Building user profiles based on sequences for content and collaborative filtering

Pablo Sánchez, Alejandro Bellogín

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

Abstract

Modeling user profiles is a necessary step for most information filtering systems – such as recommender systems – to provide personalized recommendations. However, most of them work with users or items as vectors, by applying different types of mathematical operations between them and neglecting sequential or content-based information. Hence, in this paper we study how to propose an adaptive mechanism to obtain user sequences using different sources of information, allowing the generation of hybrid recommendations as a seamless, transparent technique from the system viewpoint. As a proof of concept, we develop the Longest Common Subsequence (LCS) algorithm as a similarity metric to compare the user sequences, where, in the process of adapting this algorithm to recommendation, we include different parameters to control the efficiency by reducing the information used in the algorithm (*preference filter*), to decide when a neighbor is considered useful enough to be included in the process (*confidence filter*), to identify whether two interactions are equivalent (*δ-matching threshold*), and to normalize the length of the LCS in a bounded interval (*normalization functions*). These parameters can be extended to work with any type of sequential algorithm.

We evaluate our approach with several state-of-the-art recommendation algorithms using different evaluation metrics measuring the accuracy, diversity, and novelty of the recommendations, and analyze the impact of the proposed parameters. We have found that our approach offers a competitive performance, outperforming content, collaborative, and hybrid baselines, and producing positive results when either content- or rating-based information is exploited.

Keywords: Hybrid recommender systems, Preference filtering, Content-based filtering, Collaborative filtering, Longest Common Subsequence

1. Introduction

Recommender Systems, even though they have been studied in depth during the last decade, remain a constant source of innovation. With the global growth of the Internet, they have become a necessary tool for a large number of online applications due to their ability to make personalized recommendations by adapting to different types of user profiles aiming to achieve better customer satisfaction [1]. In order to produce interesting and personalized suggestions, it is usually necessary to work with large amounts of data [1, 2], and depending on how this information is exploited and the type of strategy developed, several types of recommender systems can be distinguished, although the two most classical and widespread techniques are the content-based (CB) and collaborative filtering (CF) ones. The former learns to recommend items similar to the ones the user liked before [3], whereas the latter suggests items that users with similar tastes liked in the past [1].

However, both techniques present issues in some situations. Content-based algorithms have difficulties making serendipitous recommendations, as they suggest items that are very similar to the ones the user liked in the past, especially when the recommenders are more focused on producing accurate recommendations leaving aside other aspects like novelty or diversity [4]. Collaborative filtering techniques are less likely to have this problem, but when the user-item matrix has many rows or columns – users or items – without any rating, they will not be able to generate recommendations for those cases, which in the end may deteriorate the user experience with the system (an issue known in the community as *cold-start* [5]). A typical alternative to avoid these individual drawbacks is to use hybrid recommender systems, where different strategies are combined to improve the performance of separate systems [6, 7].

In this context, it is possible and desirable to innovate and improve both CF and CB techniques. Normally, most of these approximations work by transforming users or items into vectors and then applying some kind of similarity metric like Cosine similarity or Pearson Correlation [8, 9]. In this paper, we assume that sequences might be useful to build the user profiles that will be considered when computing the similarities. Thus, we propose to generate *sequences* of interactions from the user profiles under different assumptions and using different sources of information, to later exploit these sequences using pattern matching techniques. In particular, we compute the Longest Common Subsequence (LCS) between user sequences, by extending the standard algorithm to the recommendation context and use it as a new similarity metric, although the proposed transformation may also work for other string similarity algorithms.

Our work. In this paper, we present a framework to generate user profiles based on sequences by taking ratings or item features as input, producing similarities between users transparently and efficiently. Our approach thus operates with any user sequence generated according to the proposed framework, focusing on similarities based on an extension of the LCS technique for recommendation, where several configurations or settings become available, depending on the assumptions or necessities inherent in the system. In this work we explore three data transformation functions to take different sources of information into account (ratings, genres, and directors) and generate user sequences based on this information; once these sequences have been built, we allow the use of a threshold to determine when a matching is found between two users; we also experiment with two parameters that filter out low interactions (preference filter) and not valid neighbors (confidence filter); on top of this, we propose four possible normalizations to bound the result obtained from the LCS technique into the $[0, 1]$ interval.

Research questions. To better understand the behavior of our proposed approach, we formulate and address the following research questions: **(RQ1)** Which information source (between pure collaborative filtering or content-based with genres and directors) performs the best in terms of ranking quality (e.g., nDCG) when creating user sequences? **(RQ2)** Which combination of parameters of confidence, preference, threshold, and normalizations achieve better results? **(RQ3)** What is the performance of our approach on beyond-accuracy metrics (i.e., diversity and novelty)? **(RQ4)** How does this approach compare against other state-of-the-art algorithms?

With these goals in mind, we experiment with and analyze the proposed technique using a popular dataset in the recommender systems community that contains content-based information together with user-item interactions. In summary, the main contributions of this work are:

- A generic framework to generate interaction sequences of user profiles, by means of different transformation functions (with additional configuration parameters) that can deal with heterogeneous information sources.
- The use of the Longest Common Subsequence (LCS) technique as a similarity metric to compare two users in a recommender system, once they have been transformed into interaction sequences.
- A thorough comparison between different versions of the algorithm and standard recommendation techniques using both ranking quality and novelty and diversity metrics.

The reminder of the paper is organized as follows: in Section 2 we provide a detailed explanation of actual approaches about content-based and collaborative filtering recommender systems. Section 3 presents our proposal to generate user profiles based on sequences, where we integrate LCS as a similarity metric to be used as a hybrid content-based and collaborative filtering recommender system as well as the different parameters and configurations allowed in our model. In Section 4 we show the results of the proposed approach and its comparison with other known algorithms in terms of novelty, diversity, and ranking quality evaluation. Then, Section 5 presents some works where similar techniques have been explored, and we end in Section 6 with some conclusions and possible extensions for the future.

2. Background

As introduced before, the purpose of a recommender system is to make recommendations to users by analyzing their tastes, preferences, and interests. Content-based (CB) and collaborative filtering (CF) are the two most popular recommendation approaches. Collaborative filtering techniques suggest items to users based on the preferences of similar users [9], while content-based approaches recommend items to users that are similar to the ones they liked in the past [3]. To clarify our notation in the paper, we will use symbols $\mathcal{I}$, $\mathcal{U}$, and $\mathcal{R}$ to denote the items, users, and ratings (usually in the $[1, 5]$ range) available in the system.

In CB, recommendations are made normally in three steps using independent components [3]: first, the content analyzer, which makes the data pre-processing by extracting the relevant information about the items. Then, the profile learner that builds a user profile using the user's preference, and, third, the filtering component that exploits the user's profile and suggest items by matching them against the profile.

On the other hand, CF is usually divided in two categories: memory-based and model-based techniques. Memory-based algorithms make predictions using the stored interactions of the users directly. They normally compute similarities between users and items to generate the recommendations. Model-based techniques, in contrast, learn a predictive model, for instance, by transforming users and items into a latent factor space [9, 10].

In this paper, we focus on memory-based algorithms – also known as "k-nearest neighbors (kNN)" – which are further divided into user-based and item-based kNNs. The standard definition of a user-based kNN algorithm is:

$$s(u,i) \propto \sum_{v \in \mathcal{N}_i(u)} r_{vi} w_{uv} \tag{1}$$

In this case, $s(u,i)$ denotes the predicted score for user u and item i, w_{uv} is the weight (or similarity) between users u and v, whereas $\mathcal{N}_i(u)$ denotes the neighbors (closest users with respect to a similarity metric) of user u that have rated item i. Equation 1 is usually normalized when rating accuracy is optimized (i.e., we aim to reduce the system error); however, when the system is evaluated with ranking metrics such as precision or nDCG, ignoring the similarity weights in the normalization tends to produce better results [11].

Three popular similarity metrics to compute $w_{uv} = \text{sim}(u,v)$ between users are Cosine similarity, Pearson correlation, and Jaccard coefficient:

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

$$\text{PC}(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{3}$$

$$\text{Jaccard}(u,v) = \frac{|\mathcal{I}_{uv}|}{|\mathcal{I}_u \cup \mathcal{I}_v|} = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u \cup \mathcal{I}_v|} \tag{4}$$

where $\mathcal{I}_u$ denotes the items rated by u, $\mathcal{I}_{uv}$ the items rated by both u and v, and $\bar{r}_u$ is the average rating of user u. These traditional similarity metrics have some drawbacks that make them not suitable under some situations and have led to the emergence of other similarity metrics [12]. Normally, most of them are only useful when the number of items rated by both users is large [13]. Cosine similarity does not consider the users' mean nor their variance and the Jaccard coefficient does not work with the rating values. Besides, all of them are undefined when working with repetitions[1] and it is not easy to add a temporal dimension to them either. In the following section, we propose a novel similarity metric that aims to address some of these issues while, at the same time, achieves competitive performance.

[1]Traditionally, it has been considered that a user can only consume each item once, however, in the real world, there are many users who tend to consume several times the items they liked.

3. On generating sequences of user profiles

We propose to use the Longest Common Subsequence (LCS) as a similarity metric that can be integrated in a generic recommender system (either using a content-based or a collaborative filtering technique). In the following, we will consider the case of user similarity, that is, $\text{sim}(u, v)$, although equivalent formulations could be derived for the case of item similarity. For that, in this section, we first describe how user profiles can be generated based on sequences in a generic, formal way (Section 3.1). We then describe in Section 3.2 how we adapt LCS to recommendation based on how it works for strings, extending the approach presented in [14] that only worked in a collaborative filtering scenario – where users are represented as ratings. Finally, Sections 3.3 and 3.4 present two additional procedures that can be applied to almost any user similarity metric to enhance its performance: preference and similarity filtering and similarity value normalization.

