	0.468	0.304	0.101	0.265	0.433	0.239	0.103	0.259	0.389	0.182	0.100
SMF	0.267	0.447	0.362	0.107	0.265	0.418	0.282	0.108	0.259	0.380	0.210	0.104
AR	0.258	0.467	0.467	0.089	0.256	0.435	0.337	0.090	0.250	0.393	0.233	0.088
SF-SKNN	0.249	0.438	0.432	0.057	0.249	0.430	0.348	0.068	0.245	0.403	0.249	0.074
V-SKNN	0.233	0.521	0.432	0.096	0.230	0.482	0.296	0.096	0.222	0.422	0.197	0.092
S-SKNN	0.219	0.499	0.435	0.087	0.216	0.456	0.280	0.092	0.207	0.388	0.174	0.095
S-KNN	0.172	0.456	0.309	0.093	0.167	0.380	0.201	0.097	0.154	0.290	0.125	0.103
BPR-MF	0.104	0.162	0.609	0.058	0.103	0.152	0.415	0.069	0.102	0.141	0.247	0.083
fpMC	0.051	0.075	0.812	0.021	0.050	0.067	0.629	0.022	0.049	0.061	0.434	0.025
FISM	0.004	0.011	0.624	0.020	0.004	0.008	0.444	0.020	0.004	0.006	0.290	0.021
FOSSIL	0.002	0.005	0.671	0.034	0.002	0.004	0.493	0.036	0.002	0.003	0.333	0.037

Table 18. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the 8TRACKS dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
AR	0.0071	0.0255	0.4530	0.0912	0.0066	0.0173	0.3178	0.1075	0.0057	0.0108	0.1998	0.1251
SMF	0.0064	0.0231	0.1528	0.0864	0.0058	0.0148	0.1076	0.0916	0.0051	0.0092	0.0753	0.0958
SR	0.0064	0.0171	0.4967	0.0531	0.0061	0.0125	0.3645	0.0636	0.0056	0.0090	0.2395	0.0744
SF-SKNN	0.0064	0.0119	0.3049	0.0363	0.0063	0.0102	0.2439	0.0517	0.0060	0.0083	0.1794	0.0690
V-SKNN	0.0057	0.0352	0.4081	0.1194	0.0048	0.0211	0.2523	0.1353	0.0034	0.0102	0.1496	0.1364
S-KNN	0.0053	0.0376	0.2431	0.1080	0.0041	0.0198	0.1544	0.1153	0.0026	0.0079	0.0975	0.1065
IKNN	0.0051	0.0177	0.6956	0.0245	0.0047	0.0124	0.5101	0.0267	0.0041	0.0078	0.3293	0.0286
GRU4REC	0.0050	0.0189	0.0693	0.1223	0.0045	0.0118	0.0512	0.1326	0.0039	0.0070	0.0385	0.1414
S-SKNN	0.0048	0.0293	0.4509	0.0807	0.0040	0.0182	0.2741	0.0961	0.0026	0.0081	0.1404	0.1028
MC	0.0046	0.0099	0.3496	0.0321	0.0045	0.0079	0.2756	0.0402	0.0043	0.0062	0.2022	0.0493
BPR-MF	0.0002	0.0004	0.6138	0.0088	0.0002	0.0003	0.4253	0.0131	0.0002	0.0003	0.2576	0.0203
FOSSIL	0.0001	0.0002	0.6703	0.0084	0.0001	0.0002	0.4941	0.0121	0.0001	0.0001	0.3338	0.0178
fpMC	0.0000	0.0001	0.7608	0.0031	0.0000	0.0001	0.5729	0.0035	0.0000	0.0001	0.3879	0.0043
FISM	0.0000	0.0001	0.6210	0.0027	0.0000	0.0000	0.4460	0.0028	0.0000	0.0000	0.2952	0.0030

