Information Processing & Management 57.3 (2020): 102228

DOI: https://doi.org/10.1016/j.ipm.2020.102228

Time and Sequence Awareness in Similarity Metrics for Recommendation

Pablo Sánchez, Alejandro Bellogín

Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain

Abstract

Modeling the temporal context efficiently and effectively is essential to provide useful recommendations to users. In this work, we focus on improving neighborhood-based approaches where we integrate three different mechanisms to exploit temporal information. We first present an improved version of a similarity metric between users using a temporal decay function, then, we propose an adaptation of the Longest Common Subsequence algorithm to be used as a time-aware similarity metric, and we also redefine the neighborhood-based recommenders to be interpreted as ranking fusion techniques where the neighbor interaction sequence can be exploited by considering the last common interaction between the neighbor and the user.

We demonstrate the effectiveness of these approaches by comparing them with other state-of-the-art recommender systems such as Matrix Factorization, Neural Networks, and Markov Chains under two realistic time-aware evaluation methodologies (per user and community-based). We use several evaluation metrics to measure both the quality of the recommendations – in terms of ranking relevance – and their temporal novelty or freshness. According to the obtained results, our proposals are highly competitive and obtain better results than the rest of the analyzed algorithms, producing improvements under the two evaluation dimensions tested consistently through three real-world datasets.

Keywords: Recommender Systems, Time-aware, Sequence, Neighborhood, Collaborative filtering

1. Introduction

Recommender Systems (RS) are software tools whose main objective is to suggest items that may be relevant to the users of a system [1, 2]. They are especially significant today because of their ability to deal with the growing massive information on the Internet by filtering data and providing personalized recommendations to the users [3]. For that reason, a large number of companies like Netflix (streaming platform), Spotify (music), LinkedIn (business oriented social network), TripAdvisor (travel-content website), and many more make use of them, in order to adapt to the needs of each user individually. Depending on how these recommendations are made, different types of models can be distinguished. Two of the most well-known approaches include the content-based (CB) recommenders, which suggest items similar to the ones the user

liked in the past [4] and the collaborative filtering (CF), that exploit the preferences between users and items to perform recommendations [1]. However, as each proposal has drawbacks under some circumstances, typically several strategies are combined by creating hybrid approaches in order to alleviate these problems [2].

Beyond these classic algorithms that typically work with user-item-rating tuples or with user/item features, it has been argued recently that incorporating contextual knowledge like location, time, or even the weather can be crucial in order to perform recommendations in some situations [5, 6]. Among the different contexts, temporal information is one of the most interesting ones to be integrated into the recommendation algorithms, due to its facility to be captured and because it usually discriminates better than other dimensions, allowing us to detect patterns in the user behavior and recommend items matching the current user's needs [5, 7]. Nevertheless, the integration of this type of information into other classical strategies such as those based on user or item similarities is not trivial and it has typically been proposed as heuristic filters [8, 7]; because of this, it is one of the main problems we aim to address in this work. On the other hand, other recommendation techniques have emerged to model the temporal context with more or less success, mainly based on Markov Chains [9], Matrix Factorization [10] (or different combinations of methods using these models [11, 12]), and Neural Networks [13, 14, 6]. However, and as discussed in recent works [15], recommendation based on temporal information is not the same as recommendation based on sequences: those methods that exploit the time dimension need the actual time a user interacted with an item, whereas those that exploit sequences only use the order of the interacted items (this order, of course, could be derived from the temporal information, but it could also be derived from any other information that encodes the order).

At the same time, RS evaluation has been traditionally linked to the analysis of the relevance of the recommendations using Information Retrieval (IR) metrics such as Precision, MAP, or nDCG normally in a cross-validation (random) evaluation methodology [16]. Nonetheless, some researchers alerted about the use of more realistic evaluation methodologies by taking the interaction time into account when creating the splits [7]. The use of such methodologies is not straightforward, and there are several options worth of exploration [7], although, in summary, they are characterized by the fact that all the interactions of a particular user in the test set must occur in the original dataset *after* her interactions included in the training set. In this work, since we deal with temporal information and time-aware recommendation algorithms, we will follow some of these temporal evaluation methodologies and analyze their effect on the performance of the studied methods. Furthermore, once that temporal information becomes available, other evaluation dimensions besides relevance can be measured. One of the most obvious and interesting metrics, which has been recently formalized, is the temporal novelty, or *freshness*, of the recommendations [17], which will also be included in our measurements.

Our work. In this paper, we propose to use the temporal information in two independent ways. Firstly, we state that we can define new similarities that consider preferences as temporal sequences, instead of time-agnostic representations of user and item interactions. In this regard, we advocate the use of the Longest Common Subsequence (LCS) algorithm [18] as a similarity metric because of its useful properties that can be

exploited in the context of recommender systems. This is an extension of the approach presented in [19], where we adapted that algorithm to work in the recommendation domain, however the temporal dimension was not considered either in the modeling step nor in the evaluation, in contrast to what we propose and analyze here.

Secondly, we also propose to revisit the neighborhood-based recommenders as ranking fusion algorithms, following and extending the approach we introduced in [20]. Thus, in this new scheme (named as *backward-forward*), each neighbor will provide the target user a list of items by exploiting the last common interaction between the target user and the neighbor, recommending the items that are close to that last interaction. The final ranking is then obtained combining all the neighbors lists of recommendations. Under this perspective, modeling the temporal aspect of user preferences is straightforward – as we shall show here – and provides an intuitive rationale about what is being recommended and why. As a novelty with respect to our previous work, we use different normalization components when fusing the neighbor lists, we explore the use of sequence-aware similarity metrics (both in isolation and integrated in the backward-forward approach), and we compare our proposals against a more complete set of state-of-the-art recommenders in more, larger datasets. Additionally, we reformulate a well-known time-aware similarity metric so that this model could perform better with ranking-oriented evaluation metrics under more realistic scenarios.

Research questions. In order to test the performance of our proposal, we have followed two instances of a time-aware evaluation methodology on three real-world datasets aimed at answering the following research questions: (**RQ1**) Are time- and sequence-aware neighbor-based recommenders competitive against other state-of-the-art algorithms that consider user/item similarities? (**RQ2**) Is it possible to formulate this problem using ranking fusion techniques from Information Retrieval, so that we can incorporate temporal sequences into this formulation? (**RQ3**) How do time- and sequence-aware approaches affect the resulting freshness of the recommended items? Specifically, the **main contributions** of this paper are:

- A new approach that redefines the classical generation of item rankings by considering the order (sequentiality) in neighbor-based recommender systems, borrowing ideas from Aggregated Search and Information Retrieval in general that can be used in combination with any user similarity.
- The adaptation of the Longest Common Subsequence algorithm as a sequence-aware similarity metric on RS in such a way that users are considered more similar when they interact with the items in the same order, together with a modification of a time-aware similarity metric more oriented towards item ranking tasks.
- A thorough comparison between our approach and other classical algorithms used in the recommendation area analyzing different perspectives of evaluation (relevance and freshness) under time-aware evaluation methodologies on three real-world datasets.

Implications. As we will see in the presented results, our proposals obtain competitive results under different evaluation strategies, demonstrating the premise that performance can be improved by reformulating the problem while using time- and

sequence-aware similarities and neighborhoods. In particular, the use of a time-aware similarity metric seems to produce the best outcome despite its simplicity. Furthermore, our approaches usually perform better than other, more complex models, while, at the same time, they remain easy to understand and to explain why a recommendation was produced, a very important aspect of recommender systems nowadays [21].

The remainder of the paper is organized as follows: in Section 2 we present with more details some well-known state-of-the-art recommendation approaches together with a brief introduction about how they are evaluated, also including a definition of the freshness measure we shall use later in this paper. In Section 3, we present our approaches aware of time and interaction sequences, including a modification of a previously proposed time-aware similarity metric, the use of the Longest Common Subsequence as a sequence-aware similarity metric, and a novel neighborhood approach based on rank fusion techniques that integrates sequences seamlessly. In Section 4, we present the evaluation methodology followed in the experiments, including the datasets used and how the optimal parameters were found, and then, in Section 5, we present the results obtained, answering the previous research questions. Finally, Section 6 summarizes the main contributions of this paper and discuss some lines of work that might be addressed in the future.

2. Background

Formally, the recommendation process is sometimes formulated as an optimization problem whose main objective is to suggest the most useful items to users based on their interests and needs [3]:

$$i^*(u) = \arg\max_{i \in \mathcal{I}} g(u, i) \tag{1}$$

that is, the goal is to find the optimal item i^* that maximizes the utility function g which measures the interest of user u on any item i, where $\mathcal{I}$ denotes the complete set of items in the system.

While the objective is the same for every recommendation model, we can distinguish different types of methods depending on how they work with the data and how they make the recommendations. As mentioned before, the most extended ones are content-based (CB) and collaborative filtering (CF) recommenders; while the former exploit the intrinsic features of the items in order to recommend other items similar to the ones the user liked in the past, the latter exploit the information stored in the user $\times$ item matrix in order to perform the recommendations.

In this article, we will focus on CF approaches, specifically in those based on similarity metrics. Among the different types of CF techniques, we explain in detail the neighborhood-based approaches in Section 2.1, then, in Sections 2.2 and 2.3 we present time-aware and sequential recommendation approaches and a brief description of how recommender systems are evaluated nowadays.

2.1. Neighborhood-based approaches

The collaborative filtering family of recommendation algorithms can be divided in two different classes: neighborhood approaches (also called memory-based CF or

k-NN recommenders) and the latent factor models (often called Matrix Factorization approaches or MF). The latter represent both items and users as vectors with the same dimension as the number of latent factors, after that transformation the ratings are predicted by the scalar product of both vectors, which are typically learned using Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) techniques [10]. Neighborhood-based approaches, on the other hand, obtain similarities between users or items and make recommendations based on those similarities.

The classical formulation of the user-based neighborhood recommender is the following:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_u} r_{vi} w_{uv}}{\sum_{v \in N_u} |w_{uv}|} \tag{2}$$

where w_{uv} is the similarity between user u and v and N_u are the user's u neighbors.

The previous formulation is oriented to the rating prediction problem, i.e., it aims to predict a predicted rating $\hat{r}_{ui}$ as close as possible to the real rating. Recently, more attention has been paid to the task of generating an item ranking for a given user. In such scenario, Equation 3 – where the denominator is not used and, hence, the output of such computation is not in the range of ratings because the similarity normalization has been removed – has shown much better performance when evaluated with ranking instead of error metrics [22]:

$$\hat{r}_{ui} = \sum_{v \in N_u} r_{vi} w_{uv} \tag{3}$$

Besides the size of the neighbors, usually represented with the variable k, the performance of neighborhood recommenders critically depend on the similarity metric being used. The most extended approaches are the Cosine Similarity (CS), the Pearson Correlation (PC), and the Jaccard Index (JI):

$$\text{CS}(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2 \sum_{j \in \mathcal{I}_v} r_{vj}^2}} \tag{4}$$

$$\text{PC}(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{5}$$

$$\text{JI}(u, v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u \cup \mathcal{I}_v|} \tag{6}$$

where $\bar{r}_u$ is the average of the ratings used by user u, $\mathcal{I}_u$ denotes the items rated by u, and $\mathcal{I}_{uv}$ represents the items rated by both u and v.

These formulations show different neighborhood-based recommendation approaches from a user-based perspective, item-based approaches can be similarly derived in a complementary way [23]. Nevertheless, although these techniques may not achieve a performance as competitive as other more complex methods in their basic formulation, they have important advantages as efficient real-time personalization, easy implementation, and explainable recommendations [23].

2.2. Time-aware and sequential recommendation

Temporal information in RS has proven to be of vital importance to be able to understand the user behavior and, therefore, several approaches that incorporate this dimension have been proposed in order to produce better recommendations. For example, in [10] the authors incorporate the timestamps of the ratings in the SVD++ minimization formula, in [24] the short-term and long-term preferences of the users are exploited using a temporal graph model, and in [14] a Convolutional Neural Network model is proposed to learn sequential patterns applying convolutional filters to the item embeddings. Another interesting approach related to this topic closer to our paper is defined in [8], where the authors incorporate a time decay (TDec) function in an item-based neighborhood recommender as follows:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_i} r_{uj} \cdot sim(i,j) \cdot f_\lambda(t_{uj})}{\sum_{j \in N_i} sim(i,j) \cdot f_\lambda(t_{uj})} \tag{7}$$

where r_{ui} is the rating given by user u to item i, $sim(i,j)$ denotes the similarity between items i and j, and t_{uj} is the moment when u rated item j. The time function $f_\lambda(t)$ is defined as $f_\lambda(t) = e^{-\lambda t}$, with $\lambda = \frac{1}{T_0}$ (T_0 in days). The authors introduce the exponential function where λ is the decay rate or half-life parameter. It is defined so that old data obtains small weights, which means that larger values of t correspond to older interaction times, where $t = 0$ is the present. It is not clear, however, how such values of t are computed or normalized from the original timestamps, since no other information is provided; in particular, it is not obvious what $t = 0$ actually represents for the authors, is it the last interaction in training? in the whole dataset?