3.1. A framework to generate user profiles based on sequences

In our approach, we propose to generate sequences – defined as a collection of elements where repetitions may exist – from the information related to a user, so that sequence similarity functions can be defined and applied to two such sequences, in order to derive how close/different those two users are. With this goal in mind, we shall define a transformation function f as follows. Assuming the user is described as a set of items and ratings (or any other numerical information related to the items, such as click counts, access frequency, or binary interaction), the following steps are necessary to generate a sequence in a generic way:

1. Extend any information about the items rated by the user. This is typical of content-based algorithms [3], but collaborative filtering algorithms can also exploit side information from items or even extend the user profile using other techniques [15, 16]. Formally, we need a function that, for every item, returns a set of elements associated to such item; that is: $e : \mathcal{I} \times \mathcal{R} \to \mathcal{I} \times \mathcal{T}^k$, where $k > 0$ denotes the number of those elements that function e is able to associate with every item, and $\mathcal{T}$ represents those elements, modeled in general as tuples. Classical CF would use the identity function in this step: $e_{ir}(i, r) = (i, \{i, r\})$. On the other hand, content-based methods would exploit the feature space so that every item is linked to their corresponding features: $e_{Ar}(i, r) = (i, \{A_j(i), r\}_j)$, where the feature space A could be genres, directors, or actors in the movie domain, text features in news recommendation, and the rhythm pattern or the spectogram in music.
2. Represent the tuples created above as symbols interpretable by the sequence similarity function. Although this step is not really needed (we can *move* this knowledge to the similarity) it also helps to increase the efficiency of the whole process, as we shall see later. Hence, we propose to use $t : \mathcal{I} \times \mathcal{T}^k \to \mathcal{I} \times \mathbb{Z}^k$, where a proper transformation between $\mathcal{T}$ and $\mathbb{Z}$ (the set of integer numbers) is required. We propose to work with integer sequences because they are equivalent to character sequences (strings, where many sequence similarity functions have been defined [17]) while they allow comparisons to be more computationally tractable.

As a simple example, associated to the function e_{ir} we would have a transformation function $t_{ir}(i, r) = 10 \cdot \mathrm{id}(i) + r$ in such a way that we can also recover the original elements of the tuple (the item id and its corresponding rating) given its output. The factor of 10 that multiplies the id helps us to separate the item id and the rating while combining them into a single "character". In fact, another possible function is defined by only using the ratings, as we can directly set $t_i(i) = \mathrm{id}(i)$. Nonetheless, if we decide to use them, note that if the ratings are in the $[0, 10]$ interval, the transformation function should be modified accordingly, i.e., $t_{ir}(i, r) = 100 \cdot \mathrm{id}(i) + r$, in order to make that recovery possible – i.e., to have a bijective function.

3. Arrange the symbols into a sequence. In string matching, the ordering of the sequence is important, and it is an aspect that some sequence similarity functions are able to exploit. In this paper, we will simplify this step and sort the items in the sequence according to their item id, although it is worth noting that any other global ordering of the items would be equivalent to this one, for example, item popularity – we leave as future work sorting the items in a user basis (like ordering the items rated by timestamp, from the oldest to the most recent ones). In this way, the sequence arranging function we propose will take several pairs of items and tuples generated as described before and will output a sequence of symbols, prepared to be processed by any sequence similarity algorithm. Formally, such a function will be defined as $s(\{i_j, (n_{jk})_k\}_j) = ((n_{jk})_k)_{j=1}^{|\mathcal{I}|}$.

Finally, the sequence generation function f could be seen as a composition of the three functions presented above: $f = s \circ t \circ e$.

To clarify the process of sequence generation explained before, let us present an example considering a dataset based on movies, which usually have some content-based information associated like actors, directors, or genres. We have the film *Star Wars IV*, with id 1, and a user u who has rated it with a 5 as rating value. If we use function e_{gr} to extend this information based on genres – i.e., $A = G$ and then $e_{gr}(i, r) = (i, \{G_j(i), r\}_j)$ – we could find that item 1 has two genres: Sci-Fi (id 7) and Adventure (id 10). According to the definition of e_{gr}, this function leads to the tuple $(1, \{\{\text{Sci-Fi}, 5\}, \{\text{Adventure}, 5\}\})$. After that, we would represent these tuples as useful symbols for the sequence similarity function (LCS, in our case) using a reasonable t_{gr} function. By taking a similar one to t_{ir}, we could transform each genre into its id and use that value in combination with its associated rating, creating the tuple $(1, \{75, 105\})$. Finally, to generate the sequence corresponding to this user, we simply take the tuples associated to the only item this user has rated: $(75, 105)$. However, if the user had also rated *The Godfather* (with a rating value of 4 and whose id is 15), then the output would be slightly different. This movie has Drama (id 2) and Crime (id 6) as genres. The tuple related to this second movie would be (following the same steps as before, i.e., using $t_{gr} \circ e_{gr}$): $(15, \{24, 64\})$. The final step would produce the sequence $(75, 105, 24, 64)$, since the id of *Star Wars* is lower than the one for *The Godfather*. Note that if e_{ir} and t_{ir} functions are used, that is, pure collaborative filtering information is being exploited, then each item will only generate one tuple and the final generated sequence will be shorter: $(15, 154)$.

Algorithm 1 Longest Common Subsequence

```
procedure LCS($x, y$)                         ▷ The LCS of $x$ and $y$
    $L[0 \cdots m, 0 \cdots n] \leftarrow 0$
    for $i \leftarrow 1, m$ do
        for $j \leftarrow 1, n$ do
            if $x_i = y_j$ then
                $L[i, j] \leftarrow L[i-1, j-1] + 1$          ▷ It is a match
            else
                $L[i, j] \leftarrow \max(L[i, j-1], L[i-1, j])$
            end if
        end for
    end for
    return $L[m, n]$        ▷ $L[m, n]$ is the length of the LCS between $x$ and $y$
end procedure
```

3.2. A sequence similarity metric: the Longest Common Subsequence

As defined before, a sequence is a collection of elements where repetitions may exist. Consequently, the Longest Common Subsequence (LCS) problem consists in finding the longest subsequence among n given sequences over an alphabet $\Sigma = (\sigma_1, \cdots, \sigma_s)$. A subsequence β of a sequence α is another sequence composed of $[1, |\beta|]$ elements of α without changing the order. The LCS problem is usually computed between two sequences ($n = 2$) and it can be solved by applying dynamic programming as shown in Algorithm 1. The procedure is to fill an $(m+1) \times (n+1)$ matrix initialized to 0, where m and n are the lengths of each of the sequences x and y involved in the computation, following this formula:

$$L[i, j] = \begin{cases} 0 & \text{if i = 0 or j = 0} \\ L[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(L[i, j-1], L[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases} \qquad (5)$$

The final position $L[m, n]$ contains the length of the LCS between the two sequences. It is important to state that sometimes the longest common subsequence is not a unique string, although the length of such sequence is unique. Note that we need an extra row and column in the matrix in order to start the loop iterations with a default length value of 0, equivalent to the base case of the recursion described by Equation 5. As an example, we show in Table 1 the matrix L between sequences AGGT and GCGT, whose LCS is 3 (for the subsequence GGT), as can be seen in the circled value.

Table 1: Example of computation of LCS between sequences AGGT and GCGT.

	∅	A	G	G	T
∅	0	0	0	0	0
G	0	0	1	1	1
C	0	0	1	1	1
G	0	0	1	2	2
T	0	0	1	2	$\textcircled{3}$

Algorithm 2 Longest Common Subsequence for Recommender Systems

```
1:  procedure LCS_RECSYS($u, v, f, \delta$)          ▷ The LCS of users $u$ and $v$ applying transformation $f$
2:      $(x, y) \leftarrow (f(u), f(v))$              ▷ String $x$ contains $m$ symbols
3:      $L[0 \cdots m, 0 \cdots n] \leftarrow 0$
4:      for $i \leftarrow 1, m$ do
5:          for $j \leftarrow 1, n$ do
6:              if match($x_i, y_j, \delta$) then     ▷ There is a $\delta$-matching
7:                  $L[i, j] \leftarrow L[i-1, j-1] + 1$
8:              else
9:                  $L[i, j] \leftarrow \max(L[i, j-1], L[i-1, j])$
10:             end if
11:         end for
12:     end for
13:     return $L[m, n]$
14: end procedure
```

In this case, both time and space complexity are $O(mn)$. However, space complexity can be reduced to $O(m)$ or $O(n)$ if only the length of LCS is needed as we do not need the previous rows. This algorithm is often used to compare DNA strings and in version control systems [18], but it has not been extended – to the best of our knowledge – to the Recommender Systems area, besides our previous work presented in [14].

In this paper, we propose a user similarity metric $\text{sim}(u, v)$ that is able to deal with users represented as content features (content-based scenario) or as ratings (collaborative filtering scenario) in a simple, coherent way under a formulation based on the Longest Common Subsequence (LCS) algorithm. As the Longest Common Subsequence problem is defined between two sequences, in order to apply such algorithm to recommendation, a transformation between the information available in the recommender system and sequences is needed first – i.e., a function f as defined by the framework presented before.

For the sake of simplification, we aim to use the length of the longest common subsequence as a proxy for the similarity between two users in a recommender system, we leave as future work exploiting the LCS algorithm to explore patterns from the data through combining different elements of the recommender systems. Additionally, it should be noted that, once users or items are represented as sequences, other string distance or alignment algorithms such as Jaro (and Jaro-Winkler) or Smith-Waterman [19] could be applied to such sequences, by following a similar development to the one presented here; however, we believe this task is out of the scope of this paper and leave this analysis for the future.

As we present in Algorithm 2, it is possible to adapt the LCS algorithm from string matching to recommendation with only two modifications: a transformation function f (from users to sequences) and a δ-matching allowing us to configure when two symbols of the alphabet are considered equal. We introduce this latter modification – following the methodology proposed in [14] – to address the inherent fuzziness of user preferences. Classical similarity metrics like Pearson correlation (see Equation 3) perform comparisons of user ratings with respect to the user's mean, other metrics and algorithms do this in relative terms; this *matching threshold* δ softens the matching condition of the LCS algorithm by deciding that two symbols are equivalent whenever their difference is below such threshold δ. Hence, higher values of δ model that we allow larger differences between ratings so they are still considered as "equal" (i.e., they represent the same symbol).