Table 19. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the AOTM dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SMF	0.0111	0.0298	0.2457	0.1998	0.0105	0.0205	0.1795	0.2085	0.0097	0.0149	0.1265	0.2136
SF-SKNN	0.0111	0.0145	0.3559	0.0508	0.0110	0.0139	0.3022	0.0686	0.0109	0.0130	0.2306	0.0875
SR	0.0077	0.0195	0.5864	0.0533	0.0073	0.0149	0.4481	0.0599	0.0068	0.0107	0.3015	0.0626
GRU4REC	0.0072	0.0157	0.4653	0.1151	0.0070	0.0125	0.3550	0.1131	0.0067	0.0102	0.2432	0.1141
MC	0.0063	0.0133	0.3803	0.0498	0.0062	0.0112	0.3198	0.0602	0.0059	0.0089	0.2449	0.0681
AR	0.0059	0.0233	0.5532	0.1049	0.0053	0.0146	0.4003	0.1178	0.0046	0.0089	0.2543	0.1318
V-SKNN	0.0055	0.0378	0.5363	0.1397	0.0043	0.0208	0.3357	0.1550	0.0027	0.0085	0.1927	0.1592
S-SKNN	0.0055	0.0397	0.5357	0.1289	0.0042	0.0209	0.3228	0.1475	0.0025	0.0077	0.1718	0.1558
S-KNN	0.0054	0.0429	0.2802	0.1678	0.0038	0.0200	0.1785	0.1666	0.0021	0.0063	0.1108	0.1549
IKNN	0.0049	0.0187	0.7880	0.0473	0.0045	0.0122	0.5777	0.0481	0.0038	0.0073	0.3591	0.0500
FOSSIL	0.0007	0.0027	0.5529	0.0978	0.0006	0.0017	0.3717	0.1139	0.0005	0.0010	0.2311	0.1308
BPR-MF	0.0005	0.0018	0.5659	0.0968	0.0005	0.0012	0.3550	0.1180	0.0004	0.0007	0.1900	0.1403
fpMC	0.0003	0.0007	0.7851	0.0264	0.0003	0.0006	0.5867	0.0289	0.0003	0.0004	0.3884	0.0325
FISM	0.0001	0.0004	0.6172	0.0272	0.0001	0.0002	0.4296	0.0288	0.0001	0.0001	0.2743	0.0311

Table 20. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the 30MUSIC dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.2377	0.3323	0.3893	0.0232	0.2363	0.3120	0.2913	0.0273	0.2326	0.2845	0.1920	0.0294
MC	0.2318	0.2844	0.2038	0.0205	0.2314	0.2780	0.1804	0.0265	0.2298	0.2663	0.1467	0.0309
GRU4REC	0.2264	0.3257	0.3447	0.0556	0.2249	0.3042	0.2423	0.0567	0.2215	0.2796	0.1629	0.0557
SF-SKNN	0.2079	0.2856	0.1854	0.0219	0.2078	0.2834	0.1634	0.0302	0.2062	0.2726	0.1318	0.0371
SMF	0.1777	0.2843	0.1508	0.1048	0.1756	0.2547	0.1117	0.1062	0.1712	0.2223	0.0817	0.1057
V-SKNN	0.1099	0.3819	0.3170	0.0538	0.1040	0.3002	0.1944	0.0573	0.0882	0.1813	0.1165	0.0612
IKNN	0.1086	0.2971	0.4596	0.0226	0.1053	0.2501	0.3122	0.0249	0.0956	0.1788	0.1961	0.0248
S-SKNN	0.1077	0.3856	0.2931	0.0515	0.1014	0.2975	0.1759	0.0569	0.0851	0.1753	0.1043	0.0629
AR	0.0960	0.3088	0.3524	0.0394	0.0911	0.2395	0.2375	0.0433	0.0803	0.1580	0.1466	0.0476
S-KNN	0.0898	0.3443	0.1912	0.0574	0.0832	0.2501	0.1155	0.0637	0.0689	0.1424	0.0691	0.0714
BPR-MF	0.0427	0.0580	0.4521	0.0281	0.0425	0.0548	0.2792	0.0381	0.0421	0.0515	0.1523	0.0518
fpMC	0.0293	0.0359	0.6544	0.0078	0.0291	0.0334	0.4556	0.0085	0.0289	0.0317	0.2872	0.0096
FISM	0.0029	0.0047	0.4676	0.0084	0.0028	0.0038	0.3052	0.0089	0.0028	0.0034	0.1859	0.0095
FOSSIL	0.0029	0.0100	0.3347	0.0297	0.0027	0.0073	0.1919	0.0436	0.0023	0.0048	0.0914	0.0634