This approach based on a time decay function has also been adapted to user-based recommenders, as shown in [7]:

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum_{v \in N_u} (r_{vi} - \overline{r}_v) \cdot sim(u,v) \cdot e^{-\lambda \cdot (t - t_{vi})}}{\sum_{v \in N_u} sim(u,v)} \tag{8}$$

where we observe that the authors used a mean-centering formulation for the user-based recommender [23]; moreover, in this case the time decay function receives the difference of two timestamps (instead of only one as before): the neighbor's interaction time t_{vi} and the prediction time t. Again, it is not obvious how such time t is used in this formulation, although it is very likely that it would represent the timestamp that appears in the test set, which exposes an unrealistic situation since the recommender would be using a different t for every item in the test set, exploiting information that should remain hidden to the algorithm in the prediction stage.

Leaving aside this time decay similarity function, time analysis of user's preferences has also favored the emergence of algorithms that exploit these user-item interactions as sequences in their models. In this type of *sequential* recommendation researchers model each user in the system as a sequence of actions $S(u) = (S_1, S_2, S_3, \cdots, S_n)$ – taking the same notation as in [12] – where each action corresponds to one item the user has consumed. Thus, the recommendation problem here is to predict the next item to be consumed by the user. While most traditional algorithms rely on the full history of the user, in sequence recommendation only the latest interactions of the user are normally considered (or, at least, the most recent ratings receive more attention than the others).

One classical approach to identify these sequential patterns is to use an L-Markov chain model, where L denotes the number of previous actions that are taken into account in order to make the recommendations. The L-order Markov chain for modeling user sequences can be represented as:

$$P(S_t \mid S_{t-1}, S_{t-2}, \cdots, S_{t-L}) \tag{9}$$

When considering a first order Markov chain ($L = 1$), the probability of choosing item j given the actual item i at the next step, $p(j \mid i)$, is obtained by using maximum likelihood estimation on the item-to-item transition matrix. This model, although simple, is at the core of many approximations. For example, the authors of [11] proposed a more complex approach where each user has its own transition matrix, leading to a global representation of a tensor. Besides, they also incorporated matrix factorization techniques in their probabilistic framework, combining both approaches (MF and Markov Chains) into a single model: Factorizing Personalized Markov Chain (FPMC).

Another interesting approach that model sequences of actions is defined in [12]. In that article, the authors use an L-order Markov Chain making a weighted sum for the short and long term dynamics of the user preferences. The proposed model (named Fossil, for Factorized Sequential Prediction with Item Similarity Models) is a combination of both Factored Item Similarity Models (FISM) and Markov Chains with personalization. According to the reported results, the Fossil approach outperforms many sequential state-of-the-art algorithms, including the FISM techniques, FPMC, and BPRMF model (a matrix factorization technique optimized using BPR, i.e., Bayesian Personalized Ranking), although the researchers did not analyze the performance of other classic recommenders such as standard MF approaches or neighborhood-based algorithms. Nevertheless, both approaches emphasize the use of Markov Chains for their recommendations, neglecting other possible ways of modeling the temporal context, as we propose in this paper.

In addition to the aforementioned algorithms, neural networks techniques have also acquired special relevance recently due to their ability to model sequences. One example is GRU4Rec, proposed in [13], a Recurrent Neural Network session-based algorithm that captures the sequential dependencies between the user preferences in order to make recommendations. Additionally, a Convolutional Neural Network method was proposed in [14], also known as Caser, that exploits both the sequential patterns of the data and the general preferences of the users in order to make the predictions. These techniques are prevalent nowadays in the recommender systems literature, however it should be noted that, in contrast to the main focus of our paper, they do not exploit user or item similarities in their models.

Finally, it is worth mentioning neighborhood approaches applied to other tasks, such as session-based recommendation, typically applied to the e-commerce domain. These methods also exploit the sequential context, such as the sequential k-NN technique from [25] and the STAN model from [26], which have proven to be competitive against other modern algorithms, including those based on deep learning. However, we consider they are out of the scope of this paper as they are tailored for a specific task (i.e., session-based recommendation, as it is the case of the GRU4Rec model), while in this work we aim to exploit the temporal and sequential contexts without any further assumptions.

2.3. Recommender systems evaluation

The aim of RS evaluation is to determine which recommenders (or configurations of recommenders) are better than others based on the results obtained in certain metrics under a specific evaluation methodology. However, in recent years the evaluation paradigm has moved in new, unexplored directions. On the one hand, researchers have realized that evaluating recommender systems based on random splits does not translate well to the real-world scenario where algorithms receive information constantly, either in streaming or in batches, but where the unknown actions will happen *after* the already observed interactions. In fact, this matches the typical definition of recommender systems: *suggesting items to users based on their previous actions*; nonetheless, for many years these systems have been evaluated where the future and the past was randomly mixed. To address these issues, time-aware evaluation methodologies have been proposed and their effects analyzed with time-aware and time-agnostic recommendation algorithms [7].

In [7], a taxonomy of recommendation evaluation protocols or methodologies is presented, where they are classified according to the following conditions: base set (it specifies if the splitting is applied to the whole dataset or to each sub-dataset independently, such as one dataset for each user), ordering (it establishes an ordered sequence for the ratings), and size (criterion to compute the number of ratings to be assigned to each training and test split). As expected, the ordering condition is the one that controls whether an evaluation methodology is sensitive to the temporal dimension, the other two conditions entail different constraints that may produce more or less realistic methodologies, such as *fixed* vs *proportion* size (where all the users in the test set contain exactly the same number of interactions or a ratio of their whole user profile) or *community-centered* vs *user-centered* base set (where all the ratings are used as a whole or each user is considered as a separate dataset).

On the other hand, the score provided by a recommender was traditionally interpreted as an approximation of the rating that the user would give to that item; because of this, researchers started using error metrics like Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) in order to test the quality of their recommenders, where the lower the error, the better. However, some researchers alerted that this kind of metrics were not prepared for real-world problems as they only considered the observed items, not all the items in the collection [27, 28]. Thus, IR ranking metrics like Precision, Recall, MAP, or nDCG became more popular in RS, where the goal now shifted to include as many relevant items as possible in the ranking provided to the user, so that (usually) the higher the value in these metrics, the better the quality of the recommender.

Moreover, despite the importance of relevance in recommendations, there has been a growing awareness on measuring other evaluation dimensions like novelty and diversity, as sometimes producing only accurate recommendations may not surprise or discover new items to the target user [29, 30]. When this principle is applied in a temporal context, the concept of *temporal novelty* or *freshness* arises (i.e., retrieving newer items according to a temporal model). Although there is no general consensus on how to compute freshness in general (see [31, 32, 33]), there has been an attempt recently to define this concept in recommendation [17]. That work takes as starting point a general framework for novelty and diversity evaluation metrics defined in [34], where different

temporal models for the items are incorporated. The authors propose four models to account for the novelty of the items, producing four different definitions for when an item is considered novel. The MIN model (where an item novelty is measured according to the median timestamp of its occurrences in the system) is proposed as the least biased among these four options while keeping most of the desired properties for this type of metric.

3. Sequence-aware neighborhood-based recommenders

We have presented in the previous section many Collaborative Filtering approaches that take into account temporal information to improve the performance of their recommendations, however most of them neglect neighborhood-based approaches, which constitutes our main motivation to study and incorporate such information into those methods. Hence, we now propose three techniques that aim to outperform the current state-of-the-art while integrating and properly modeling temporal information in neighborhood-based recommenders. First, in Section 3.1 we reformulate a time-aware similarity to incorporate some features to improve its reproducibility, practicality, and efficiency, then, in Section 3.2 we define how to use the LCS algorithm as a sequence-aware similarity metric, and in Section 3.3 we propose a novel framework to incorporate sequence-awareness into neighbor-based recommenders by reformulating the problem as rank aggregation. Finally, in Section 3.4 we briefly analyze the complexity of the proposed approaches.

3.1. Revisiting time-aware similarity metrics

As presented in Section 2.2, other researchers have proposed time-aware similarities integrated in a neighborhood-based recommender, either for an item-based approach [8] or for a user-based one [7]. However, these methods have some limitations. As stated before, it is not fully clear what the timestamp used in those similarity computations means or where it is captured from. This raises problems about reproducibility but, more important yet, about their practical utility: if the actual time when the user rated or interacted with a particular item is needed, the usefulness of a recommender system is very limited, since the only missing piece of information is the score the user would give to such an item *at that moment*. This problem statement is inherited from the first studied recommendation tasks, which aimed at predicting the rating given by a user to an item to decrease the system error [3], similar to a classification or regression task in Machine Learning. However, the Recommender System community has evolved and its focus is now on ranking-oriented tasks, such as providing an item ranking to a user that matches the user preferences [27].

Because of this, we propose a reformulation of these time-aware similarity metrics. Our starting point is Equation 8 instead of Equation 7, since we shall work with user-based neighborhood recommenders, however, similar developments could be derived for a time-aware item similarity. Hence, putting all this together, we propose the following formulation for a time-aware user similarity:

$$\hat{r}_{ui} = \sum_{v \in N_u} sim(u, v) \cdot r_{vi} \cdot f_c(t_u^L, t_{v,i}) \tag{10}$$

where $t_{v,i}$ is the time when user v rated item i, t_u^L is the time of the last rating of user u in the training set, and $f_c(\cdot,\cdot)$ is our *soften* time decay function. This function, as in the presented approaches from the literature, is based on an exponential function, which, by default, has the following form: $f_c(t_1,t_2) = e^{-\lambda \cdot \text{diff}(t_1,t_2)}$, where $\text{diff}(t_1,t_2)$ indicates the difference in the same time units as λ (recall that λ is the decay rate and is usually defined in days by default) between two timestamps (i.e., $t_1 - t_2$). A soften value is applied when $\text{diff}(t_1,t_2) < 0$ and $abs(\text{diff}(t_1,t_2)) > \frac{1}{\lambda} \cdot c$, where we force $\text{diff}(t_1,t_2) = -\frac{1}{\lambda} \cdot c$, which bounds the time decay factor to a value of e^c, to avoid large values when the neighbor rated the item further away in the future with respect to the last interaction of the target user. It should be noted that factor c, thus, allows us to control up to which point in the future (c times the period $T_0 = \lambda^{-1}$) all the neighbor interactions will have the same weight.

Hence, in summary, our adaptation of a time-aware user similarity – based on the one defined in [7] – incorporates the following features: it specifies how the timestamps are selected and compared, making easier to reproduce this method but also having in mind a realistic use case scenario where the test set is unknown for the recommendation method. Moreover, the score produced by the method is not bounded in a given rating scale, in line with recent methods that perform better in ranking-oriented tasks [22]; and, because of this last change, a soften function is applied to avoid producing very large score values.

3.2. Sequence-aware similarity metrics

As presented in Section 2.2, there is a family of recommendation approaches that model the temporal information by exploiting the sequences of user interactions. We now propose another user similarity approach for neighbor-based recommenders in this line, where instead of integrating a time decay function or modeling explicitly the actual time each user interacted with any item, we build user sequences based on their historical preferences and define a similarity measure based on those sequences.

With this goal in mind, we propose to adapt the Longest Common Subsequence (LCS) algorithm as a similarity metric between user sequences. The original LCS problem consists on finding the longest common subsequence (that is, a sequence that can be obtained from another sequence by removing, not necessarily consecutive, none or some of its symbols), normally between two strings. In general, such subsequence is not unique even though the length of such subsequence is unique. This problem arises in a number of applications, from text editing to molecular sequence comparisons [18], and more recently in [35] to compare movement paths.