The transformation function f is the most critical component of our approach, since, depending on its definition, it could generate different sequences – with different resulting evaluation performance – using the same information about users. These sequences (generated either using content-based or collaborative information) will be then transparently used by the LCS algorithm to find similarities between users which, in turn, will be integrated in a technique based on nearest neighbors, so that recommendations can be generated in the classical way. As we shall see, the way these sequences are generated has a critical impact in the final performance of the recommendation algorithm.

3.3. Preference and similarity filtering

The LCS algorithm has a complexity of $O(mn)$, with m and n being the length of the sequences to compare. When receiving very large sequences, computing LCS between all users may become too expensive in terms of computational cost. We can reduce the length of both sequences by filtering out the less important preferences. We introduce a parameter, denoted as γ (we name it as *preference filter*) to indicate which items will be considered when computing the LCS algorithm. The idea is that items with low ratings may not be interesting when finding neighbors of a particular user, an issue derived by the missing-not-at-random problem, that states that users tend to rate what they prefer [20, 21].

This filter can be introduced as a prefiltering step, where low preferences from users are filtered out, and these processed users are the input for the transformation function f; this would introduce a fourth component in the definition of the transformation function: $f^\gamma = s \circ t \circ e \circ \gamma$. Another possibility for modeling this step is to modify one of the functions involved in the definition of f so that the input to the function remains the same. In that case, it would be enough to have an extending function e^γ that only outputs values whenever the associated rating is above the γ threshold; hence, the corresponding $f^\gamma = s \circ t \circ e^\gamma$ would work as explained in the previous section. In both cases, Algorithm 2 would work unaware of this filter.

On the other hand, while obtaining similar users to a target one, we can set a minimum value of similarity to consider another user as a (valid or useful) neighbor. This parameter can be seen as the number of items that both users have rated in a similar way (or depending on the threshold δ, with a value $\leq \delta$). We have added this parameter naming it as *confidence filter*, represented by τ. This parameter imposes a harder constraint on the potential neighbor, and hence it reduces the number of possible neighbors that a user may have.

It is important to note that, although these two filtering approaches aim at increasing the accuracy of the discovered neighbors (because only important preferences are being considered or only the highest similarities are taken into account), the final coverage of the algorithm can be damaged if these parameters are very restrictive, since less neighbors will satisfy these constraints, which may produce less recommendations for each user.

3.4. Similarity normalization

The LCS algorithm obtains a value in the interval $[0, \min(|f(u), f(v)|)]$. However, in neighbor-based recommendation similarity metrics are usually normalized to have a range in $[-1, 1]$ or $[0, 1]$, and different normalization techniques are then applied in order to estimate the predicted score $s(u, i)$ [9]. Following the same rationale, we propose four different normalizations for our LCS-based similarity metric:

$$\begin{aligned}
\text{sim}_1^{f,\delta}(u, v) &= \text{LCS\_Recsys}(u, v, f, \delta) && (6)\\
\text{sim}_2^{f,\delta}(u, v) &= \frac{\text{sim}_1^{f,\delta}(u, v)^2}{|f(u)| \cdot |f(v)|} && (7)\\
\text{sim}_3^{f,\delta}(u, v) &= \frac{2 \cdot \text{sim}_1^{f,\delta}(u, v)}{|f(u)| + |f(v)|} && (8)\\
\text{sim}_4^{f,\delta}(u, v) &= \frac{\text{sim}_1^{f,\delta}(u, v)}{\max(|f(u)|, |f(v)|)} && (9)\\
\text{sim}_5^{f,\delta}(u, v) &= \frac{\text{sim}_1^{f,\delta}(u, v)}{\min(|f(u)|, |f(v)|)} && (10)
\end{aligned}$$

Except for Equation 6, that produces the LCS-based similarity with no normalization, that is, as calculated by the Algorithm 2, the other equations present different normalizations of Equation 6, producing values in the $[0, 1]$ interval. It is important to note that although the computational cost of the LCS algorithm may be high, once sim_1 is computed, the other normalizations can be obtained straightforward, with almost no extra cost. In general, these normalizations favor longer subsequences found inside short sequences, as they include the sequence lengths in the denominator as a penalization. These functions were proposed in [22] to compare the output of the LCS algorithm, in a similar way as we aim to do here.

Finally, although the LCS algorithm is an NP-hard problem, 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. Once the similarities are stored, they can be used in a nearest-neighbor recommender transparently, without knowing if the similarity weight comes from an LCS-based measure or a classical one.
- 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 (we can create five recommenders by the cost of one).

Table 2: Items consumed by users u_1 and u_2.

Movie (id)	**Director (id)**	**Genres (ids)**	u_1	u_2
The Wild Bunch (M1)	Sam Peckinpah (D1)	Western (G1), Robbery (G2)	5	
Seven Samurais (M2)	Akira Kurosawa (D2)	Action (G3), Drama (G4), Adventure (G5)	4	5
The Iron Cross (M3)	Sam Peckinpah (D1)	War (G6)	3	
Gladiator (M4)	Riddley Scott (D3)	Action (G3), Drama (G4), Adventure (G5)	4	2
Alien (M5)	Riddley Scott (D3)	Sci-Fi (G7), Terror (G8)		5
The Magnificent Seven (M8)	John Sturges (D4)	Western (G1), Adventure (G5)		4

- 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).
- The proposed method to generate the sequences can be applied to any other algorithm like the Levenshtein distance, the Jaro-Winkler similarity or even approximation algorithms of LCS.

3.5. Toy example

To better understand how the proposed similarity function works under different transformation functions, we show in this section a toy example where two users have rated four movies each. Table 2 shows the items consumed by the first user, u_1, and the second user, u_2. In this table, content features such as genres and directors are also present, together with their corresponding ids, to understand how the transformation function generates the sequences in each situation. Table 3 shows the sequences obtained for each definition of transformation function f, and the result computed by the LCS-based similarity function. We denote this transformation function as f_i when the transformation only uses the ids of the items, without the ratings, whereas we use f_{ir}, f_{dr}, and f_{gr} when we use ratings with items, directors, and genres, respectively.

The first thing we notice in Table 3 is that the matching threshold does not affect the user representation, which allows us to separate the process in two steps: we generate as many user sequences as transformation functions we want to test, and then we compute the LCS-based similarity according to different parameters. It is important to note, however, that different preference filters γ generate different sequences. The confidence filter, on the other hand, only affects the final output of the comparison. In the examples presented here, the same similarity value is obtained when the preference filter is used ($\gamma > 0$) and when it is ignored, although this will not be true in general; it is interesting to observe, nonetheless, that the application of the preference filter produces shorter sequences, hence allowing for more efficient computation of the LCS algorithm.

We also observe that different representation spaces (items, directors, genres) create shorter or longer user sequences, which, in the end, affect the final similarity value.

Table 3: User representation as sequences and LCS-based similarity for different transformation functions, matching thresholds (δ), and preference (γ) and confidence (τ) filters.

f	δ	γ	τ	$f(u_1)$	$f(u_2)$	$\text{sim}_1^{f,\delta}(u_1,u_2)$	$\text{sim}_2^{f,\delta}(u_1,u_2)$
f_i	0	0	0	$(1,2,3,4)$	$(2,4,5,8)$	2	1/4
f_{ir}	0	0	0	$(15,24,33,44)$	$(25,42,55,84)$	0	0
	1	0	0			1	1/16
	1	4	0	$(15,24,44)$	$(25,55,84)$	1	1/9
f_{dr}	0	0	0	$(15,24,13,34)$	$(25,32,35,44)$	0	0
	1	0	0			2	1/4
	1	0	3			0	0
	0	5	0	(15)	$(25,35)$	0	0
f_{gr}	0	0	0	$(15,25,34,44,54,63,34,44,54)$	$(35,45,55,32,42,52,75,85,14,54)$	1	1/90
	1	0	0			4	2/45
	1	4	0	$(15,25,34,44,54,34,44,54)$	$(35,45,55,75,85,14,54)$	4	1/14
	1	4	2			4	1/14

Because of this, to allow fair comparisons of similarities without having to tune or analyze each of them separately, the use of normalization functions is key.

4. Empirical evaluation

In this section, we show several experiments to analyze the proposed similarity metric using the LCS algorithm, based on both rating and content information. In Section 4.1 we present the dataset used in this work together with the evaluation metrics that will be reported later and other experimental settings. Then, Section 4.2 describes the baselines that will be used to compare against the proposed method; and finally, in Section 4.3 we present the results obtained for different experiments.

4.1. Experimental setup

In this paper, since we want to test the similarity metric proposed in Section 3, we work with a dataset where ratings and content-based features are available at the same time. The HetRec version of the MovieLens dataset (described in [23] and available from the GroupLens website[2]) fits these requirements. This dataset is a subset of the Movielens10M release of the MovieLens datasets, where only users that have both ratings and tags are included. This dataset is composed by 2,113 users, 10,197 movies, 4,060 directors, 20 movie genres, and 855,598 ratings. The ratings are made on a 5-star scale with half-star ratings (from 0.5 stars to 5.0 stars). Because of this, every transformation function that concatenates the rating of the users to items representation will be done by multiplying the item id by 100 and the rating value by 10, in order to support the half stars. For example, if user u has rated item 1 with a 3.5 rating, the representation using f_{ir} transformation will be 135.

[2]`http://grouplens.org/datasets/hetrec-2011/`

All the recommenders have been implemented on top of the RankSys framework[3]. We focus our evaluation on ranking-based metrics, hence no error metrics (such as MAE or RMSE) will be reported. The rankings are generated following the TrainItems methodology described in [24], where every item in the training split, except the ones already seen by the user in training, is considered as a possible candidate to be part of a user's final ranking. Additionally, we include novelty and diversity metrics available in the RankSys framework using movie genres as item features.