Table 21. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the NOWPLAYING dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.1053	0.2033	0.4656	0.0247	0.1031	0.1712	0.3605	0.0284	0.0988	0.1395	0.2490	0.0305
GRU4REC	0.1018	0.1970	0.4331	0.0516	0.0995	0.1632	0.3261	0.0529	0.0958	0.1352	0.2282	0.0540
MC	0.0971	0.1582	0.2936	0.0284	0.0960	0.1417	0.2547	0.0347	0.0935	0.1236	0.2024	0.0409
SF-SKNN	0.0954	0.1647	0.2773	0.0311	0.0945	0.1524	0.2369	0.0414	0.0921	0.1344	0.1856	0.0518
SMF	0.0882	0.1825	0.2417	0.0916	0.0859	0.1484	0.1847	0.0960	0.0818	0.1181	0.1358	0.0985
V-SKNN	0.0785	0.2552	0.4283	0.0639	0.0737	0.1861	0.2904	0.0719	0.0651	0.1213	0.1819	0.0786
S-SKNN	0.0777	0.2622	0.4149	0.0624	0.0725	0.1880	0.2727	0.0699	0.0632	0.1181	0.1657	0.0771
AR	0.0710	0.2076	0.4531	0.0511	0.0672	0.1518	0.3261	0.0584	0.0611	0.1060	0.2114	0.0672
S-KNN	0.0689	0.2429	0.3007	0.0690	0.0637	0.1676	0.1962	0.0759	0.0556	0.1048	0.1226	0.0825
IKNN	0.0569	0.1822	0.5799	0.0294	0.0534	0.1321	0.4313	0.0308	0.0477	0.0884	0.2869	0.0317
BPR-MF	0.0392	0.0621	0.5903	0.0672	0.0387	0.0547	0.3764	0.0843	0.0378	0.0482	0.2043	0.0995
fpMC	0.0331	0.0470	0.7865	0.0154	0.0327	0.0418	0.5833	0.0190	0.0321	0.0371	0.3807	0.0238
FOSSIL	0.0136	0.0432	0.5950	0.0336	0.0127	0.0302	0.4035	0.0389	0.0113	0.0195	0.2502	0.0439
FISM	0.0108	0.0183	0.6451	0.0110	0.0105	0.0145	0.4551	0.0123	0.0102	0.0122	0.2934	0.0143

Table 22. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20 on the CLEF dataset (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SMF	0.234	0.706	0.650	0.083	0.222	0.529	0.582	0.097	0.198	0.354	0.511	0.109
MC	0.225	0.687	0.732	0.095	0.213	0.514	0.705	0.123	0.190	0.339	0.653	0.144
V-SKNN	0.224	0.776	0.621	0.082	0.211	0.596	0.566	0.113	0.185	0.404	0.495	0.154
SR	0.223	0.672	0.655	0.093	0.212	0.513	0.608	0.123	0.189	0.337	0.542	0.146
GRU4REC	0.220	0.568	0.174	0.094	0.212	0.462	0.129	0.118	0.195	0.331	0.101	0.138
S-KNN	0.219	0.778	0.613	0.084	0.205	0.588	0.545	0.122	0.179	0.394	0.476	0.164
AR	0.216	0.666	0.724	0.100	0.204	0.490	0.656	0.148	0.183	0.339	0.559	0.215
IKNN	0.188	0.596	0.746	0.059	0.177	0.436	0.722	0.047	0.159	0.300	0.669	0.046
fpMC	0.171	0.598	0.847	0.082	0.159	0.414	0.721	0.093	0.138	0.258	0.601	0.108
FOSSIL	0.166	0.571	0.963	0.079	0.155	0.417	0.864	0.093	0.136	0.268	0.703	0.101
FISM	0.129	0.403	0.997	0.080	0.122	0.297	0.963	0.108	0.109	0.201	0.857	0.140