To compute the LCS between two strings X and Y with lengths l_x and l_y, a dynamic programming approach is normally used that fills a matrix $C_{(l_x+1)\times(l_y+1)}$ following this equation:

$$C[i,j] = \begin{cases} 0 & \text{if i = 0 or j = 0} \\ C[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(C[i,j-1], C[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases} \tag{11}$$

where x_i and y_j represent the characters at indexes i and j (starting in 1) of strings X and Y. Hence, the final value in $C[l_x, l_y]$ with be the length of the LCS between the two input strings.

This definition of the problem is not directly applicable to the recommendation domain, since there might be different ways to define the user sequences as strings, mostly depending on the symbols used to represent those sequences. That is why in [19] we analyzed this problem and proposed a generic method to transform the user history of ratings into sequences. However, in that work the user sequences neglected the use of temporal information both in the model – user sequences were ordered according to a global item ordering, instead of in a user basis depending on how the user interacted with the items – and in the evaluation – random splits were used, not time-aware evaluation methodologies as we present later. In any case, once these sequences are created, they can be exploited by any string distance algorithm such as Jaro-Winkler or Levenshtein [36], although in this paper we shall focus on the LCS algorithm.

Hence, in this work we propose to apply the framework presented in [19] to temporal data, by creating a sequence for each user based on the items such user has interacted with, sorting those sequences from older to newer interactions, where the symbols of the sequences consist of the item ids (this allows us to apply the same procedure to both explicit and implicit datasets). Once these sequences are generated, the standard LCS algorithm can be applied, and the similarity between two users would correspond to the LCS between their respective sequences (sim_1). To obtain bounded values in the $[0, 1]$ interval as classical similarity metrics, we test the following normalizations as presented in [37, 19]:

$$\text{sim}_1(u, v) = \text{LCS}(u, v) \tag{12}$$

$$\text{sim}_2(u, v) = \frac{\text{sim}_1(u, v)^2}{|\mathcal{I}_u| \cdot |\mathcal{I}_v|} \tag{13}$$

$$\text{sim}_3(u, v) = \frac{2 \cdot \text{sim}_1(u, v)}{|\mathcal{I}_u| + |\mathcal{I}_v|} \tag{14}$$

We note that these normalization functions allow to scale the raw LCS value (sim_1) by considering the lengths of the user sequences. Besides, they can be obtained from the original $\text{sim}_1(u, v)$ almost with no extra cost, so three different recommendation scores can be computed at the same time.

It is important to mention that using LCS as a similarity metric has additional advantages against other classical similarities. It does not only fit well the sequential recommendation problem, but it also allows us to work with repetitions (where users have rated the same item more than once), while other similarities cannot take this effect into account. This is interesting because, even though in traditional recommendation a user does not rate the same item more than once [23], in some situations the fact that a user has consumed the same item several times is an indicator that the item is of interest to the user (e.g., music or tourism recommendation) [15].

3.3. Sequence-aware item selection in neighborhood approaches

Neighborhood-based recommenders can be revisited as ranking fusion algorithms where each neighbor contributes a ranking (of potentially relevant items for the target user) and the goal of the recommender system is to combine these rankings into one final output [20]. In terms of Aggregated Search [38, 39], each neighbor would be

denoted as a *judge* (in IR these judges are usually different search engines) who gives a complete ordering of all the alternative items to be ranked; each of these rankings is denoted as τ, and the final fused ranking is $\hat{\tau}$. Formally, the process of rank aggregation is divided into normalization (where the scores or the ranks of τ are normalized into a common scale, $w^{\tau}(i)$, for each item i) and combination (where the normalized weights $w^{\tau}(i)$ are combined into one fused score).

There are several methods for each of these stages, see [39] for an in-depth review of the most prominent ones. Interestingly, by taking the identity normalizer for the scores ($w^{\tau}(i) = \tau(i)$) and the so-called CombSUM combiner (where the normalized weights are simply added for each item) with a preference weight for each ranking equals to the similarity between the neighbor and the target user, we obtain a linear combination of the normalized weights, which is equivalent to the neighborhood-based recommender formulation. In fact, when we take into account the ratings of the neighbors, the "score" of user u to item i using CombSum and the identity normalizer produces the formulation in Equation 3. In this situation, each ranking τ is composed of the item-rating pairs rated by a particular neighbor, excluding, as a standard practice in the community, those items already rated by the target user in training. Further extensions and *ad-hoc* modifications could be made to these normalizers and combiners so that other formulations of this problem – such as mean-centering or Z-score normalization [23] – are obtained.

Once we have reformulated the problem of neighborhood-based recommendation as a ranking fusion technique, we now describe how we can incorporate temporal information – and, in particular, sequence-awareness – in the process. The main idea is that each neighbor will find which is her last common interaction with the target user and will create a ranking of her candidate alternatives iterating around that item, taking into account the order (sequence) in which she rated each of those alternatives. Although in this case we are not taking into account the actual moment of the interaction (i.e., we can end up recommending items from a neighbor whose last common item was rated long time ago), we can easily improve this approach by filtering out neighbors whose last common item was rated before a certain threshold date; in this paper we will not explore this option and leave it as future work. Note, however, that the temporal aspect is considered twice in this model: it is used to involve the target user (through the last common interaction) in setting the actual moment (context) of the recommendation and, at the same time, to exploit the temporal sequence (order) in which the neighbor interacted with the items.

In the following, we present our model, consisting in a method to compute the last common interaction and different strategies to exploit the order of the neighbor ratings to produce a ranking with candidate items from each neighbor; since these strategies iterate around the last common interaction, as we shall see, we call this technique the backward-forward (or BF) approach. Then, a single recommendation ranking is generated by normalizing and combining the different item rankings produced by each neighbor. There are several methods that can be applied in these stages. In the normalization step, the most common strategies are the default normalization (Def) where the normalized item value is the same as the item score, the standard normalization (Std) where a min-max normalization is applied to the score of every item in the list, and the rank-sim normalization (Rks) in which the normalized value of an item is inversely

proportional to its position in the ranking. On the other hand, for the combination scheme the most extended approach, as mentioned above, is the CombSUM combiner that simply computes a weighted average of the normalized weights of each item using an additional preference weight for each ranking. Other approaches such as CombMIN, CombMAX, and CombMNZ (where the minimum, maximum, and the number of nonzero values is used in the combination step) have been proposed [39], but these techniques are not included in our analysis.

As described before, the first step in our backward-forward approach is to obtain the last common interaction item between two users u and v. This is computed as follows:

$$n^*(u;v) = \max_k \left(i_k \in \mathcal{I}_u^t : i_k \in \mathcal{I}_v^t \right) \tag{15}$$

where $\mathcal{I}_u^t$ are the items rated by user u ordered by timestamp in ascending order:

$$\mathcal{I}_u^t = \text{sort}\left(\mathcal{I}_u, t\right) = \left(i_k^t\right)_{k=1}^{|\mathcal{I}_u|}, \text{ with } t\left(i_k^t\right) < t\left(i_{k+1}^t\right) \tag{16}$$

Once we have calculated this last common interaction (that, from Equation 15, is not symmetrical, since $n^*(u;v) \neq n^*(v;u)$ because it looks for the preferences of the first user in the second user historical preferences), we propose the following strategies to build the lists with candidate items from each neighbor (with respect to the target user u): (a) take the m items that have been rated after that common interaction ($L_m^+(v;u)$) (b) take the m items that have been rated before that common interaction ($L_m^-(v;u)$) (c) take the m items that have been rated before and after that last common interaction alternating them ($L_m^a(v;u)$) (d) take the m items that have been rated after the last common interaction first and concatenate them the m that have been rated before ($L_m^\pm(v;u)$). Formally:

$$\text{Let } \mathcal{I}^t(v;u) = \text{sort}\left(\mathcal{I}_v - \mathcal{I}_u, t\right)$$

$$L_m^+(v;u) = \left(i_k^t\right)_{n^*(v;u)}^{n^*(v;u)+m}, i_k^t \in \mathcal{I}^t(v;u) \tag{17}$$

$$L_m^-(v;u) = \left(i_k^t\right)_{n^*(v;u)-m}^{n^*(v;u)}, i_k^t \in \mathcal{I}^t(v;u) \tag{18}$$

$$L_m^a(v;u) = \left(a_k^t, b_k^t\right)_{k=1}^{\max(|L_m^+(v;u)|,|L_m^-(v;u)|)} \quad a_k^t \in L_m^+(v;u), b_k^t \in L_m^-(v;u) \tag{19}$$

$$L_m^\pm(v;u) = \left(L_m^+(v;u), L_m^-(v;u)\right) \tag{20}$$

Therefore, for each neighbor we obtain a list $L(v;u)$ with all the candidate items from that neighbor, which will be later normalized and combined, as explained before, to produce a single ranking, containing the recommendations for the target user u. Since these items are related to the last interactions between users u and v, they can be interpreted as "the recent difference" in interaction history between these users. It is important to note that any similarity can be used in such scheme when obtaining the last common interaction between the users or generating the lists with candidate items, since these two steps do not depend on the user similarity, although the neighbors might be different or the weights used at the combination step if the similarity changes.

Figure 1: Example of user interactions in the movie domain. Items denoted with an asterisk ($*$) and a yellow border correspond to the last common interaction between u and each neighbor, those with a $-$ symbol as superscript and a red border are included in $L_2^-(v;u)$, whereas those with a $+$ as superscript and a green border in $L_2^+(v;u)$.

In summary, when using this formalization, we obtain a neighbor-based recommendation model equivalent to classical formulations that can further incorporate the temporal information under different models. Nonetheless, it should be mentioned how possible it is to combine into a single model this approach with the previous proposed methods. While using sequence-aware similarity metrics together with sequence-aware neighborhoods (i.e., the proposed BF technique) is straightforward, since any similarity metric could be used to compute the neighborhood and weight each neighbor; using the time-aware k-NN approach within the BF algorithm is not trivial. The reason for this is that, as we can see in Equation 10, the temporal decay function in the proposed time-aware k-NN approach depends on the last timestamp of the user and on the timestamp of the neighbor's rating to the target item, however that information is not available in our current formulation for the sequence-aware neighborhood. In any case, it is possible to combine sequence-aware similarity metrics with the time-aware algorithm proposed, since the latter method could work with any user similarity metric and, in particular, with the proposed LCS-based similarity.

Finally, let us illustrate the whole process with an example shown in Figure 1 using the movie domain. For the sake of simplicity, we do not include the user's rating, so the reader should assume that all sequences correspond to items that the user has equally liked. In the movie domain, the temporal component is usually determining, since newer movies tend to be consumed more often than older ones. In our example, user u is the target user, and we represent three neighbors v_1, v_2, and v_3, where v_1 and v_3 have three items in common whereas v_2 shares four items with the target user. According to these interactions, the candidate items generated with respect to the different strategies presented before (limited to size 2) will be (considering that $n^*(v_1;u) = i_9, n^*(v_2;u) =$

$i_{10}, n^*(v_3; u) = i_7$):

$$L_2^+(v_1; u) = (i_{14}, i_{13}), L_2^-(v_1; u) = (i_6, i_2), L_{1,1}^{\pm}(v_1; u) = (i_{14}, i_6), L_{1,1}^{a}(v_1; u) = (i_{14}, i_6)$$
$$L_2^+(v_2; u) = (i_{12}, i_{13}), L_2^-(v_2; u) = (i_2), L_{1,1}^{\pm}(v_2; u) = (i_{12}, i_2), L_{1,1}^{a}(v_2; u) = (i_{12}, i_2)$$
$$L_2^+(v_3; u) = (i_{12}, i_{15}), L_2^-(v_3; u) = (i_5, i_6), L_{1,1}^{\pm}(v_3; u) = (i_{12}, i_5), L_{1,1}^{a}(v_3; u) = (i_{12}, i_5)$$

Note that in this case, as we are selecting only two items, L_m^a and $L_m^{\pm}$ are equivalent. Suppose now that items i_{12}, i_{13}, and i_{14} are in the test set (as mentioned before, newer items are more likely to be chosen by user u). A standard neighborhood-based recommender (that does not consider any temporal aspect of the data) would probably recommend item i_2 whereas in our approach, we are favoring more recent items, like the movie i_{12}. In fact, i_2 only appears in our method once for strategy $L^{\pm}$ and twice for L^-. Moreover, we believe that moving forward from the last common interaction is more useful in terms of recommendation performance – especially for novelty purposes – than moving backwards, this is evidenced by the strategies L^+ and $L^{\pm}$ that recommend i_{13} and i_{14}; because of this, we ignore the $L^{\pm}$ strategy, since in some preliminary experiments we found it was equivalent to L^+, due to very sparse data.