More specifically, we report the nDCG metric (normalized Discounted Cumulative Gain) as a proxy of the ranking quality produced by different configurations of the recommenders evaluated. In some situations, we also report precision (amount of returned relevant items), recall (ratio of relevant items that are returned by the algorithm), and MAP (Mean Average Precision) – all of them as implemented in the RiVal[4] library. For novelty and diversity, the metrics that we have used are the following (described in [25, 26]):

- α-nDCG: a diversity-aware ranking metric where the score of retrieved documents is penalized if they share features with documents ranked higher. The reported values use $\alpha = 0.5$.
- Aggregate diversity (AD): it returns a value proportional to the total number of items that the system recommends.
- EILD (expected intra-list diversity): rank-sensitive and rank-aware expected intra-list diversity metric.
- EPC (expected popularity complement): expected number of seen relevant recommended items not previously seen.
- EPD (expected profile distance): expected distance between the recommended items and the items in the user profile using a provided feature scheme for each item.
- Gini: it takes into account not only whether items are recommended to someone, but to how many people and how even or unevenly distributed. Following standard procedures in the literature, we report the complement of the original Gini Index so that higher results indicate more balanced and diverse recommendations.

Besides, these results have been obtained taking only into account the items in test that have been rated with 5 stars, as we consider that recommending items with lower ratings are not good recommendations. Unless stated otherwise, all these metrics (ranking quality metrics and diversity and novelty metrics) have been computed at cutoff 5, that is, only considering the top 5 items in every ranking returned by a recommender. Finally, all results have been generated using a 5-fold cross-validation evaluation, where we use 80% of the data to train the recommenders, and the remaining 20% is used for evaluation.

[3] `https://github.com/RankSys/RankSys`
[4] `http://rival.recommenders.net/`

4.2. Baselines

In this article, since we are proposing a new user similarity based on the LCS algorithm that can use content-based information, one of our main goals is to compare it against other state-of-the-art recommenders. We will not only compare it against other similarities like Cosine or Jaccard but also against other recommenders like matrix factorization and item popularity. Thus, the baselines used in the experiments are:

- Random: a random recommender that returns scores randomly for each user-item pair.
- Pop: popularity recommender. The items with more ratings will be recommended to users.
- IB: a pure collaborative filtering using a rating-based similarity between items. We experimented with Cosine and Jaccard similarity measures.
- UB: a pure collaborative filtering using a rating-based similarity between users. Besides the classical Cosine and Jaccard similarity measures, we also include in the comparison a combination of the Jaccard index and mean squared differences (MSD) proposed in [27] (JMSD).
- LDA: Latent Dirichlet Allocation for collaborative filtering recommenders. This one refers to LDARecommender from RankSys and originally proposed in [28].
- MF: a matrix factorization recommender. This one refers to MFRecommender with the PLSAFactorizer from RankSys, proposed in [29].
- PureCB: a pure content-based recommender using a Vector Space Model (VSM) with binary weights. This means that the coordinates of the vectors contains a 1 if the item has that feature and zero otherwise. Then, for each item, we compute the Cosine similarity between the user and the item. Directors and genres are tested as features.
- CBCF: a hybrid recommender system in which we take the classical user-based collaborative filtering formulation but instead of using rating-based similarities, content-based similarities are generated – using genres and directors like for PureCB – between users by transforming them into a VSM and then computing the Cosine similarity. This corresponds to the *collaborative-via-content* method proposed in [30], also described in [31].

Unless stated otherwise, the parameters of these baselines are: 100 neighbors and a TF-IDF representation for the CBCF recommender, the feature vector contains directors for CBCF and PureCB, 5 neighbors for IB and 100 neighbors for UB with JMSD and Jaccard similarities and 90 neighbors with Cosine; MF and LDA are computed with 50 factors and using default values for other parameters[5]. These parameters were

[5]Specifically, for MF we used 20 iterations and for LDA we set alpha=1, beta=0.01, burninPeriod=50, and 20 iterations.

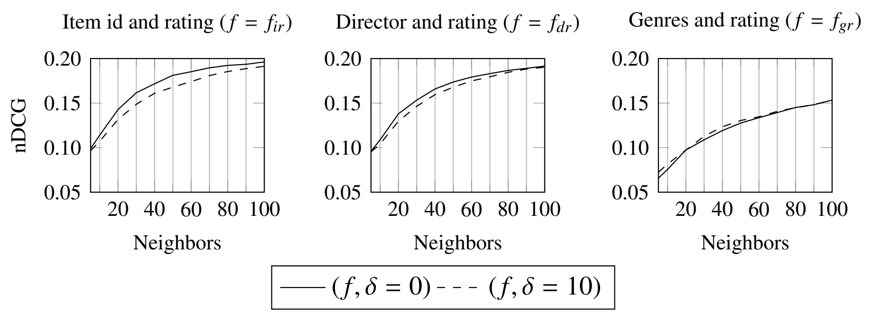

Figure 1: Results with transformations based on items (pure collaborative filtering), directors, and genres.

selected according to a preliminary test where the best configurations with respect to the nDCG@5 metric were chosen, where the optimal parameters were obtained after a grid search with the values $5, 10, 20, \ldots, 100$ for the neighborhood size (UB, IB, and CBCF).

4.3. Results

In this section, we present several results to answer the research questions presented at the beginning of the paper: (RQ1) Which LCS approach (between pure collaborative filtering or content-based with genres and directors) performs the best in terms of ranking quality (e.g., nDCG)? (Section 4.3.1) (RQ2) Which combination of parameters of confidence, preference, threshold, and normalizations achieve better results? (Section 4.3.2) (RQ3) What is the performance of this approach on beyond-accuracy metrics (i.e., diversity and novelty)? (Section 4.3.3) (RQ4) How does this approach compare against other state-of-the-art algorithms? (Section 4.3.4)

4.3.1. Performance of LCS as user similarity

In this section we explore how, depending on the function used to extend the rating information to build the user sequences, the proposed similarity metric based on LCS may perform differently, and analyze which transformation is better suited for recommendation. More specifically, and using the notation from Section 3.1, we exploit three sequence generation functions $f = s \circ t \circ e$, composed of the same sequence ordering function s and transformation function t. We shall denote as f_{ir} the pure collaborative filtering approach, that is, where function e is the identity and users are composed of a combination of the rated item and the rating value. Two content-based approaches are explored by considering f_{dr} and f_{gr} as transformation functions, the latter uses function e_{gr} as introduced in Section 3.1 (where each item is represented as its corresponding genres), and the former uses a similar function that represents the items according to their directors.

Figure 1 shows the results obtained when these three different transformation functions are used together with two δ-matching thresholds: exact matching ($\delta = 0$) and matchings allowing a difference of ± 1 in the rating value (using $\delta = 10$ as explained before because of the presence of half-star ratings in this dataset). It is interesting to observe that results are the same or worse when exact matchings are not used. We shall show later that this trend depends on the specific configurations used regarding the other parameters available in the model. Our hypothesis in this case is that worse neighbors are found when non-exact matchings are allowed, probably due to an ill-defined neighborhood caused by considering users that share *similar preferences* closer than users with *the same preferences*.

Furthermore, we also observe from the same figure that content-based information, when integrated into our LCS-based similarity metric, performs equally or worse than applying a pure collaborative filtering: compare f_{dr} and f_{gr} against f_{ir}. In this case, the worst recommender is always the one that uses genre information. This may be an expected result since genres in films are sometimes very subjective – it is actually difficult to define the genres of movies, since even if two movies have the same genres, they could be totally different. This is not the case, on the other hand, of using directors as a feature, as each director may have their own style that is evident in the films they make. We can see that this approach remains as competitive as the pure collaborative filtering one, and from now on, we will omit the results using the genre feature.

At this point, we can answer the first research question. Since in the configurations tested so far the transformation using genres performs the worst, we can say that **in this dataset, using content-based information on top of the LCS-based similarity does not outperform a pure collaborative filtering recommendation using the same similarity, although one particular transformation (directors) remains a competitive approach.**

4.3.2. Sensitivity to confidence, preference, and normalization parameters

In this section we analyze how the different parameters of the proposed user similarity affect its performance. That is, we shall compare whether the recommendation performance changes when the three variations of the proposed metric introduced before are included in the approach one at a time. Let us start with the *confidence filter* parameter τ. Recall that this parameter acts as a neighbor filter, where each candidate neighbor needs to have rated "in the same way" as the target user a configurable number of items in order to be considered as a neighbor. In our experiments, we have used many different values, but we only report three: 30, 50, and 70; smaller and larger values had almost no effect, due to sparsity reasons. Of course, the range of this parameter will always depend on the data and, hence, it will have to be tuned in consequence.

Figure 2 shows the performance of the recommenders for two transformation functions (directors and item) using the three different values of confidence mentioned before. We include in dashed lines the same recommenders with a δ-matching threshold of 10. We observe that using some kind of confidence filter entails better results. In contrast with the previous experiment (Figure 1) where no configuration reached a 0.2 value of the evaluation metric being used (nDCG@5), now by using any confidence value we raise the performance of all the configurations with respect to when no confidence ($\tau = 0$) is used. Therefore, consistent with previous results in the literature (see [8]), we find that **a posterior neighbor filtering can improve the results obtained by our approaches**. Nonetheless, better results are still obtained for the pure collaborative filtering representation of the user sequences ($f = f_{ir}$) and, at the same time, exact matching produces higher improvements than non-exact matching ($\delta = 10$).