C ADDITIONAL SINGLE SPLIT RESULTS

Table 23. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the TMALL dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
S-SKNN	0.181	0.385	0.560	0.019	0.177	0.329	0.375	0.021	0.168	0.265	0.240	0.023
S-KNN	0.177	0.398	0.461	0.020	0.173	0.334	0.306	0.022	0.163	0.263	0.199	0.024
V-SKNN	0.169	0.353	0.570	0.019	0.166	0.298	0.405	0.021	0.158	0.241	0.282	0.023
SF-SKNN	0.141	0.237	0.577	0.015	0.140	0.220	0.441	0.018	0.136	0.194	0.309	0.020
SMF	0.140	0.298	0.461	0.023	0.136	0.245	0.341	0.023	0.129	0.195	0.244	0.024
AR	0.131	0.254	0.628	0.019	0.128	0.214	0.457	0.021	0.123	0.174	0.312	0.023
SR	0.131	0.243	0.683	0.019	0.129	0.209	0.507	0.020	0.124	0.173	0.348	0.021
GRU4REC	0.123	0.263	0.171	0.029	0.120	0.213	0.117	0.032	0.114	0.169	0.082	0.034
MC	0.123	0.214	0.673	0.018	0.121	0.188	0.516	0.019	0.117	0.160	0.361	0.021
IKNN	0.049	0.147	0.801	0.006	0.047	0.111	0.644	0.006	0.042	0.077	0.471	0.007

Table 24. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the RETAILROCKET dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
S-SKNN	0.333	0.580	0.272	0.051	0.329	0.524	0.170	0.060	0.320	0.456	0.098	0.069
S-KNN	0.332	0.571	0.250	0.052	0.329	0.520	0.155	0.061	0.320	0.453	0.090	0.070
V-SKNN	0.327	0.582	0.272	0.061	0.323	0.520	0.172	0.069	0.313	0.446	0.104	0.075
SF-SKNN	0.301	0.463	0.224	0.036	0.299	0.444	0.168	0.049	0.293	0.399	0.110	0.060
SR	0.270	0.504	0.299	0.047	0.267	0.452	0.201	0.054	0.257	0.379	0.123	0.062
SMF	0.270	0.557	0.313	0.056	0.264	0.479	0.198	0.061	0.252	0.388	0.119	0.067
AR	0.265	0.485	0.286	0.058	0.261	0.425	0.189	0.064	0.253	0.366	0.115	0.074
MC	0.261	0.468	0.250	0.040	0.258	0.425	0.184	0.049	0.250	0.366	0.121	0.059
GRU4REC	0.260	0.559	0.291	0.055	0.254	0.475	0.187	0.062	0.240	0.377	0.114	0.069
IKNN	0.121	0.284	0.334	0.031	0.118	0.238	0.219	0.036	0.111	0.185	0.131	0.041

Table 25. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the ZALANDO dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.306	0.498	0.525	0.059	0.304	0.473	0.381	0.060	0.299	0.435	0.256	0.058
MC	0.304	0.469	0.504	0.052	0.302	0.452	0.381	0.055	0.298	0.422	0.262	0.056
IKNN	0.271	0.410	0.597	0.033	0.269	0.388	0.432	0.036	0.266	0.363	0.287	0.040
GRU4REC	0.267	0.483	0.290	0.073	0.265	0.442	0.223	0.074	0.258	0.394	0.167	0.073
AR	0.265	0.483	0.431	0.073	0.262	0.450	0.311	0.073	0.256	0.404	0.216	0.069
SMF	0.253	0.463	0.329	0.075	0.250	0.422	0.248	0.076	0.243	0.372	0.182	0.074
SF-SKNN	0.251	0.451	0.419	0.046	0.250	0.440	0.323	0.053	0.245	0.406	0.227	0.057
V-SKNN	0.237	0.517	0.396	0.077	0.234	0.478	0.275	0.076	0.226	0.419	0.185	0.071
S-SKNN	0.224	0.510	0.395	0.066	0.221	0.464	0.257	0.068	0.211	0.391	0.161	0.070
S-KNN	0.181	0.461	0.301	0.069	0.176	0.392	0.197	0.071	0.164	0.300	0.124	0.074