3.4. Complexity and scalability

In this section, we discuss the algorithm complexity and potential scalability issues of the proposed approaches. According to the analysis presented in [23], the memory and time complexity of user-based nearest neighbor recommenders is $O(U^2)$ and $O(U^2 p)$, where U is the number of users in the system and p is the maximum number of ratings per user (similar derivations can be found for the item-based variations). Additionally, it should be considered that the similarity computation is much less expensive than the worst-case complexity indicated before, since most users only interact with few items. Moreover, at prediction time we can consider that, for every user, we need to access to the preferences of the k neighbors, hence, in terms of time complexity this can be summarized as $O(Ukp)$.

Based on this, we argue that the time-aware similarity metric proposed in Section 3.1 incurs in no extra cost with respect to the standard similarity computations. On the other hand, this is not so clear for the sequence-aware similarity metric based on LCS presented in Section 3.2. Although the computation of the LCS algorithm is an NP-hard problem, it can be calculated in quadratic time thanks to the dynamic programming solution presented before, in particular, $O(U^2 p^2)$, since time and space complexity of the standard algorithm is $O(l_x l_y)$ for two strings X and Y with lengths l_x and l_y, although it is possible to obtain even lower bounds for this problem [18]. Besides, it can be performed offline with no impact on prediction time; hence, as stated in [40], although the sequence-aware similarity might be more expensive, this is not necessarily a critical aspect when applied to recommendation due to the following reasons:

- The computation of LCS similarities can be done offline, before the actual recommendations are requested.

- The computed similarities can be stored at training time, just as we would do with other similarity metrics.
- The normalized versions of the LCS-based similarities can be computed based on the value obtained by the sim_1 metric; therefore, the use of these normalizations do not increase the execution or training times.
- As with other similarity metrics, the computation can be easily parallelized by distributing the load in a user basis (for instance, the similarities associated to a particular user could be calculated in a specific thread, since there is no inter-user dependency).

Nevertheless, regarding the sequence-aware approach to select items from neighbors, we identify two critical steps that are different from the traditional k-NN: computing n^* and obtaining the list L, both impacting the prediction time, independently of the memory or time needed to compute the similarities. Whereas the generation of list L is actually less expensive (if the value m is small) since only m items are considered, finding the last common interaction between the users is an additional step that should be done in a user basis. This step, however, can be performed in linear time if the preferences of both users are sorted (as evidenced in Equation 15), hence its cost is $O(p \log p)$, which should be done for every pair of users, resulting in a slightly higher time complexity than the one for the standard user-based k-NN: $O(Ukp \log p)$.

4. Experimental Settings

4.1. Datasets

We experiment with three datasets from different domains where all the provided ratings have been timestamped: Epinions, Foursquare, and MovieTweetings. The Epinions dataset contains the purchases of different products made by users. Foursquare, on the other hand, comes from the tourism domain (the items in this case are touristic venues or points-of-interest, as they are usually denoted); finally, MovieTweetings is a movie dataset where IMDb ratings have been collected from Twitter [41]. For both Epinions and MovieTweetings, the users have indicated their tastes with different values (1 to 5 in the case of Epinions and 0 to 10 in the case of MovieTweetings), low values representing bad opinions about the items in both cases. On the other hand, the Foursquare dataset contains implicit feedback and we only have information about whether a user visited a specific venue. The first two datasets were also used by the authors of [12] and can be obtained from that paper. MovieTweetings is a dataset that is constantly updated, we decided to take a specific snapshot of approximately 600K ratings[1].

For all datasets, we have filtered the original data by removing all the repeated preferences, taking into account only the newest preference when a user rated the same item more than once. This processing only had an effect on Epinions and Foursquare,

[1]This snapshot can be obtained here: `https://github.com/sidooms/MovieTweetings/tree/7a1fae8d9faafe90c084065fe8bb924bff2c3cd2`.

Table 1: Statistics from the datasets used in the experiments.

Dataset	k-core	Users	Items	Ratings	Density	Scale	Unique times	Time interval
Epinions	2	22k	15k	76k	0.022%	1-5	4k	Jan 2001 - Nov 2013
Foursquare	5	16k	3k	105k	0.205%	1	102k	Dec 2011 - Apr 2012
MovieTweetings	5	15k	8k	519k	0.399%	0-10	517k	Feb 2013 - Apr 2017

since MovieTweetings does not include repetitions. Additionally, we performed a k-core in all datasets, this means that we removed every user or item that did not have at least k preferences, where $k = 5$ in MovieTweetings and Foursquare datasets, and $k = 2$ for Epinions. Because the Epinions dataset is much more sparse than the other two, when a 5-core is performed in this dataset, no ratings remain, that is why we needed to lower k in this case. The final statistics of the processed datasets are shown in Table 1.

We would like to draw the attention on the difficulty of obtaining datasets with realistic temporal information. In some preliminary analyses, we discarded Movielens (one of the most popular datasets in the area but where some researchers alerted about its non-realistic timestamps [42]) and Amazon reviews datasets[2], because too many items were consumed at the same time (in the exact same second) by a user, which does not make sense for either performing a temporal evaluation or running a time-aware recommendation algorithm. The datasets used in this paper are not perfect either (see column 'Unique times' in Table 1 which, ideally, should be close to the number of interactions in every case), but are the best ones we could find with a large number of users and items and a decent number of interactions. We hypothesize the larger difference between 'Unique times' and 'Ratings' columns in Epinions– evidencing more "repeated" timestamps – may be caused by some users making purchases in sessions, where all the items consumed in the same session are logged with the same timestamp. Figure 2 shows the temporal distribution of the ratings in the three tested datasets, and how they evolve over time.

4.2. Evaluation methodology

Since we deal with temporal information and its effect on recommender systems, we should use time-aware evaluation methodologies such as the ones introduced in Section 2.3. In this paper, we use a time-dependent rating order (the timestamps of the test split for each user occur after those of the training split) in two evaluation methodologies: one with a user-centered base set and a fixed size condition (the last 2 actions of each user with at least 6 actions are included in the test split) and another with a community-centered base set and a proportion-based size condition (the same timestamp is used for all the users, in such a way that we retain the data corresponding to the 20% of the most recent rating times for testing, and the rest for training). We name the first configuration as *Fix* and the second as *CC*. There are obvious differences between these two evaluation methodologies: whereas in *CC* the test set is always (for every user) after the training set, in *Fix* this may not be the case; besides, (almost) every user is included in the test set of *Fix* but this is not the case for *CC*. Because of these

[2]Available here: `http://jmcauley.ucsd.edu/data/amazon/`.

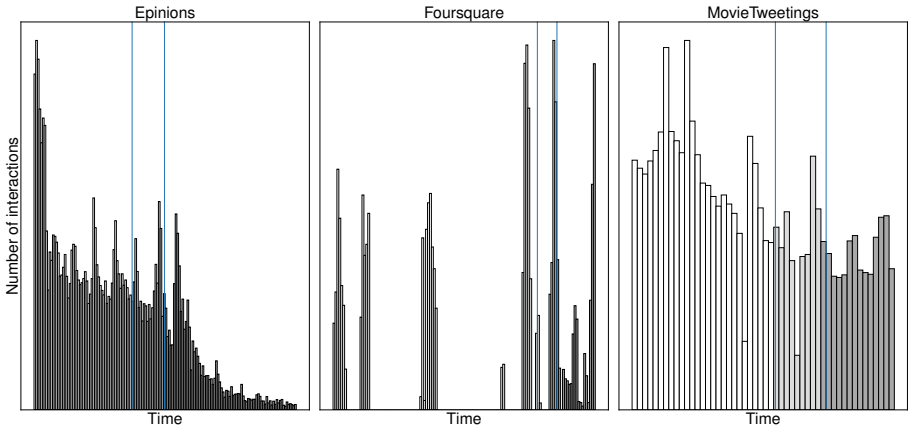

Figure 2: Distribution of ratings in the datasets for the CC split. Epinions (left), Foursquare (middle), and MovieTweetings (right). The first blue line represents the ratings used for the parameter optimization (validation training set) and the second one represents the data used in the reported results (full training set).

features, the *CC* methodology represents better a real environment, where there are some users that may not be active at some point, whereas with the *Fix* evaluation we can analyze the recommendations for all the users, not only the most active (or active in the last period when the dataset was collected) ones.

Furthermore, according to the terminology in [16], we report the results obtained following the *TrainItems* strategy to select the candidate items to be ranked by each algorithm; that is, a ranking is generated for each user by predicting a score for every item that has at least one interaction in the training set; as it is standard, we remove those items already rated in training by the user from each candidate list (in a user basis). We then compute standard Information Retrieval metrics on the rankings, considering as relevant every item rated at least with a 5 and a 9 in the test split for Epinions and MovieTweetings, whereas for Foursquare every item in the user's test set is considered as relevant. More specifically, we report the results obtained using Precision, nDCG, and MAP and we also show the results in terms of freshness using the MIN metric (measuring freshness of the items with respect to their median timestamp). All metrics are reported using a cutoff of 5. In every case, higher values means more accurate/fresh recommendations. These evaluation metrics have been implemented using the RankSys framework[3].

4.3. Parameter tuning

Unless stated otherwise, we report tuned versions of the recommendation algorithms. To allow a fair comparison of the algorithms and to avoid overfitting, we performed the following procedure (further details in Appendix A): we tested the parameters shown in Table A.4 in the training subset of the data; based on the optimal

[3]Standard IR metrics are available at `https://github.com/RankSys/RankSys`, whereas freshness metrics are available at `https://bitbucket.org/PabloSanchezP/timeawarenoveltymetrics`.

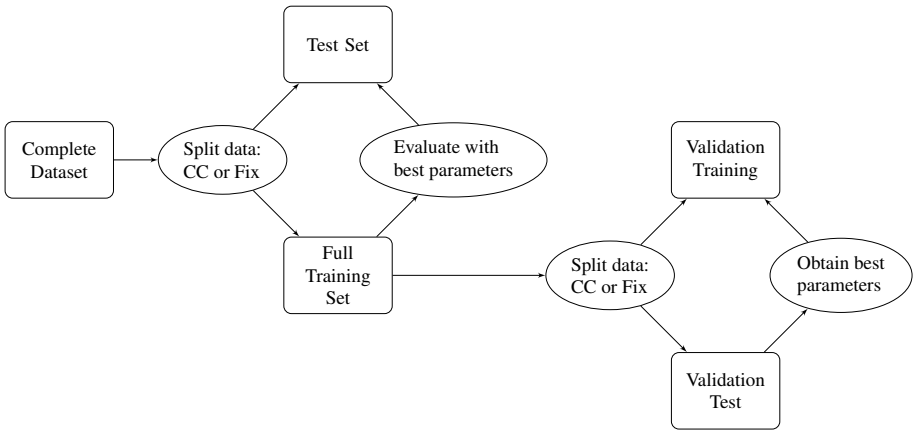

Figure 3: Methodology followed to obtain optimal parameters and the performance values of their corresponding recommendation algorithms.

parameters found there (selected according to their values of nDCG@5), we evaluate the recommenders using the remaining data (test subset, where the whole training subset is used to train the algorithms) and report the results obtained here. Note that, as summarized in Figure 3, the training subset is further divided into a validation training and a validation test, so that the parameters are selected according to the performance obtained in the validation test (for an algorithm trained on the validation training), in such a way that the test subset is only looked at to report the final results.

Observe that the optimal parameters are thus found for a smaller subset (validation training) than the one used for the reported results (test subset) – this is a common practice in Machine Learning with temporal information [43]. As a consequence, some of the parameters found might not be globally optimal, since the test set is not used to select the parameters. In fact, for some methods the optimal parameters in validation produced invalid results when used in the full training set and we had to test the next best set of parameters. For further details about the range of the parameters and the actual values of the optimal parameters found for each recommender, see Appendix A.

Furthermore, since our training and test subsets have been created using a time-aware evaluation methodology, we follow a similar procedure when creating the validation splits mentioned above. In particular, for the CC methodology we split the full training data leaving the 20% more recent items in the validation test, and the rest in the validation training. For the Fix methodology, the last 2 items of those users with at least 4 interactions are sent to the validation test, and the remaining ratings are used to create the validation training. Additionally, we show in Figure 2 the temporal evolution of the rating distribution in all the datasets for the CC methodology in the three subsets: validation training, validation test, and test.

4.4. Evaluated recommenders

In our experiments, we include an array of recommender systems that combine implementations provided in different libraries with our own code. These algorithms

can be classified into non-personalized baselines, standard collaborative filtering approaches, time-aware/sequential baselines, and our proposed approaches.