The second parameter we analyze is the *preference filter* parameter γ. This param-

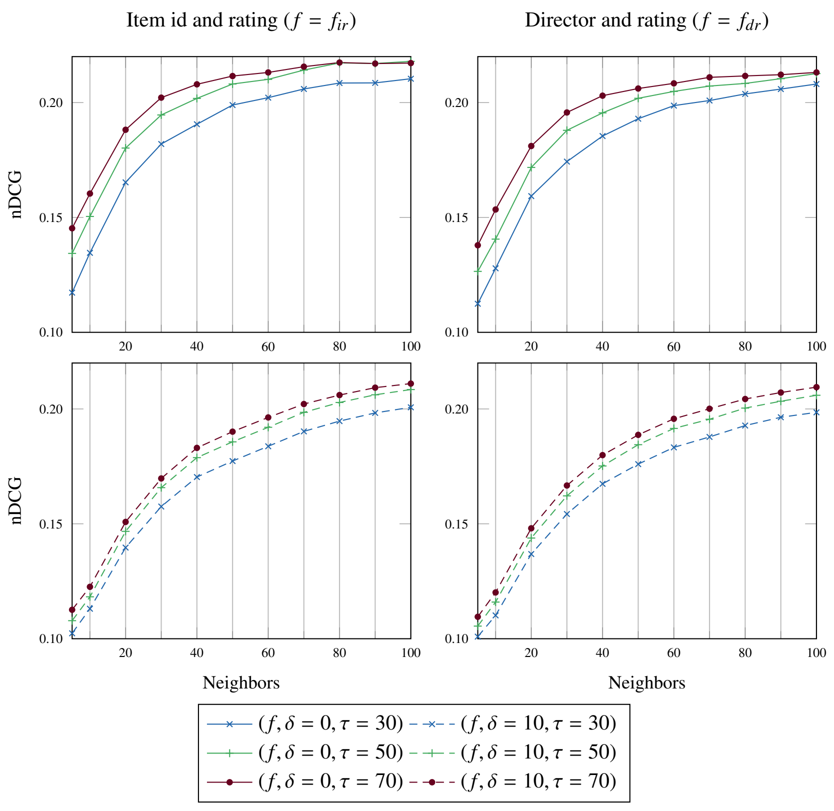

Figure 2: Results with the confidence filter parameter τ.

eter allows us to consider in the user sequences only those items having a rating value higher than a particular value. In our experiments we consider the following possibilities: larger than 3 ($\gamma = 3$), larger than 4 ($\gamma = 4$), and larger than the user mean ($\gamma = \overline{u}$). Figure 3 shows the results for this experiment.

In this case, although there is some improvement with respect to the basic model without preference filter ($\gamma = 0$ depicted in Figure 1), the change in performance is lower than the one achieved with the confidence parameter. However, it should be noted that the lack of a performance degradation is a very positive result, since in this situation the model is computing the user similarities with less data, and hence, its efficiency improves with, according to these results, no expenses in performance. Furthermore, although in some situations the results for different γ values are very close to each other, we can conclude that the best one is $\gamma = \overline{u}$, that is, the one that uses the user mean rating. This is reasonable, since only considering the values higher than a predefined threshold – such as 3 or 4 – could make us lose important information due to the inherent bias of each user when rating items.

In light of the previous results, it makes sense to explore and combine these two parameters (confidence and preference filtering) together, and see how they may affect

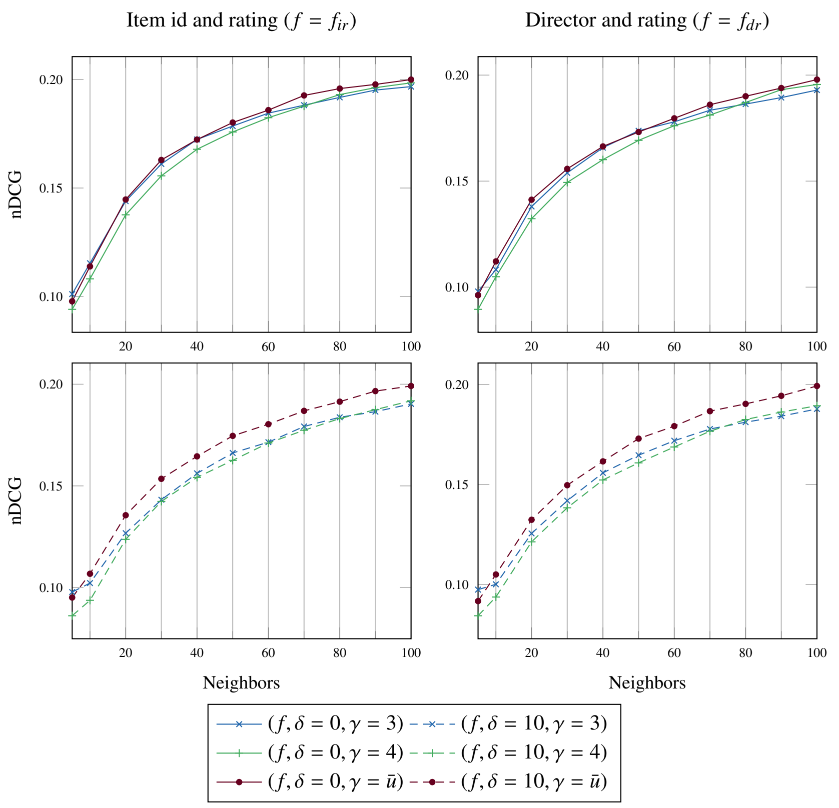

Figure 3: Results with the preference filter parameter γ.

the algorithm performance. In such a case, this means that we are filtering neighbors (according to some confidence value) using only the items that have been rated with a higher rating than the preference value. We restrict this analysis to $\gamma = \overline{u}$ since it shows better properties to model the user in a generic way, as discussed before.

When using these two parameters combined, a general improvement is achieved, as shown in Figure 4. Interestingly, in this situation larger improvements are found when a non-exact matching is allowed ($\delta = 10$). Actually, only under this setting, an improvement with respect to using only the confidence filter is obtained, hence, producing the highest performance among the different instantiations of the recommenders analyzed so far. We observe in this figure that, when combining preference and confidence filtering and a threshold of $\delta = 0$, the performance of the recommender remains constant after a number of neighbors are considered. This is because these two parameters reduce the number of possible neighbors; without using a proper value of the δ-threshold that palliates this effect, the number of neighbors will be reduced dramatically, preventing an improvement in the recommender.

Finally, we analyze the results for the third parameter of the proposed user similarity metric, the *similarity normalization* functions. Following the notation used in

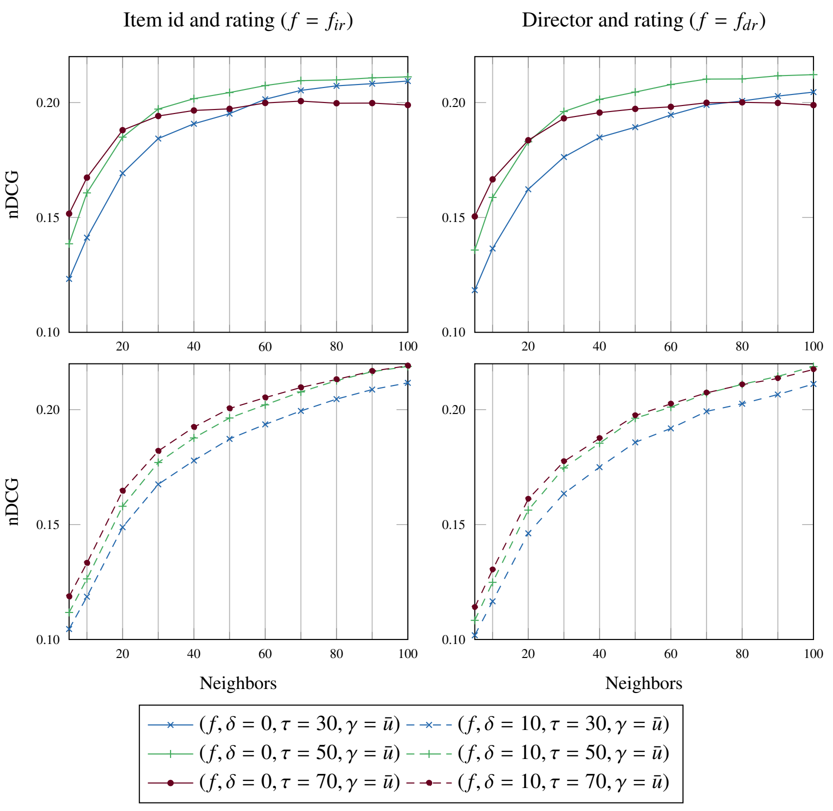

Figure 4: Results with a combination of preference and confidence filtering.

Section 3.4, we show in Figure 5 the results of the simple model when no confidence ($\tau = 0$) and preference ($\gamma = 0$) filtering is used and compare the not normalized values presented so far (that is, using sim_1) against the results when the other four normalization functions are used. Interestingly, we observe that this parameter has a huge (positive) effect in performance, since sim_1 is the worst performing normalization function in every situation. This behavior might be attributed to larger weights given to neighbors in the standard user-based recommendation formulation when the similarity is too high, which might occur too often when the similarity is not bounded, as is the case for the sim_1 function. Another important point is that the best two normalization functions are sim_2 and sim_3, which are the only ones that consider the length of both users when normalizing the original value. This is a clear indicator that this information is critical and should be used by any similarity normalization function based on LCS. We hope to explore in detail this aspect in the future.