Table 26. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the 8TRACKS dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
AR	0.0135	0.0410	0.7769	0.0399	0.0126	0.0276	0.5772	0.0545	0.0114	0.0185	0.3933	0.0774
SR	0.0123	0.0329	0.8875	0.0243	0.0116	0.0233	0.7261	0.0302	0.0107	0.0166	0.5237	0.0374
SMF	0.0115	0.0476	0.0772	0.1197	0.0102	0.0290	0.0556	0.1303	0.0087	0.0172	0.0406	0.1401
V-SKNN	0.0110	0.0490	0.7180	0.0290	0.0098	0.0313	0.5317	0.0322	0.0079	0.0167	0.4041	0.0324
MC	0.0101	0.0234	0.8365	0.0152	0.0098	0.0179	0.7050	0.0179	0.0092	0.0134	0.5477	0.0216
S-KNN	0.0098	0.0438	0.6122	0.0272	0.0086	0.0267	0.4343	0.0298	0.0068	0.0128	0.3056	0.0267
S-SKNN	0.0097	0.0402	0.8543	0.0197	0.0087	0.0265	0.6465	0.0238	0.0070	0.0137	0.3984	0.0256
GRU4REC	0.0095	0.0376	0.0593	0.1930	0.0085	0.0231	0.0445	0.2140	0.0073	0.0140	0.0336	0.2350
SF-SKNN	0.0089	0.0217	0.7713	0.0157	0.0086	0.0171	0.6555	0.0219	0.0080	0.0131	0.5072	0.0286
IKNN	0.0072	0.0251	0.9852	0.0063	0.0066	0.0165	0.8756	0.0069	0.0058	0.0107	0.6635	0.0075

Table 27. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the AOTM dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SF-SKNN	0.0275	0.0440	0.8591	0.0828	0.0273	0.0403	0.7673	0.1025	0.0267	0.0360	0.6205	0.1192
SMF	0.0204	0.0468	0.9262	0.0941	0.0197	0.0361	0.8294	0.0941	0.0186	0.0280	0.6845	0.0927
GRU4REC	0.0154	0.0427	0.6523	0.1665	0.0146	0.0312	0.5081	0.1759	0.0134	0.0225	0.3629	0.1840
SR	0.0152	0.0449	0.9439	0.1061	0.0143	0.0318	0.8140	0.1143	0.0130	0.0217	0.6154	0.1226
MC	0.0134	0.0348	0.8996	0.0813	0.0128	0.0262	0.7889	0.0902	0.0119	0.0198	0.6326	0.0985
AR	0.0119	0.0426	0.8523	0.1409	0.0109	0.0283	0.6723	0.1536	0.0095	0.0178	0.4853	0.1689
V-SKNN	0.0104	0.0721	0.7971	0.1567	0.0083	0.0415	0.6049	0.1662	0.0051	0.0174	0.4490	0.1646
IKNN	0.0100	0.0384	0.9854	0.0482	0.0090	0.0242	0.8660	0.0490	0.0079	0.0153	0.6629	0.0499
S-SKNN	0.0095	0.0737	0.8917	0.1192	0.0071	0.0406	0.6652	0.1322	0.0035	0.0124	0.4265	0.1392
S-KNN	0.0087	0.0740	0.6400	0.1414	0.0059	0.0345	0.4599	0.1416	0.0028	0.0096	0.3142	0.1387

Table 28. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the 30MUSIC dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.2690	0.3744	0.5904	0.0373	0.2672	0.3499	0.4619	0.0389	0.2629	0.3179	0.3270	0.0387
MC	0.2653	0.3302	0.4001	0.0283	0.2646	0.3203	0.3455	0.0340	0.2625	0.3051	0.2790	0.0374
GRU4REC	0.2354	0.3651	0.4029	0.0665	0.2334	0.3372	0.3077	0.0669	0.2286	0.3011	0.2289	0.0653
SMF	0.2145	0.3614	0.3974	0.0608	0.2121	0.3275	0.3013	0.0605	0.2064	0.2851	0.2200	0.0587
SF-SKNN	0.2123	0.3320	0.3404	0.0284	0.2118	0.3258	0.2929	0.0366	0.2083	0.3013	0.2364	0.0432
IKNN	0.1352	0.3412	0.6758	0.0219	0.1319	0.2943	0.5043	0.0234	0.1215	0.2176	0.3427	0.0227
V-SKNN	0.1192	0.4134	0.4384	0.0589	0.1130	0.3263	0.2911	0.0599	0.0958	0.1972	0.1927	0.0630
AR	0.1157	0.3518	0.5352	0.0437	0.1107	0.2810	0.3886	0.0456	0.0985	0.1905	0.2579	0.0493
S-SKNN	0.1153	0.4119	0.4195	0.0544	0.1087	0.3190	0.2681	0.0593	0.0913	0.1881	0.1700	0.0660
S-KNN	0.0938	0.3609	0.2848	0.0595	0.0869	0.2626	0.1799	0.0658	0.0719	0.1481	0.1129	0.0742