First, as non-personalized baselines, we report a popularity recommender (**Pop**) that ranks the items according to the number of interactions received in the training set; we also experimented with a random recommender, but we did not include it in the final results due to its low performance in all methodologies and datasets.

Second, as standard CF methods, we use the classical item-based (**IB**) and user-based (**UB**) k-NN recommenders as explained in Section 2.1, and the MF recommender using ALS optimization (denoted as **HKV** [44]), both implemented in the RankSys library. We also include the **BPRMF** algorithm proposed in [45], as provided by the MyMediaLite library[4].

Third, we include as sequential recommenders the **Fossil** approach from [12], where we adapted the code provided by the authors[5] to generate a ranking for each user, following the methodology already described, and two algorithms based on Markov Chains (denoted as **MC** and **FPMC**) also available in that code. Since the authors evaluated these methods with the AUC metric, we modified the original code in order to generate a ranking from the recommendations provided to a user, so that any IR metric could be used instead. We also consider Caser as a sequential recommender based on Neural Networks. This method (Convolutional Sequence Embedding Recommendation Model) was proposed in [14] and combines the sequential patterns inferred from the data and the general preferences of the users to make recommendations; we use the implementation provided by the authors[6].

Finally, we test the following combinations of our approaches (all of them implemented on top of the RankSys framework): our adaptation of the temporal decay method (**TDec**, Equation 10) as an example of a time-aware similarity metric, a sequence-aware user-based similarity with a classical UB approach denoted as **sUB** (where LCS with a temporal ordering is an example of such similarity), and our backward-forward approach using a standard k-NN similarity (**BFUB**) and a sequence-aware similarity (**BFsUB**).

5. Empirical evaluation

In this section, we analyze the recommenders presented before according to the explained evaluation methodology. First, in Section 5.1 we present and discuss the results obtained when using a global temporal split, then, we do the same process for the per user temporal split in Section 5.2. Finally, Section 5.3 shows further analyses regarding our proposed approaches, where we present a detailed sensitivity analysis based on the parameters of the backward-forward method.

5.1. Performance analysis: global temporal split

In Figure 4, we show the results obtained in the three datasets under the CC evaluation methodology. We present three ranking metrics (MAP, nDCG, and Precision)

[4]Available at `http://www.mymedialite.net`.

[5]Available at `https://drive.google.com/file/d/0B9Ck8jw-TZUEeEhSWXU2WWlocOk/view`.

[6]We use the Python implementation available here: `https://github.com/graytowne/caser_pytorch`.

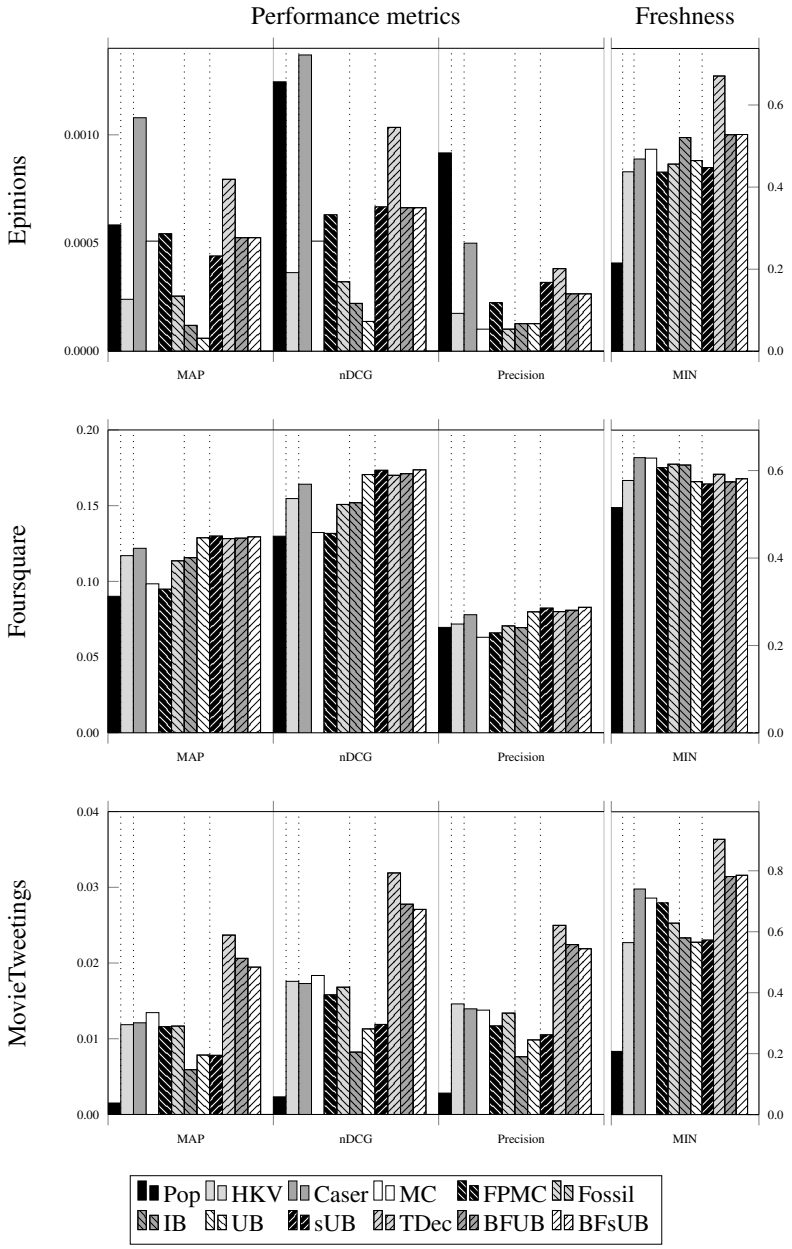

Figure 4: Performance of recommenders using the global temporal split (CC evaluation methodology) at cutoff 5. Recommendation methods are separated by vertical dotted lines into five groups (non-personalized, matrix factorization, sequential baselines, classical nearest-neighbors, and the proposed time- and sequential-aware approaches) to facilitate the identification of the different approaches.

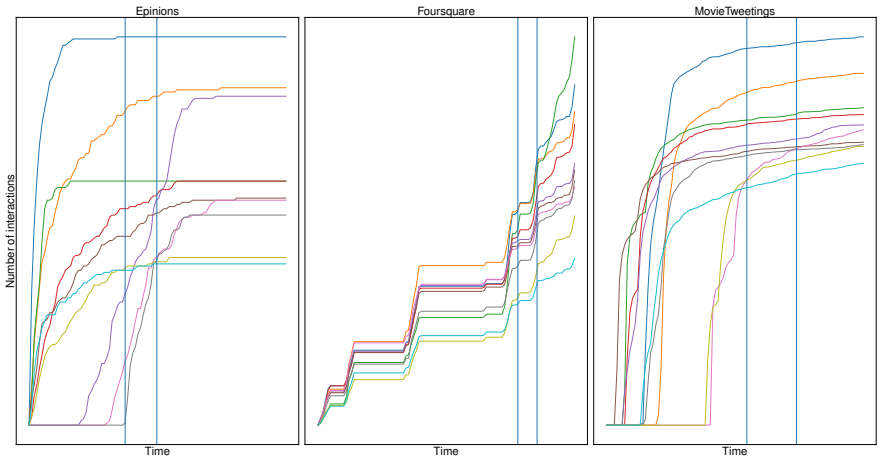

Figure 5: Rating evolution of the ten most popular items in the datasets. Epinions (left), Foursquare (middle), and MovieTweetings (right).

and one freshness metric (MIN) as explained in Section 4.2. Let us focus first on the relevance metrics, in particular, in the low results achieved in this kind of metrics, especially in both Epinions and MovieTweetings. There are several reasons that explain this behavior: on the one hand, we are using a high relevance threshold (we only consider as relevant items those rated with a 5 in Epinions and 9 or 10 in MovieTweetings). However, for Foursquare this is not so critical since in that case we deal with implicit information, hence, all items in the test set are relevant for the user, which is why these metrics obtain higher values, i.e., around 0.2. On the other hand, since we have performed a temporal split, we may end up having users and items in the test set that do not appear in the training data, thus, modeling user profiles under these circumstances becomes a difficult task.

It is also important to consider the sparsity of the datasets, since, obviously, high sparsity datasets leads to more difficult conditions for the algorithms, which translates into very low performance results, as we observe in Epinions (whose density is only 0.022%, in comparison with MovieTweetings with 0.399%, as reported in Table 1). In any case, results in Epinions are very sensitive to the ranking metric, since a different optimal recommender is obtained depending on the evaluation metric. Nonetheless, it should be noted that the relative differences in the figure for Epinions are much smaller (one or two orders of magnitude) with respect to the other datasets.

Furthermore, a strong popularity bias is exposed in the Epinions dataset, since this is the second best recommender in terms of relevance. This bias is a well-known effect in the recommender systems literature [46, 47], where this type of not-personalized recommender sometimes produces very good results only due to statistical, aggregated effects. In our case, this high performance in such dataset can be attributed to the effect we show in Figure 5, where some of the most popular items (in training) tend to receive more interactions in the test set, something also noticeable in the Foursquare dataset, where this recommender is also very competitive with respect to other personalized

algorithms. However, in Epinions this effect is combined with a small coverage of the rest of the algorithms (because the density in this dataset is very small), which results in a recommender that produces good (in statistical terms) recommendations for many people. Nonetheless, this bias is observed to a lower extent when the Fix methodology is used (see Section 5.2) and in the other datasets (mostly in Foursquare, although Pop never achieves the best performance).

We also observe a low performance of time-agnostic recommenders (UB, IB, and HKV) in Epinions. Although these algorithms are typically very competitive (i.e., using a random evaluation partition in other domains such as movies or music), they are clearly at a disadvantage when using a temporal split. This might be attributed to the fact that both matrix factorization and neighborhood-based methods take into account all user ratings in the same way, but usually the most modern ratings tend to be more important as they better represent the users' tastes in the future (test split). In the other datasets a similar situation is shown, although less extreme for some cases, for instance, HKV is at the level of sequential recommenders in Foursquare and MovieTweetings, whereas UB is among the best recommenders in Foursquare. On the other hand, IB tends to be the worst performing ones in all the cases. We note that BPRMF is not shown in the figures because it is always outperformed by another method of the same family, HKV.

It is surprising that some of the sequential recommenders (MC, FPMC, and Fossil) do not produce good results in any of the domains. One possible reason is that in the original paper [12] no ranking metrics were tested, only the AUC metric was used, the same metric these algorithms aim to optimize; moreover, they did not use any evaluation threshold for any dataset, so that every item in the test set was considered relevant [12]. Besides, they focused on predicting the next item to consume, as they left for testing the recommender just the last item rated by the user in the dataset. At the same time, state-of-the-art recommenders such as nearest-neighbor were ignored in the experimental comparison in that paper. Similarly, Caser does not perform as well as in the original paper [14] except in Epinions, where it is the best recommender. We hypothesize that we have not replicated those positive results mostly because the authors performed a heavy pruning of the dataset – removing all the cold-start users (those with less than 15 or 10 interactions, depending on the dataset) – and they did not include more traditional recommenders (such as k-NN or MF) which are very competitive in our experiments. In any case, sequential recommenders remain competitive, even beating other baselines such as HKV in Epinions, or IB and UB in MovieTweetings; in these two datasets, Caser and MC show the best performance among the sequential baselines, whereas Caser and Fossil obtain the best results in Foursquare.

On the other hand, our time-aware and sequential-aware approaches (sUB, TDec, BFUB, and BFsUB) tend to perform better than most of the other approaches, except in Epinions due to the aforementioned popularity bias and the large difference between the number of ratings and unique times (see Table 1), which, as discussed before, is an indication about the number of repeated timestamps and, hence, about the confidence of the temporal dimension captured in that dataset. For instance, sUB is always better than UB, evidencing that using a sequence-aware similarity metric allows to better capture the preferences of the user when a temporal evaluation is performed. Moreover, BFUB and BFsUB are also better than UB – recall these three methods are based

on neighbors –, although depending on the dataset BFUB or BFsUB performs better, but, in general, the differences between these methods are not significant. Finally, the TDec approach shows very positive results, being always among the top performing recommenders, despite its simplicity. Hence, we can conclude that **using temporal information in a recommender when performing a CC temporal split improves the ranking quality**.