Considering the results from the last experiment, we can conclude that most of the performance values reported in the previous experiments are far from optimal, which means that there should be some room for improvement, since we have shown that our model improves its performance when using normalization functions (more specif-

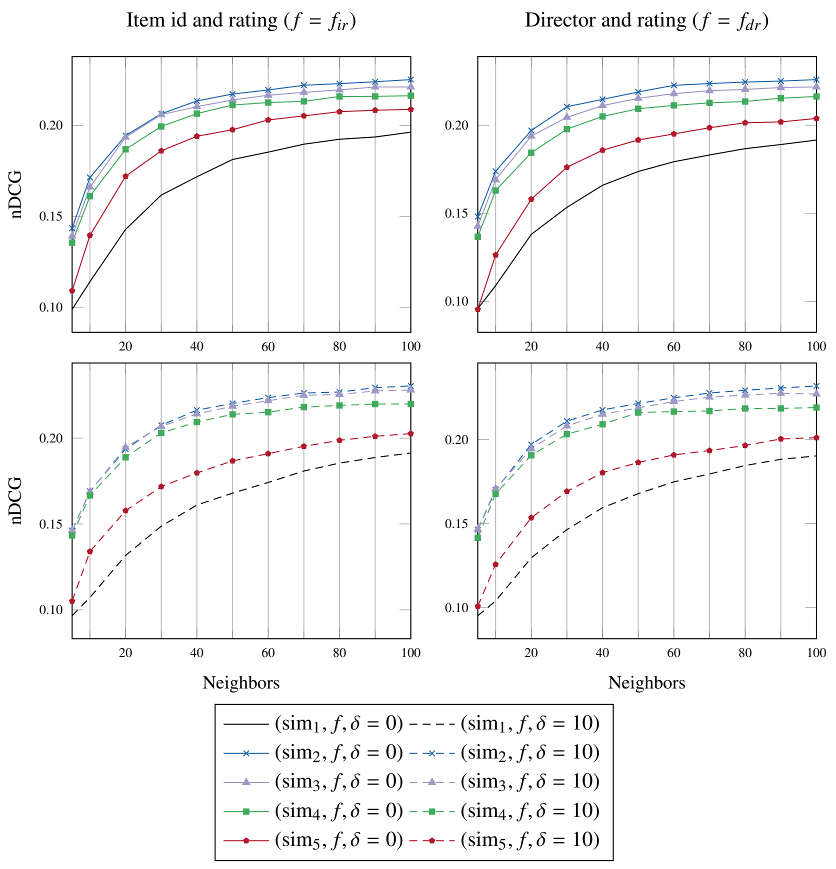

Figure 5: Results for different normalization functions.

ically, when using sim_2 and sim_3) and all the results reported were using no normalization, i.e., sim_1. Figure 6 shows the effect of all these parameters when used in combination compared against the setting when only one of the filters are used; in every case, the best value of each parameter is used (denoted, as mentioned in the caption, with the symbol $*$). From the figure, we make three observations: (a) the performance improvement between no normalization (sim_1) and normalization (sim_*) is always significant, as already shown in Figure 5; (b) the best performance is obtained when non-exact matching ($\delta = 10$) is allowed; (c) the optimal configurations are different depending on whether exact matchings are allowed: whereas for $\delta = 0$ the best configuration is using either the best confidence or preference filter value (very close to each other when using ratings and directors), for $\delta = 10$ the best configuration is consistently found for a combination of the best confidence and preference filter values.

At this point, we can provide the following answer to the second research question: **for pure collaborative filtering and content-based information, the best configuration in terms of ranking quality is achieved by combining the three best values**

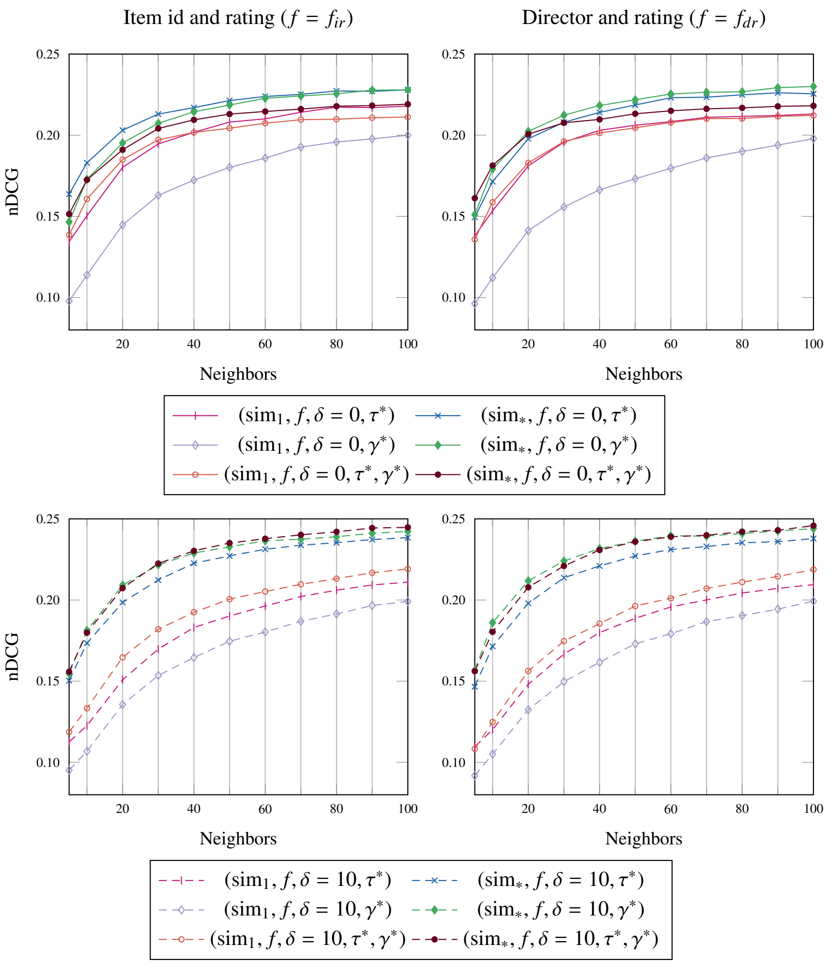

Figure 6: Results with the best confidence and preference filters and normalizations. Top row shows results using $\delta = 0$, bottom row using $\delta = 10$. The $*$ symbol denotes the best value among the previously reported ones is being used in the combination.

of normalization, preference, and confidence. More specifically, we achieve a 25% of improvement over the same recommender in nDCG without any of these variables, where the normalization function plays a critical role in this improvement.

Based on this, some general guidelines to select these different parameters can be derived, at least for rating-based datasets similar to the one used in the paper: normalization sim_2 always improves the results, exact matching ($\delta = 0$) works well with collaborative data whereas not exact matching shows better results for content-based data, and confidence and preference filters tend to improve the performance, although the actual value for the confidence filter should be tuned to the specific dataset (the preference filter can be automatically set to the user's mean for further generality).

Table 4: Performance of some of the most representative configurations of the proposed approach in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, α-nDCG, EILD, and Gini) at cutoff 5. The configuration for each recommender is denoted following the order $(\text{sim}, f, \delta, \tau, \gamma)$, that is: normalization function, transformation function, threshold for δ-matching, confidence filter, preference filter. The neigborhood size in every case is $k = 100$.

Recommender	nDCG	P	R	MAP	EPC	EPD	AD	α-nDCG	EILD	Gini
$(\text{sim}_1, f_{ir}, 0, 0, 0)$	0.196	0.130	0.135	0.086	0.495	0.736	1.06%	0.146	0.693	0.003
$(\text{sim}_1, f_{ir}, 10, 0, 0)$	0.191	0.128	0.131	0.082	0.506	0.734	1.00%	0.143	0.690	0.003
$(\text{sim}_1, f_{dr}, 0, 0, 0)$	0.192	0.128	0.130	0.083	0.501	**0.737**	1.05%	0.142	0.697	0.003
$(\text{sim}_1, f_{dr}, 10, 0, 0)$	0.190	0.128	0.130	0.081	0.508	0.735	0.99%	0.142	0.692	0.003
$(\text{sim}_1, f_{ir}, 0, 0, \bar{u})$	0.200	0.133	0.136	0.087	0.493	0.735	1.09%	0.149	0.697	0.003
$(\text{sim}_1, f_{dr}, 10, 0, \bar{u})$	0.199	0.133	0.137	0.086	**0.509**	0.732	1.04%	0.148	**0.698**	0.003
$(\text{sim}_1, f_{ir}, 0, 50, 0)$	0.218	0.176	0.106	0.069	0.237	0.300	3.99%	0.066	0.284	**0.010**
$(\text{sim}_1, f_{dr}, 0, 50, 0)$	0.213	0.171	0.108	0.069	0.258	0.328	**4.05%**	0.071	0.311	0.009
$(\text{sim}_2, f_{ir}, 0, 0, 0)$	0.225	0.146	0.154	0.102	0.476	0.730	1.19%	0.170	0.691	0.003
$(\text{sim}_2, f_{dr}, 10, 0, 0)$	0.232	0.152	**0.159**	**0.105**	0.484	0.724	1.32%	**0.175**	0.683	0.003
$(\text{sim}_1, f_{ir}, 10, 70, \bar{u})$	0.219	0.172	0.114	0.074	0.288	0.370	3.30%	0.083	0.356	0.007
$(\text{sim}_1, f_{dr}, 10, 50, \bar{u})$	0.219	0.166	0.127	0.082	0.350	0.470	2.98%	0.108	0.451	0.005
$(\text{sim}_2, f_{ir}, 10, 50, \bar{u})$	0.245	**0.187**	0.140	0.093	0.338	0.457	3.29%	0.119	0.433	0.007
$(\text{sim}_2, f_{dr}, 10, 30, \bar{u})$	**0.246**	0.179	0.152	0.101	0.406	0.571	2.76%	0.150	0.538	0.005

4.3.3. Impact on beyond-accuracy metrics

It is a well-known issue in the recommender systems literature that high accuracy or effectiveness in ranking metrics is difficult to balance with other dimensions of evaluation such as diversity and novelty [32]. One paradigmatic example of this behavior is a recommender that suggests the most popular items: it usually shows high effectiveness but at the expense of producing not diverse (always the same items are recommended) and not novel (novelty is usually defined as the inverse function of popularity) recommendations.