Table 29. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the NOWPLAYING dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
SR	0.0856	0.1825	0.4629	0.0309	0.0831	0.1466	0.3390	0.0346	0.0789	0.1146	0.2290	0.0390
MC	0.0813	0.1474	0.3576	0.0248	0.0798	0.1255	0.2833	0.0287	0.0769	0.1041	0.2090	0.0336
SF-SKNN	0.0787	0.1602	0.3015	0.0264	0.0774	0.1431	0.2383	0.0333	0.0737	0.1157	0.1743	0.0403
SMF	0.0782	0.1881	0.3560	0.0322	0.0753	0.1454	0.2585	0.0332	0.0705	0.1085	0.1799	0.0341
GRU4REC	0.0771	0.1792	0.2202	0.0523	0.0742	0.1370	0.1647	0.0568	0.0700	0.1053	0.1196	0.0604
V-SKNN	0.0670	0.2291	0.3358	0.0445	0.0624	0.1627	0.2248	0.0497	0.0543	0.1019	0.1465	0.0543
S-SKNN	0.0669	0.2406	0.3436	0.0407	0.0618	0.1674	0.2186	0.0468	0.0532	0.1017	0.1329	0.0523
AR	0.0647	0.1866	0.4137	0.0381	0.0613	0.1378	0.2869	0.0441	0.0555	0.0947	0.1830	0.0531
S-KNN	0.0604	0.2241	0.2312	0.0453	0.0554	0.1512	0.1487	0.0509	0.0472	0.0888	0.0936	0.0560
IKNN	0.0467	0.1554	0.5526	0.0144	0.0437	0.1120	0.3960	0.0145	0.0384	0.0724	0.2597	0.0141

Table 30. Hit rate (HR), Mean reciprocal rank (MRR), Precision (P), Recall (R), item coverage (COV), and average popularity (POP) results for a list length of 20, 10, and 5 on the CLEF dataset with a single split (sorted by MRR@20).

Algorithm	MRR@20	HR@20	COV@20	POP@20	MRR@10	HR@10	COV@10	POP@10	MRR@5	HR@5	COV@5	POP@5
GRU4REC	0.253	0.724	0.154	0.051	0.242	0.564	0.123	0.064	0.219	0.384	0.101	0.088
SR	0.213	0.623	0.754	0.055	0.202	0.451	0.715	0.084	0.183	0.308	0.636	0.110
MC	0.213	0.625	0.755	0.056	0.202	0.459	0.724	0.080	0.184	0.325	0.652	0.110
SMF	0.207	0.646	0.765	0.042	0.194	0.472	0.709	0.042	0.172	0.300	0.639	0.023
V-SKNN	0.197	0.683	0.756	0.053	0.183	0.487	0.682	0.080	0.160	0.316	0.585	0.082
S-SKNN	0.190	0.670	0.764	0.052	0.175	0.463	0.682	0.081	0.154	0.304	0.595	0.101
S-KNN	0.186	0.661	0.760	0.052	0.172	0.453	0.675	0.083	0.151	0.298	0.592	0.103
SF-SKNN	0.179	0.636	0.750	0.052	0.165	0.433	0.680	0.082	0.145	0.284	0.593	0.104
AR	0.178	0.631	0.733	0.058	0.164	0.435	0.653	0.103	0.143	0.278	0.517	0.170
IKNN	0.158	0.510	0.796	0.009	0.148	0.363	0.757	0.010	0.131	0.242	0.673	0.010