In terms of temporal novelty (freshness) we notice some interesting patterns. First, we see that in Epinions and MovieTweetings datasets, TDec, BFsUB, and BFUB obtain better results than the rest of the recommenders. This is an expected result since these methods give more importance to the items that are closer to the test split or to the last interacted items by the neighbors, hence, it is more likely that these recommendations are more temporally novel than those from time-agnostic approaches. Nevertheless it is also relevant that this effect does not happen in Foursquare, where most of the recommenders achieve a comparable level of the freshness metric MIN. This could be attributed to the fact that the timespan in this dataset is rather short (less than a year), whereas in the other datasets many years of interactions are available, so the temporal patterns that we may be learning and evaluating upon in Foursquare may not be discriminating enough.

5.2. Performance analysis: per user temporal split

Let us now analyze the results obtained when using the per user temporal split (Fix evaluation methodology). Figure 6 shows the results for this scenario using the same notation presented before. We observe in Epinions a considerable change in the behavior of the recommenders with respect to when the CC methodology was used: the performance values are higher now and also the trend in the algorithms is different, the popularity bias (now less strong) being one of the most obvious changes. On the other hand, results in Foursquare remains quite stable, whereas in MovieTweetings the trend in terms of types of recommenders (non-personalized, matrix factorization, sequential, nearest-neighbors) is very similar, although some methods perform now better than before (such as BFsUB, which is now better than BFUB, the opposite of what we observed in Figure 4). This different behavior might be attributed to the actual differences already explained about these two methodologies: in Fix all test users are also in the training set and more ratings are available from those users in training, since only users with previous interactions are considered in test. A related aspect that could also affect the behavior of the recommenders is how realistic these datasets are; in fact, we have detected that in Epinions there are many repeated timestamps (meaning that a user has consumed more than one item in the same moment), as shown in Table 1, thus, exploiting the temporal information in this dataset might be less reliable for some users. Probably for this reason, results in Epinions are very sensitive to the ranking metric, as already discussed in the previous section.

Regarding the behavior of the recommendation approaches, we observe some situations that are very different to what we found using the global temporal split. Both IB and HKV do not perform as bad as before, in fact, they are better or at the same level than sequential recommenders. Interestingly, the baselines based on sequences (MC, FPMC, and Fossil) do not achieve very good results, even though this evaluation methodology is closer to the one presented in the original paper [12]; however, as discussed before, we should consider that in that paper the authors reported results based

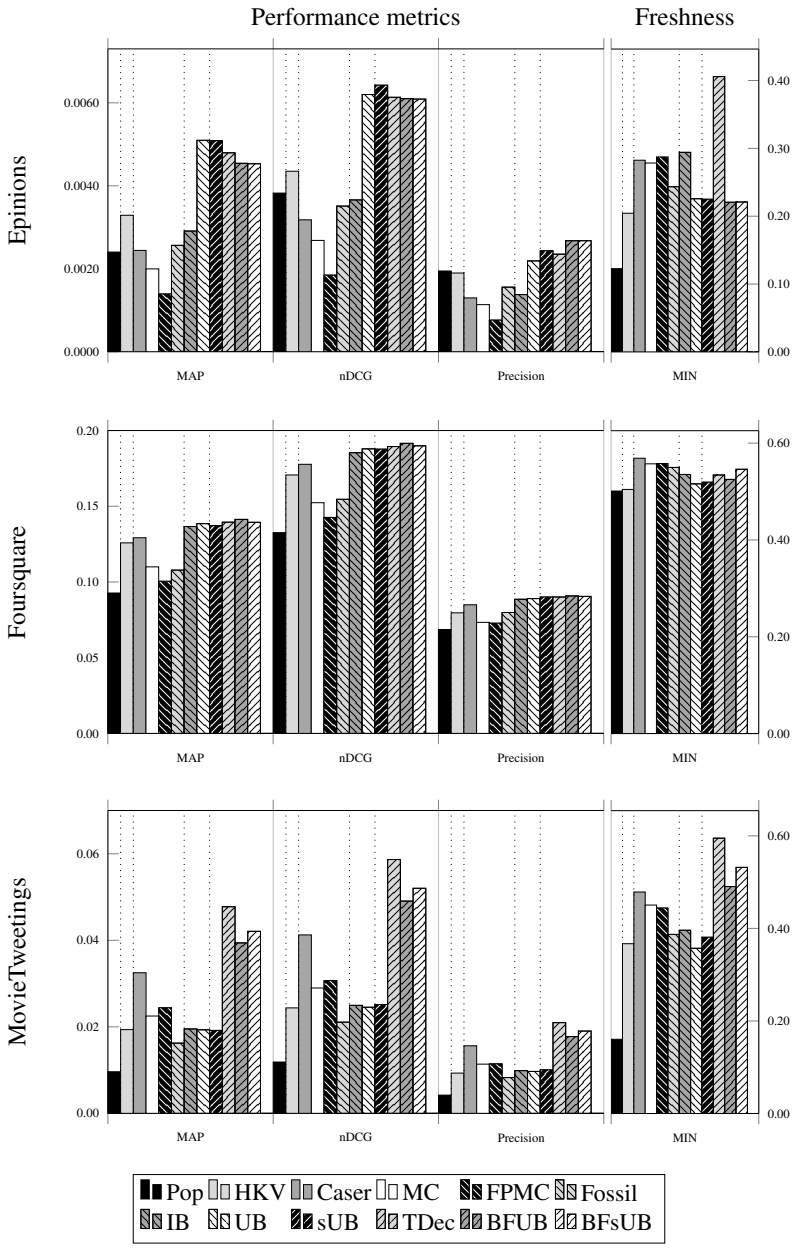

Figure 6: Performance of recommenders using the per user temporal split (Fix evaluation methodology) at cutoff 5. Same vertical lines as in Figure 4.

Table 2: Sensitivity of Backward-Forward components using the global temporal split (CC evaluation) at cutoff 5. Column Nrm shows normalization strategies as defined in Section 3.3, whereas column Wgt indicates whether to consider the weight of the neighbors (W) or not (U). Best results for each Sim in bold. Columns 10·MAP and 10·nDCG in Epinions show the corresponding metric values but multiplied by 10 due to the original values being too small (because the dataset is very sparse).

Recommender			Epinions			Foursquare			MovieTweetings		
Sim	Nrm	Wgt	10·MAP	10·nDCG	MIN	MAP	nDCG	MIN	MAP	nDCG	MIN
BFUB	Def	U	**0.005**	0.007	0.528	0.129	0.171	0.574	**0.021**	**0.028**	0.781
		W	**0.005**	**0.008**	0.539	**0.130**	**0.173**	0.577	**0.021**	**0.028**	**0.782**
	Std	U	**0.005**	0.006	0.534	0.129	0.171	0.574	0.019	0.026	0.769
		W	0.005	0.007	**0.543**	**0.130**	**0.173**	0.577	0.019	0.026	0.755
	Rks	U	0.002	0.003	0.536	0.127	0.169	0.577	0.017	0.024	0.770
		W	0.004	0.007	0.533	0.127	0.169	**0.579**	0.018	0.025	0.771
BFsUB	Def	U	**0.005**	**0.007**	0.528	0.128	0.172	0.579	**0.019**	**0.027**	0.785
		W	0.001	0.002	**0.555**	**0.129**	**0.174**	0.581	**0.019**	0.026	0.787
	Std	U	0.003	0.003	0.534	0.128	0.172	0.579	0.017	0.024	0.819
		W	0.003	0.003	0.552	**0.129**	**0.174**	0.581	0.017	0.024	**0.822**
	Rks	U	0.002	0.003	0.536	0.128	0.172	0.582	0.018	0.024	0.774
		W	0.001	0.002	0.538	**0.129**	0.173	**0.583**	0.017	0.024	0.775

on the AUC metric whereas we are evaluating with other ranking metrics. Moreover, MC is not the best performing sequential baseline anymore, but FPMC (in MovieTweetings) and Fossil (in Epinions and Foursquare) always obtain better results.

We now note that one aspect where both evaluation methodologies agree on is that the proposed time-aware and sequential-aware recommenders are always the best performing approaches. Here, we observe this is especially clear in MovieTweetings, where time-agnostic recommenders obtain very bad results; however, in the other datasets, our proposed approaches (TDec, BFUB, BFsUB) achieve the highest values, sometimes tied with UB (Epinions) or with IB and UB (Foursquare). Hence, we conclude that **using temporal information in a recommender when performing a Fix temporal split does not necessarily improves the ranking quality but it helps achieving high performance**, mostly due to inconsistencies in the way this evaluation methodology deals with the preference data.

Finally, when analyzing the temporal novelty of the recommendations, we observe an strange behavior in the Epinions dataset. The BF approaches achieve lower values than most of the baselines; this effect, despite being counter-intuitive, can be explained if we bear in mind that this evaluation methodology is not considering a global split for all users, so there can be users that stop being active at the beginning of the system lifetime and their recommendations may be less fresh than those from other users. The TDec recommender is not affected by this effect, as in that case the method considers the timestamp, not the specific interaction sequence of the user. Nevertheless, we note that for the MovieTweetings dataset it is clear that the time-aware neighborhood recommenders are able to improve the performance of their corresponding time-agnostic approaches in terms of both relevance and freshness.

Table 3: Sensitivity of Backward-Forward components using a per user temporal split (Fix evaluation). Same notation as in Table 2.

Recommender			Epinions			Foursquare			MovieTweetings		
Sim	Nrm	Wgt	10·MAP	10·nDCG	MIN	MAP	nDCG	MIN	MAP	nDCG	MIN
BFUB	Def	U	0.045	**0.061**	0.221	0.139	0.188	0.526	**0.039**	0.048	0.488
		W	0.043	0.054	0.227	**0.141**	**0.192**	0.525	**0.039**	**0.049**	**0.490**
	Std	U	0.043	**0.061**	0.233	0.139	0.188	0.526	0.030	0.038	0.477
		W	**0.047**	0.058	**0.244**	**0.141**	**0.192**	0.525	0.029	0.036	0.477
	Rks	U	0.038	0.051	0.231	0.137	0.187	**0.527**	0.033	0.042	0.473
		W	0.040	0.052	0.236	0.140	0.190	**0.527**	0.034	0.042	0.475
BFsUB	Def	U	0.045	0.061	0.221	0.138	0.188	**0.550**	0.042	0.052	0.529
		W	**0.052**	0.064	0.227	0.138	0.189	0.546	0.042	0.052	0.532
	Std	U	0.042	0.059	**0.233**	0.138	0.188	**0.550**	**0.045**	**0.055**	0.527
		W	**0.052**	**0.067**	0.231	0.138	0.189	0.546	**0.045**	**0.055**	0.529
	Rks	U	0.038	0.052	0.231	0.138	0.188	0.546	0.038	0.047	0.532
		W	0.049	0.063	0.229	**0.140**	**0.191**	0.546	0.038	0.048	**0.536**

5.3. Sensitivity of backward-forward components

To better understand the behavior of the different components in the proposed neighborhood approach where items are selected based on a sequence-aware strategy, we show in Tables 2 (for CC) and 3 (for Fix) the results for the best configurations using a sequence-aware similarity metric (LCS) against a classic neighborhood similarity (BFsUB vs BFUB) using our backward-forward approach. In these tables, we compare these similarities when using the normalization strategies (Nrm) defined in Section 3.3 and testing whether to consider the weight of the neighbors or not (column Wgt).

Based on these results, we observe that the performance achieved by the sequential similarity is usually not the best one except in a few cases (Epinions and MovieTweetings with the Fix strategy), however, the differences are, in general, small. We hypothesize that considering sequences by using the BF method is enough to capture the user preferences, and by also using a sequential similarity we impose too many constraints on the algorithm to find proper, valid neighbors.

Moreover, regarding the other components of the BF approach, we observe that usually the best results are obtained with the Default (Def) aggregation function in both evaluation strategies, although other functions do not really produce very different results; however, the use of the neighbor similarity to weight her contribution is important and tends to obtain much better results when it is used (W), except in Epinions and MovieTweetings using BFsUB in CC.

Finally, in terms of freshness we can see that, in general, BFsUB reports higher results than BFUB. This is an expected behavior since, when selecting the neighbors, we give more importance to the users that have rated the same items in the same order, hence, the items that are recommended at the end of the sequences are more likely to be retrieved. If the sequences are long enough, the neighbors will represent better the trends in the system. Nevertheless, we can see that we do not find a general trend with respect to the normalization functions, although the use of the neighbor similarity to weight the neighbor ranking improves the results, as observed when analyzing the relevance metrics.