Table 4 shows some of the most representative configurations of the user similarity proposed and analyzed in previous sections. The tradeoff between diversity or novelty and accuracy is obvious from the table: it is not possible to find a configuration that optimizes at the same time novelty (EPC and EFD metrics) or diversity (AD, α-nDCG, EILD, and Gini) and accuracy (nDCG, precision, recall, and MAP). Nonetheless, we observe some interesting patterns from these results. First, the confidence filter τ affects negatively the novelty of the recommendations. This might be attributed to less long-tailed items being recommended because users with more items in common with the target user are preferred, hence leaving less room for novel items with respect to the target user. At the same time, this parameter seems to produce more globally diverse recommendations, evidenced by the highest value achieved in AD and Gini for a configuration using $\tau = 50$, especially when compared against the same configuration using $\tau = 0$, where AD is only 1.06% and Gini has a value of 0.003. This is not the case, however, for those metrics that measure the diversity between the items in the recommended list (i.e., α-nDCG and EILD), since higher values of τ will consider neighbors more similar to the target user, reducing the potential to recommend diverse items.

Second, the preference filter parameter seems to have no effect on the diversity and novelty aspects of the recommendation lists being generated. As already discussed in previous sections, where we found that this parameter does not change significantly the quality of the recommendations, this is not a negative aspect *per se*, since it means

Table 5: Performance of baselines and representative configurations of the proposed approach following the notation introduced in Table 4 in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, α-nDCG, EILD, and Gini). Statistical significant improvements (according to Wilcoxon test, $p < 0.05$) between the proposed approaches and each baseline with nDCG is denoted as a superscript.

Recommender	nDCG	P	R	MAP	EPC	EPD	AD	α-nDCG	EILD	Gini
1-Random	0.000	0.000	0.000	0.000	**0.972**	**0.756**	**63.47%**	0.001	**0.745**	**0.478**
2-Pop	0.160	0.105	0.112	0.069	0.444	0.741	0.48%	0.123	0.700	0.002
3-IB-Cos	0.179	0.119	0.126	0.077	0.508	0.710	2.99%	0.145	0.672	0.004
4-IB-Jac	0.162	0.109	0.116	0.069	0.521	0.712	3.35%	0.132	0.660	0.004
5-UB-Cos	0.235	0.153	0.161	0.107	0.490	0.722	1.46%	0.177	0.678	0.004
6-UB-Jac	0.233	0.152	0.161	0.106	0.484	0.723	1.35%	0.176	0.682	0.003
7-UB-JMSD	0.222	0.145	0.154	0.101	0.482	0.724	1.39%	0.169	0.685	0.003
8-LDA	0.216	0.142	0.150	0.097	0.548	0.713	4.80%	0.165	0.650	0.010
9-MF	0.184	0.124	0.128	0.079	0.570	0.713	5.72%	0.140	0.644	0.012
10-PureCB	0.010	0.007	0.010	0.005	0.853	0.739	10.09%	0.028	0.657	0.020
11-CBCF	0.237	0.155	**0.162**	**0.108**	0.505	0.722	1.67%	**0.179**	0.666	0.004
$(\text{sim}_1, f_{ir}, 0, 0, 0)^{1-4,10}$	0.196	0.130	0.135	0.086	0.495	0.736	1.06%	0.146	0.693	0.003
$(\text{sim}_1, f_{dr}, 0, 0, 0)^{1-4,10}$	0.192	0.128	0.130	0.083	0.501	0.737	1.05%	0.142	0.697	0.003
$(\text{sim}_2, f_{dr}, 10, 30, \bar{u})^{1-11}$	**0.246**	**0.179**	0.152	0.101	0.405	0.570	2.67%	0.150	0.535	0.005

that equivalent suggestions might be generated with a fraction of the information being processed, allowing more efficient computations, producing more recommendations per cycle in real-time environments.

Third, the normalization functions produce a significant improvement in accuracy (as already discussed), and as we observe in this table, they do not seem to affect other metrics, since we can compare the same configuration with and without normalization and realize that only the ranking quality metrics are modified. This is a very interesting result regarding the diversity-accuracy tradeoff, since we can conclude that by changing the normalization function we can improve one of the dimensions (in this case, accuracy) without altering the others (diversity and novelty).

Therefore, we can finally answer the third research question by stating that, even though there is no single solution that solves the diversity-accuracy tradeoff, **some configurations evidence high values of novelty and diversity, while maintaining decent levels of accuracy**. In the next sections we extend this comparison with other algorithms from the state-of-the-art and discuss how competitive these results really are.

4.3.4. Performance comparison with other algorithms

To properly contextualize the results presented in the paper, we show in Table 5 the results for each of the baselines presented in Section 4.2. We observe that one of the content-based baselines (PureCB) obtains a much worse performance than the other, which is the best performing recommendation technique from the rest of the baselines (although the UB with Cosine achieves a similar performance). Consistent with previous results, content-based techniques produce more novel (less popular) recommendations, as evidenced by the higher values of EPC. Recall that this recommender retrieves items that match the features that are most liked for the user, so even if an item is not known, if its genres or directors are consumed by the user, the item will be recommended. However, as discussed in the previous section, it is not easy to produce novel and accurate recommendations: for instance, the random recommender also obtains very high values of novelty (and even diversity); this is probably the reason that

Table 6: Performance of baselines and representative configurations of the proposed approach following the notation introduced in Table 4 in Movielens10M dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, α-nDCG, EILD, and Gini). Statistical significant improvements denoted as in Table 5.

Recommender	nDCG	P	R	MAP	EPC	EPD	AD	α-nDCG	EILD	Gini
1-Random	0.000	0.000	0.000	0.000	**0.989**	**0.755**	**62.16%**	0.001	0.742	**0.491**
2-Pop	0.106	0.069	0.061	0.040	0.576	0.748	0.44%	0.078	**0.749**	0.001
3-IB-Cos	0.175	0.120	0.121	0.071	0.748	0.718	3.26%	0.137	0.643	0.005
4-IB-Jac	0.141	0.099	0.100	0.056	0.744	0.723	3.36%	0.111	0.637	0.005
5-UB-Cos	0.268	0.172	0.179	**0.124**	0.711	0.715	2.81%	**0.200**	0.662	0.005
6-UB-Jac	0.267	0.172	**0.180**	0.123	0.709	0.716	2.89%	0.199	0.667	0.005
7-UB-JMSD	0.106	0.072	0.068	0.040	0.616	0.753	1.73%	0.076	0.723	0.003
8-LDA	0.209	0.143	0.142	0.088	0.734	0.716	2.93%	0.157	0.660	0.007
9-MF	0.191	0.132	0.128	0.077	0.758	0.721	3.24%	0.146	0.654	0.008
10-PureCB	0.012	0.008	0.010	0.006	0.911	0.738	9.59%	0.030	0.659	0.020
11-CBCF	0.142	0.099	0.083	0.052	0.649	0.749	1.63%	0.100	0.703	0.004
$(\text{sim}_1, f_{ir}, 0, 0, 0)^{1-4,7-11}$	0.228	0.151	0.153	0.100	0.686	0.736	1.06%	0.164	0.680	0.003
$(\text{sim}_1, f_{dr}, 0, 0, 0)^{1-4,7-11}$	0.224	0.148	0.149	0.098	0.685	0.738	1.02%	0.159	0.684	0.003
$(\text{sim}_2, f_{dr}, 10, 30, \bar{u})^{1-11}$	**0.284**	**0.199**	0.177	0.124	0.587	0.593	2.31%	0.179	0.551	0.005

the PureCB recommender obtains values so high of EPC. However, as we can see, very high values in these metrics can lead to very low results in ranking evaluation.

Finally, regarding the diversity of the recommendations, PureCB outperforms the other techniques (if we do not take into account the random recommender as it obtains a value of 0 in all ranking evaluation metrics) in terms of Gini and AD followed by MF, the Popularity recommender achieves the highest value of EILD, and CBCF, followed by UB, obtain the highest values of α-nDCG, which combines accuracy and diversity into a single measure. These results make clear that it is very difficult to produce diverse recommendations under different definitions, since, like for novelty, each evaluation metric measures a different nuance of the concept of diversity.

Now, if we compare the results from Table 4 with those of Table 5, we find that every configuration of the proposed LCS-based user similarity outperforms some of the baselines, namely, Pop, IB, and PureCB. Furthermore, some of the configurations also outperform MF and, more importantly, UB, which is the real baseline to beat, since the proposed approach uses the same recommendation technique as UB but changing the user similarity metric. We note that, for the subset of configurations included in Table 5 – the two most basic configurations of our approach and the best combination (with respect to nDCG) –, the performance improvements are significant in every case. More specifically, the best LCS-based configurations outperform UB in terms of nDCG, precision, AD, and Gini, although this comparison might not be completely fair due to the different number of parameters that can be configured in each method.

Moreover, let us analyze now why pure classic collaborative filtering algorithms are not able to beat the best baseline, the hybrid CBCF. It is important to note that the HetRec dataset is biased to content-based information, since only users who have provided both ratings and tags are considered. For this reason, these baseline recommenders have been tested in an additional experiment where a large subset of the Movielens10M dataset – where the HetRec dataset was extracted from – was used instead, including many more ratings than the HetRec dataset, and hence favoring the collaborative filtering algorithms. In particular, this dataset was obtained by removing any item not included in the original HetRec dataset – which are the only ones with content features – to not put at an unnecessary strong disadvantage on the content-based methods. Then, those users not in the HetRec dataset were included in the training split, and for the remaining users, their ratings were randomly splitted following an 80-20 training-test partition.

The results obtained in this experiment are shown in Table 6. We observe the trend is clearly different: UB, IB, and MF algorithms outperform the hybrid baseline CBCF, which no longer benefits from having as much content-based information as before. Furthermore, when the same subset of configurations as before is compared in this context, we observe that our approaches are also able to outperform the hybrid recommender by a large margin. Additionally, the optimal configuration in HetRec is also comparable to the UB recommender – the best performing baseline in this case –, showing better performance in terms of nDCG and precision, where this difference is statistically significant at least for the nDCG metric.