We conclude that, although the proposed approach takes many parameters, some of them do not contribute much to the final performance and could be considered as fixed, such as the weighting component (in general it is better to consider the neighbor similarity as a weight) or the normalization function (where the Default function usually obtains very good results).

We would like to note that, even though the proposed sequence-aware neighborhood model (with or without a sequential similarity) usually does not beat the time-aware similarity as it was defined here, we believe both approaches are complementary and could serve different purposes. While the sequence-aware neighborhoods are easy to understand and explain why the recommendations are being generated, this is not so straightforward to achieve with the time-aware similarity which, on top of that, is very sensitive to different parameters such as the decay function or the parameters of the soften function, which will need to be tuned appropriately for every new dataset. Besides, we consider that the proposed sequence-aware models could work better in an online environment since, as indicated above, the timestamps collected in these datasets do not always reflect the exact moment in which the user consumed the items. Nevertheless, we also argue that these two approaches could be combined into a sequence-aware neighborhood where neighbors are computed by performing a time-aware similarity metric. However, we leave this possibility as a future work.

6. Conclusions and future work

In this paper, we have presented a new formulation for neighborhood-based recommendation (backward-forward approach) that allows to integrate the temporal dimension based on rank fusion techniques seamlessly and successfully, according to the reported experiments. Besides, we have also presented the use of LCS as a sequence-aware similarity metric, which can further benefit from our proposed backward-forward approach. Additionally, we have reformulated the time-aware similarity metrics to incorporate features aiming at improving their usefulness and efficiency.

We present experiments on three datasets, under two different evaluation methodologies, one considering a global temporal split for all users and another that selects the most recent interactions as the test set. We have found that the results depend on the selected methodology, although the presented approaches are competitive in both of them, in constrast with some of the baselines that only perform well in one methodology. The proposed approaches also outperform recent state-of-the-art algorithms specifically tailored for sequential recommendation.

More specifically, when formulating the recommendation problem as an aggregation of several rankings and introducing the temporal dimension in the process, the performance clearly improves, up to a 30% with respect to another neighbor-based recommender without using the temporal component and up to a 75% with respect to a sequential baseline. Furthermore, by using a realistic and ranking-oriented time-aware similarity metric with a standard, time-agnostic neighbor-based recommender, we also obtain very good results in almost every evaluated scenario, which evidences the importance of time-awareness when modeling the user behavior in a recommender system.

In summary, according to the presented results the three research questions stated in the introduction have been positively answered. We have shown that the time-

aware and sequence-aware neighborhood recommenders are able to outperform other state-of-the-art recommenders, and in particular those that consider user/item similarities (RQ1); not only time-agnostic approaches but also other sequential recommenders based on Markov Chains, in both relevance and freshness evaluation dimensions. Moreover, we have also seen many differences in terms of performance results between the CC and the Fix strategies; for instance, while the first one represents a more realistic environment, its results are lower than the ones obtained with the Fix methodology. On the other hand, even under the Fix methodology (which does not follow a realistic evaluation protocol) these approaches are able to learn the user preferences better.

We have also observed the utility of the LCS algorithm as a sequence-aware similarity metric and the methods based on sequences to select items in neighborhood approaches. In this sense, the backward-forward approach outperforms most of the baselines in all situations (RQ2), except the proposed reformulation of a time-aware similarity metric, modified from its original definition in [8, 7], that tends to be among the top-performing algorithms. Nonetheless, we believe that even when the proposed sequence-aware approaches do not outperform this time-aware similarity metric, they are more intuitive and easier to understand, in the sense that they produce recommendations directly by interpreting the users interactions as a temporal sequence.

With respect to the temporal novelty of recommendations, it is clear that, in general, time-aware neighborhood-based recommenders outperform the rest of the baselines in terms of freshness (RQ3). This might be an expected result *a posteriori*, but it was not so clear before the experiments, in part because the metric we are using was proposed recently and, hence, it has not been fully analyzed [17], but also, because as we have shown, this metric is sensitive to the evaluation methodology. We want to emphasize that this is the first time an exhaustive temporal evaluation has been performed using recommendation algorithms from different types (matrix factorization, sequential, time-aware) on datasets whose temporal information is as realistic as possible; because of this, it becomes very important to understand – as we show here – that it is possible to improve both relevance and temporal novelty when making recommendations in a temporal scenario, which translates into presenting relevant or interesting items that are also fresh to the user.

As a consequence of these results, we advocate for simple models that properly exploit the temporal dimension. The theoretical and practical implications of our findings can be summarized as that, by learning how these sequence- and time-aware recommender systems work, new recommenders that better model the temporal context may arise. Moreover, since complex techniques or algorithms with many parameters do not perform, in general, as good as our proposed approaches, researchers and practitioners should focus on simple techniques, while more effort should be devoted on modeling the temporal and sequence information.

The results obtained open up several possibilities for the future. First, since the backward-forward framework proposed is general enough to work with any rank aggregation function, we plan to explore the use of alternative aggregation functions – such as those based on the score distribution [48] – when integrated in our proposal. Furthermore, an exhaustive analysis – with more datasets, baselines such as SVD++ with temporal information [10] or other Neural Networks techniques [13, 49, 50], and

other evaluation methodologies – should be made to better understand each component of the proposed models. For instance, the number of items allowed to be selected before and after the last common interaction, together with alternative definitions for sequence-aware similarity metrics. As an example, we aim to extend the proposed similarity based on LCS by exploiting other dimensions to build the sequence upon (such as item features, ratings, or combinations thereof) or even by applying filters to select those items that have been rated with a higher value than a specific threshold to create the user sequences, as recently proposed in [40]. Furthermore, it would be interesting to observe the impact on users' online behavior once they receive the recommendations, as it was recently analyzed for social networks in [51]. Besides, by performing a validation step to obtain the best parameters of the recommenders and optimize their performance, we observed how difficult it was to find consistent results (from the validation subset to the full dataset), especially in temporal splits as the ones used here; because of this, we aim to analyze these issues in more detail in the future to obtain guidelines or theoretical guarantees that the parameters learned using a temporal validation split would fit well using the complete data, since we believe this is the most realistic way to tune the parameters, as if it was done in a real-world system.

Acknowledgements

This work has been funded by the Ministerio de Ciencia, Innovación y Universidades (reference TIN2016-80630-P) and by the European Social Fund (ESF), within the 2017 call for predoctoral contracts. The authors thank the reviewers for their thoughtful comments and suggestions.

Appendix A. Reproducibility

We now present more details about how the recommenders were parametrized in order to ease the reproducibility of our experiments. Firstly, Table A.4 shows all the parameters tested for each recommender (i.e., we evaluated an instance of the recommender for every possible value of every parameter of the table, except those marked as fixed, which denote parameters that were not tuned). The best parameters, as explained in Section 4.3, were obtained for both methodologies (CC and Fix) according to the highest nDCG@5 values found by those recommenders in the validation test (see Figure 3). These parameters are included in Tables A.5 and A.6 for the CC and Fix methodologies, respectively. All the necessary scripts and source code to replicate the results, together with the value of the optimal parameters, can be found in the following Bitbucket repository: PabloSanchezP/BFRecommendation.

It should be noted that some of the algorithms start with a random initialization so the results may change. To reduce this effect, we decided to repeat 5 times each execution and calculate the average, so the best parameters are selected according to the average performance. This applies to BPRMF, HKV, Fossil, MC, and FPMC approaches.

Table A.4: Tested recommenders and parameters to be optimized. For BPRMF, the parameters k, λ_u, λ_i, λ_0, and λ_j correspond to MyMediaLite's parameters num_factors, reg_u (regularization for user factors), reg_i (regularization for positive item factors), bias_reg (regularization for the bias term) respectively, and reg_j (regularization for negative item factors). For Caser, the parameters T, d, nh, L, nv, ac_conv, ac_fcs, drop_rate correspond to the number of targets (T), number of latent dimensions (d), number of horizontal filters (nh), length of the sequence (L), number of vertical filters (nv), activation functions for convolutional layers (ac_conv) and for fully-connected layers (ac_fc), and the drop ratio when performing dropout (drop_rate). For the rest of the notation, SJ, VC, and VJ denote SetJaccard, VectorCosine, and VectorJaccard (as named in RankSys) respectively and L_m^+ and L_m^- represent the forward and backward indexes to consider when using the BF approach and n_iters represent the number of iterations of the model.

Rec	**Parameters to optimize (in parenthesis, those fixed)**
Pop	None
HKV	$k = \{10, 50, 100\}$, $\alpha = \{0.1, 1, 10, 100\}$, $\lambda = \{0.1, 1, 10\}$ (n_iters = 20)
BPRMF	$k = \{10, 50, 100\}$, $\lambda_u = \lambda_i = \{0.0005, 0.001, 0.0025, 0.005, 0.01, 0.1\}$, $\lambda_0 = \{0, 0.5, 1\}$ ($\lambda_j = \{\lambda_u/10\}$, n_iters = 50, l_rate = 0.05, UniformUserSampling = false, WithReplacement = false)
Caser	T = {1, 2}, d = {10, 50}, nh = {4, 16} (L = 2, nv = 4, drop_rate = 0.5, ac_conv = relu, ac_fc = relu, batch_size = 512, $l_2 = 10^{-6}$, n_iters = 30, l_rate = 0.003)
MC	$k = \{2, 5, 10, 20\}$, $\lambda = \{0.1, 0.2\}$
FPMC	$k = \{2, 5, 10, 20\}$, $\lambda = \{0.1, 0.2\}$
Fossil	$k = \{2, 5, 10, 20\}$, $\lambda = \{0.1, 0.2\}$, $L = \{1, 2, 3\}$
IB	sim = {SJ, VC, VJ}, $k = \{5, 10, \ldots, 100\}$
UB	sim = {SJ, VC, VJ}, $k = \{5, 10, \ldots, 100\}$
sUB	$k = \{5, 10, \ldots, 100\}$, $\text{LCS}_n = \{\text{sim}_1, \text{sim}_2, \text{sim}_3\}$ (sim = {LCS})
TDec	sim = {SJ, VC, VJ}, $k = \{5, 10, \ldots, 100\}$, $\lambda = \{0.1, 0.05, 0.02, 0.01\}$ (c=2)
BFUB	$L_m^- = \{2, 5, 10\}$, $L_m^+ = \{2, 5, 10\}$, $k = \{5, 10, \ldots, 100\}$, sim = {SJ, VC, VJ}, Wgt = {W, U }, Nrm = {Def, Std, Rks }
BFsUB	$L_m^- = \{2, 5, 10\}$, $L_m^+ = \{2, 5, 10\}$, $k = \{5, 10, \ldots, 100\}$, $\text{LCS}_n = \{\text{sim}_1, \text{sim}_2, \text{sim}_3\}$, Wgt = {W, U }, Nrm = {Def, Std, Rks }

Table A.5: Optimal parameters found for the CC evaluation methodology. Notation as in Table A.4.

Rec	**Epinions**	**Foursquare**	**MovieTweetings**
HKV: k, α, λ	50, 100, 1	10, 10, 0.1	10, 10, 0.1
BPRMF: k, λ_u=λ_i, λ_0	10, 0.1, 0.5	100, 0.0005, 0	100, 0.005, 0.5
Caser: T, d, nh	2, 10, 4	2, 50, 4	1, 10, 16
MC: k, λ	2, 0.2	10, 0.1	5, 0.1
FPMC: k, λ	20, 0.2	5, 0.1	2, 0.1
Fossil: k, λ, L	20, 0.1, 3	5, 0.1, 1	10, 0.1, 2
IB: sim, k	VJ, 100	VC, 70	VJ, 5
UB: sim, k	VJ, 70	SJ, 100	VC, 70
sUB: k, LCS_n	50, sim_1	100, sim_1	100, sim_2
TDec: sim, k, λ	SJ, 60, 0.05	SJ, 100, 0.01	SJ, 90, 0.02
BFUB: L_m^-, L_m^+, k, sim, Wgt, Nrm	0, 2, 50, SJ, U, Def	10, 10, 100, SJ, U, Def	0, 10, 90, VJ, U, Def
BFsUB: L_m^-, L_m^+, k, LCS_n, Wgt, Nrm	0, 2, 50, sim_3, U, Def	5, 5, 100, sim_1, W, Def	0, 10, 80, sim_3, U, Def

Table A.6: Optimal parameters found for the Fix evaluation methodology. Notation as in Table A.4.

Rec	**Epinions**	**Foursquare**	**MovieTweetings**
HKV: k, α, λ	100, 1, 1	10, 10, 10	10, 0.1, 0.1
BPRMF: k, λ_u=λ_i, λ_0	100, 0.005, 0	100, 0.001, 0	100, 0.0005, 1
Caser: T, d, nh	2, 50, 4	2, 50, 4	1, 50, 16
MC: k, λ	5, 0.1	20, 0.1	20, 0.1
FPMC: k, λ	20, 0.1	10, 0.1	2, 0.1
Fossil: k, λ, L	5, 0.1, 1	5, 0.2, 1	20, 0.1, 2
IB: sim, k	VJ, 90	VC, 100	VC, 5
UB: sim, k	SJ, 90	VC, 100	SJ, 100
sUB: k, LCS_n	90, sim_3	100, sim_1	50, sim_1
TDec: sim, k, λ	SJ, 90, 0.1	VC, 100, 0.01	SJ, 100, 0.1
BFUB: L_m^-, L_m^+, k, sim, Wgt, Nrm	10, 10, 100, SJ, U, Def	5, 5, 100, VC, W, Def	0, 10, 100, SJ, W, Def
BFsUB: L_m^-, L_m^+, k, LCS_n, Wgt, Nrm	10, 10, 100, sim_3, U, Def	10, 10, 100, sim_1, W, Rks	10, 10, 70, sim_1, W, Def

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender systems: Introduction and challenges, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 1–34.

[2] R. D. Burke, Hybrid web recommender systems, in: P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.), The Adaptive Web, Methods and Strategies of Web Personalization, Vol. 4321 of Lecture Notes in Computer Science, Springer, 2007, pp. 377–408.

[3] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 734–749.

[4] M. de Gemmis, P. Lops, C. Musto, F. Narducci, G. Semeraro, Semantics-aware content-based recommender systems, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 119–159.

[5] G. Adomavicius, A. Tuzhilin, Context-aware recommender systems, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 191–226.

[6] Q. Liu, S. Wu, D. Wang, Z. Li, L. Wang, Context-aware sequential recommendation, in: F. Bonchi, J. Domingo-Ferrer, R. A. Baeza-Yates, Z. Zhou, X. Wu (Eds.), IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain, IEEE, 2016, pp. 1053–1058.

[7] P. G. Campos, F. Díez, I. Cantador, Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols, User Model. User-Adapt. Interact. 24 (1-2) (2014) 67–119.

[8] Y. Ding, X. Li, Time weight collaborative filtering, in: O. Herzog, H. Schek, N. Fuhr, A. Chowdhury, W. Teiken (Eds.), Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005, ACM, 2005, pp. 485–492.

[9] G. Shani, D. Heckerman, R. I. Brafman, An mdp-based recommender system, Journal of Machine Learning Research 6 (2005) 1265–1295.

[10] Y. Koren, R. M. Bell, Advances in collaborative filtering, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 77–118.

[11] S. Rendle, C. Freudenthaler, L. Schmidt-Thieme, Factorizing personalized markov chains for next-basket recommendation, in: M. Rappa, P. Jones, J. Freire, S. Chakrabarti (Eds.), Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010, ACM, 2010, pp. 811–820.

[12] R. He, J. McAuley, Fusing similarity models with markov chains for sparse sequential recommendation, in: F. Bonchi, J. Domingo-Ferrer, R. A. Baeza-Yates, Z. Zhou, X. Wu (Eds.), IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain, IEEE, 2016, pp. 191–200.

[13] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, in: Y. Bengio, Y. LeCun (Eds.), 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[14] J. Tang, K. Wang, Personalized top-n sequential recommendation via convolutional sequence embedding, in: Y. Chang, C. Zhai, Y. Liu, Y. Maarek (Eds.), Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018, ACM, 2018, pp. 565–573.

[15] M. Quadrana, P. Cremonesi, D. Jannach, Sequence-aware recommender systems, in: T. Mitrovic, J. Zhang, L. Chen, D. Chin (Eds.), Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization, UMAP 2018, Singapore, July 08-11, 2018, ACM, 2018, pp. 373–374.

[16] A. Said, A. Bellogín, Comparative recommender system evaluation: benchmarking recommendation frameworks, in: A. Kobsa, M. X. Zhou, M. Ester, Y. Koren (Eds.), Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014, ACM, 2014, pp. 129–136.

[17] P. Sánchez, A. Bellogín, Time-aware novelty metrics for recommender systems, in: G. Pasi, B. Piwowarski, L. Azzopardi, A. Hanbury (Eds.), Advances in Information Retrieval - 40th European Conference on IR Research, ECIR 2018, Grenoble, France, March 26-29, 2018, Proceedings, Vol. 10772 of Lecture Notes in Computer Science, Springer, 2018, pp. 357–370.

[18] A. Apostolico, String editing and longest common subsequences, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages: Volume 2. Linear Modeling: Background and Application, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 361–398.

[19] A. Bellogín, P. Sánchez, Collaborative filtering based on subsequence matching: A new approach, Inf. Sci. 418 (2017) 432–446.

[20] A. Bellogín, P. Sánchez, Revisiting neighbourhood-based recommenders for temporal scenarios, in: M. Bieliková, V. Bogina, T. Kuflik, R. Sasson (Eds.), Proceedings of the 1st Workshop on Temporal Reasoning in Recommender Systems co-located with 11th International Conference on Recommender Systems (RecSys 2017), Como, Italy, August 27-31, 2017., Vol. 1922 of CEUR Workshop Proceedings, CEUR-WS.org, 2017, pp. 40–44.

[21] N. Tintarev, J. Masthoff, Explaining recommendations: Design and evaluation, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 353–382.

[22] F. Aiolli, Efficient top-n recommendation for very large scale binary rated datasets, in: Q. Yang, I. King, Q. Li, P. Pu, G. Karypis (Eds.), Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013, ACM, 2013, pp. 273–280.

[23] X. Ning, C. Desrosiers, G. Karypis, A comprehensive survey of neighborhood-based recommendation methods, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 37–76.

[24] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, J. Sun, Temporal recommendation on graphs via long- and short-term preference fusion, in: B. Rao, B. Krishnapuram, A. Tomkins, Q. Yang (Eds.), Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010, ACM, 2010, pp. 723–732.

[25] M. Ludewig, D. Jannach, Evaluation of session-based recommendation algorithms, User Model. User-Adapt. Interact. 28 (4-5) (2018) 331–390.

[26] D. Garg, P. Gupta, P. Malhotra, L. Vig, G. Shroff, Sequence and time aware neighborhood for session-based recommendations: STAN, in: B. Piwowarski, M. Chevalier, É. Gaussier, Y. Maarek, J. Nie, F. Scholer (Eds.), Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019, ACM, 2019, pp. 1069–1072.

[27] H. Steck, Evaluation of recommendations: rating-prediction and ranking, in: Q. Yang, I. King, Q. Li, P. Pu, G. Karypis (Eds.), Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013, ACM, 2013, pp. 213–220.

[28] S. M. McNee, J. Riedl, J. A. Konstan, Being accurate is not enough: how accuracy metrics have hurt recommender systems, in: G. M. Olson, R. Jeffries (Eds.), Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006, ACM, 2006, pp. 1097–1101.

[29] P. Castells, N. J. Hurley, S. Vargas, Novelty and diversity in recommender systems, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Recommender Systems Handbook, Springer, 2015, pp. 881–918.

[30] M. Karimi, D. Jannach, M. Jugovac, News recommender systems - survey and roads ahead, Inf. Process. Manage. 54 (6) (2018) 1203–1227.

[31] N. Lathia, S. Hailes, L. Capra, X. Amatriain, Temporal diversity in recommender systems, in: F. Crestani, S. Marchand-Maillet, H. Chen, E. N. Efthimiadis, J. Savoy (Eds.), Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010, ACM, 2010, pp. 210–217.

[32] S. Chou, Y. Yang, Y. Lin, Evaluating music recommendation in a real-world setting: On data splitting and evaluation metrics, in: 2015 IEEE International Conference on Multimedia and Expo, ICME 2015, Turin, Italy, June 29 - July 3, 2015, IEEE Computer Society, 2015, pp. 1–6.

[33] H. Wang, A. Dong, L. Li, Y. Chang, E. Gabrilovich, Joint relevance and freshness learning from clickthroughs for news search, in: A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, S. Staab (Eds.), Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012, ACM, 2012, pp. 579–588.

[34] S. Vargas, P. Castells, Rank and relevance in novelty and diversity metrics for recommender systems, in: B. Mobasher, R. D. Burke, D. Jannach, G. Adomavicius (Eds.), Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011, ACM, 2011, pp. 109–116.

[35] P. Yan, D. D. Zeng, Clustering customer shopping trips with network structure, in: Proceedings of the International Conference on Information Systems, ICIS 2008, Paris, France, December 14-17, 2008, Association for Information Systems, 2008, p. 28.

[36] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios, Duplicate record detection: A survey, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 1–16.

[37] F. T. de la Rosa, M. T. G. López, R. M. Gasca, Analysis and visualization of the DX community with information extracted from the web, in: K. V. Andersen, J. K. Debenham, R. R. Wagner (Eds.), Database and Expert Systems Applications, 16th International Conference, DEXA 2005, Copenhagen, Denmark, August 22-26, 2005, Proceedings, Vol. 3588 of Lecture Notes in Computer Science, Springer, 2005, pp. 726–735.

[38] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, Rank aggregation methods for the web, in: V. Y. Shen, N. Saito, M. R. Lyu, M. E. Zurko (Eds.), Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001, ACM, 2001, pp. 613–622.

[39] M. E. Renda, U. Straccia, Web metasearch: Rank vs. score based rank aggregation methods, in: G. B. Lamont, H. Haddad, G. A. Papadopoulos, B. Panda (Eds.), Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), March 9-12, 2003, Melbourne, FL, USA, ACM, 2003, pp. 841–846.

[40] P. Sánchez, A. Bellogín, Building user profiles based on sequences for content and collaborative filtering, Inf. Process. Manage. 56 (1) (2019) 192–211.

[41] S. Dooms, A. Bellogín, T. D. Pessemier, L. Martens, A framework for dataset benchmarking and its application to a new movie rating dataset, ACM TIST 7 (3) (2016) 41:1–41:28.

[42] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, TiiS 5 (4) (2016) 19:1–19:19.

[43] A. Catalina, A. Torres-Barrán, J. R. Dorronsoro, Satellite based nowcasting of PV energy over peninsular spain, in: I. Rojas, G. Joya, A. Català (Eds.), Advances in Computational Intelligence - 14th International Work-Conference on Artificial Neural Networks, IWANN 2017, Cadiz, Spain, June 14-16, 2017, Proceedings, Part I, Vol. 10305 of Lecture Notes in Computer Science, Springer, 2017, pp. 685–697.

[44] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy, IEEE Computer Society, 2008, pp. 263–272.

[45] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, BPR: bayesian personalized ranking from implicit feedback, in: J. A. Bilmes, A. Y. Ng (Eds.), UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009, AUAI Press, 2009, pp. 452–461.

[46] D. Jannach, L. Lerche, I. Kamehkhosh, M. Jugovac, What recommenders recommend: an analysis of recommendation biases and possible countermeasures, User Model. User-Adapt. Interact. 25 (5) (2015) 427–491.

[47] A. Bellogín, P. Castells, I. Cantador, Statistical biases in information retrieval metrics for recommender systems, Inf. Retr. Journal 20 (6) (2017) 606–634.

[48] R. Manmatha, T. M. Rath, F. Feng, Modeling score distributions for combining the outputs of search engines, in: W. B. Croft, D. J. Harper, D. H. Kraft, J. Zobel (Eds.), SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA, ACM, 2001, pp. 267–275.

[49] T. Donkers, B. Loepp, J. Ziegler, Sequential user-based recurrent neural network recommendations, in: P. Cremonesi, F. Ricci, S. Berkovsky, A. Tuzhilin (Eds.), Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017, ACM, 2017, pp. 152–160.

[50] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T. Chua, Neural collaborative filtering, in: R. Barrett, R. Cummings, E. Agichtein, E. Gabrilovich (Eds.), Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017, ACM, 2017, pp. 173–182.

[51] S. A. M. Falavarjani, F. Zarrinkalam, J. Jovanovic, E. Bagheri, A. A. Ghorbani, The reflection of offline activities on users online social behavior: An observational study, Inf. Process. Manage. 56 (6) (2019) 1–20.