Regarding the efficiency comparison of our approaches, it should be noted that the computational cost of LCS-based similarities is higher than classical user similarities such as Cosine. However, as noted in Section 3.4, such difference in computation time only affects the training time of the algorithms, which is possible to parallelize in any case, and hence, the time can be greatly reduced. More specifically, we have obtained that a user similarity metric based on Cosine needs around 135.8 seconds to compute all the similarities needed when training for the HetRec dataset, this time goes up to 2,304.5 seconds when using the most basic configuration of our LCS-based similarity. Interestingly, by using the preference filter this training time can be reduced to 793.8 seconds, while, as discussed before, the performance remains the same even though less data is being used.

In summary, for the fourth research question, we can conclude that **our approach is very competitive against other state-of-the-art recommenders**, both using a dataset heavily biased towards content-based information or under a more typical situation where more ratings are available. Specifically, very positive results have been found when the LCS-based user similarity uses either a pure collaborative filtering configuration or when content-based information is exploited.

5. Related work

As we pointed out in Section 3.2, LCS is an algorithm that has an application in biology [17] and file comparison [18]. However, researchers have not widely used it in recommendation. Notwithstanding this, there are some works where it is used as a pattern finding algorithm, as in [33], where the authors analyzed the potential of applying LCS in e-commerce applications by recommending items that might be relevant for the users, reporting metrics like precision, recall, and F1. In [34], the potential utility of LCS is introduced in an online Web Usage Mining system, obtaining a best case accuracy of 73%.

These are examples of applications of LCS in recommendation but, to the best of our knowledge, there are no other references using this algorithm as a similarity measure, as we propose in this paper. The only example we have found is our previous work, where we introduced LCS as a similarity measure in a rating-based recommender system but only using it as a pure collaborative filtering approach [14]. As we have shown in this paper, content-based information like directors or genres in the case of movies, can easily be added due to the algorithm's generality. Besides, we have proposed four new normalization techniques to improve the recommendations and two additional parameters were incorporated in the model that did not appear in that work.

Furthermore, there are other techniques that can be applied for pattern matching instead of LCS. Markov Chains seems to be a useful technique as shown in [35], achieving good performance under sparse conditions. In [36], the authors used personalized Markov Chains (one for each user) over sequential data, outperforming matrix factorization in both sparse and dense data.

6. Conclusions and future work

In this paper we have presented a generic framework to define sequences from the users' interactions by exploiting both collaborative and content-based information, allowing us to create hybrid recommender systems. We have also presented a user similarity metric for collaborative filtering recommendation that can deal with these sequences based on the Longest Common Subsequence (although our framework can also work with any other sequence similarity algorithm such as those defined in the string matching literature) and we have shown that it can produce competitive recommendations under many different contexts. Firstly, it produces better recommendations than other collaborative filtering techniques when the dataset is too biased towards content-based information and secondly, when this information is closer to other rating-based datasets, it outperforms both content-based and collaborative filtering algorithms.

Furthermore, by exploiting some of its parameters, the proposed approach could find a good balance between execution time, diversity or novelty, and accuracy. Specifically, we have improved the accuracy of our LCS similarity metric by introducing normalization functions without hurting the novelty and diversity and we have also found that the preference filtering parameter shows a huge potential, since it does not decrement the performance of the approach but it helps to reduce its computational load, improving its efficiency. Nonetheless, considering the results obtained, we believe there is still room for improvement by exploring additional contextual dimensions, which are straightforward to incorporate due to the generality of our sequence generation approach.

More importantly, we believe the proposed approach has further potential to be generalized in other datasets besides those based on ratings, such as the one tested here. As a matter of fact, both the confidence filter and the normalization functions can be applied to frequency-based datasets – where the interactions between the users and the system are not bounded – and one-class datasets – where only one signal is available in the dataset without repetitions or negative feedback. Additionally, similarity metrics based on the LCS algorithm – or other string similarities or sequence alignment algorithms such as Jaro similarity or Smith-Waterman [19] – naturally incorporate sequences with repeated elements. Even though this aspect is not usually taken into account in the recommendation field, in many cases (e.g., for music suggestions or point-of-interest recommendation [37, 38]) it is common for the user to consume the same item several times; thus, this can be an interesting characteristic of this type of algorithms. Hence, we aim to explore these types of datasets in the future and analyze if these parameters are so beneficial in those contexts as they prove to be in rating-based datasets.

Finally, the general model presented here might be further extended in the future with more transformations – so that other features (like demographic information, tags, etc.) can be exploited – and other methods to order the sequence, especially those using the temporal dimension whenever it is available, opening up new possibilities to provide time-aware recommendations [39] under an integrated formulation that is not possible with classical similarity metrics.

7. Acknowledgements

This article was supported by the Spanish Ministry of Economy, Industry and Competitiveness (TIN2016-80630-P).

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] C. Xu, A novel recommendation method based on social network using matrix factorization technique, Inf. Process. Manage. 54 (3) (2018) 463–474.

[3] M. de Gemmis, P. Lops, C. Musto, F. Narducci, G. Semeraro, Semantics-aware content-based recommender systems, in: Recommender Systems Handbook, Springer, 2015, pp. 119–159.

[4] M. de Gemmis, P. Lops, G. Semeraro, C. Musto, An investigation on the serendipity problem in recommender systems, Inf. Process. Manage. 51 (5) (2015) 695–717.

[5] J. Xu, Y. Yao, H. Tong, X. Tao, J. Lu, Rapare: A generic strategy for cold-start rating prediction problem, IEEE Trans. Knowl. Data Eng. 29 (6) (2017) 1296–1309.

[6] O. Kassák, M. Kompan, M. Bieliková, Personalized hybrid recommendation for group of users: Top-n multimedia recommender, Inf. Process. Manage. 52 (3) (2016) 459–477.

[7] 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.

[8] J. L. Herlocker, J. A. Konstan, J. Riedl, An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms, Inf. Retr. 5 (4) (2002) 287–310.

[9] 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.

[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] P. Cremonesi, Y. Koren, R. Turrin, Performance of recommender algorithms on top-n recommendation tasks, in: X. Amatriain, M. Torrens, P. Resnick, M. Zanker (Eds.), Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010, ACM, 2010, pp. 39–46.

[12] H. Liu, Z. Hu, A. U. Mian, H. Tian, X. Zhu, A new user similarity model to improve the accuracy of collaborative filtering, Knowl.-Based Syst. 56 (2014) 156–166.

[13] J. Niu, L. Wang, X. Liu, S. Yu, FUIR: fusing user and item information to deal with data sparsity by using side information in recommendation systems, J. Network and Computer Applications 70 (2016) 41–50.

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

[15] Y. Shi, M. Larson, A. Hanjalic, Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges, ACM Comput. Surv. 47 (1) (2014) 3:1–3:45.

[16] A. Bellogín, J. Wang, P. Castells, Structured collaborative filtering, in: C. Macdonald, I. Ounis, I. Ruthven (Eds.), Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011, ACM, 2011, pp. 2257–2260.

[17] A. Apostolico, String editing and longest common subsequences, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 2, Springer-Verlag New York, Inc., New York, NY, USA, 1997, pp. 361–398.

[18] L. Bergroth, H. Hakonen, T. Raita, A survey of longest common subsequence algorithms, in: P. de la Fuente (Ed.), Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000, A Coruña, Spain, September 27-29, 2000, IEEE Computer Society, 2000, pp. 39–48.

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

[20] B. M. Marlin, R. S. Zemel, S. T. Roweis, M. Slaney, Collaborative filtering and the missing at random assumption, in: R. Parr, L. C. van der Gaag (Eds.), UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007, AUAI Press, 2007, pp. 267–275.

[21] H. Steck, Training and testing of recommender systems on data missing not at random, 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. 713–722.

[22] 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.

[23] I. Cantador, P. Brusilovsky, T. Kuflik, Second workshop on information heterogeneity and fusion in recommender systems (Hetrec2011), 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. 387–388.

[24] 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.

[25] 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.

[26] 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.

[27] J. Bobadilla, F. Serradilla, J. Bernal, A new collaborative filtering metric that improves the behavior of recommender systems, Knowl.-Based Syst. 23 (6) (2010) 520–528.

[28] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, Journal of Machine Learning Research 3 (2003) 993–1022.

[29] T. Hofmann, Latent semantic models for collaborative filtering, ACM Trans. Inf. Syst. 22 (1) (2004) 89–115.

[30] M. J. Pazzani, A framework for collaborative, content-based and demographic filtering, Artif. Intell. Rev. 13 (5-6) (1999) 393–408.

[31] M. Balabanović, Y. Shoham, Fab: Content-based, collaborative recommendation, Commun. ACM 40 (3) (1997) 66–72.

[32] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, Y.-C. Zhang, Solving the apparent diversity-accuracy dilemma of recommender systems, Proceedings of the National Academy of Sciences 107 (10) (2010) 4511–4515.

[33] Y. Sneha, G. Mahadevan, M. M. Prakash, An online recommendation system based on web usage mining and semantic web using lcs algorithm, in: Electronics Computer Technology (ICECT), 2011 3rd International Conference on, Vol. 2, IEEE, 2011, pp. 223–226.

[34] M. Jalali, N. Mustapha, M. N. Sulaiman, A. Mamat, A web usage mining approach based on LCS algorithm in online predicting recommendation systems, in: 12th International Conference on Information Visualisation, IV 2008, 8-11 July 2008, London, UK, IEEE Computer Society, 2008, pp. 302–307.

[35] T. Tran, D. Q. Phung, S. Venkatesh, Collaborative filtering via sparse markov random fields, Inf. Sci. 369 (2016) 221–237.

[36] 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.

[37] Y. Liu, T. Pham, G. Cong, Q. Yuan, An experimental evaluation of point-of-interest recommendation in location-based social networks, PVLDB 10 (10) (2017) 1010–1021.

[38] H. Wang, M. Terrovitis, N. Mamoulis, Location recommendation in location-based social networks using user check-in data, in: SIGSPATIAL/GIS, ACM, 2013, pp. 364–373.

[39] 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.