Collaborative Filtering based on Subsequence Matching: A New Approach

Alejandro Bellogín[a], Pablo Sánchez[a]

[a]*Escuela Politécnica Superior, Universidad Autónoma de Madrid Madrid, Spain*

Abstract

Neighbourhood-based techniques, although very popular in recommendation systems, show different performance results depending on the specific parameters being used; besides the neighbourhood size, a critical component of these recommenders is the similarity metric. Therefore, by considering more information associated to the users – such as taking into account the ordering of the items as they were consumed or the whole interaction pattern between users and items – it should be possible to define more complete, and better performing, similarity metrics for collaborative filtering. In this paper we propose a technique to compare users – also extendable to items –, working with them as sequences instead of vectors, hence enabling a new perspective to analyse the user behaviour by finding other users who have similar sequential patterns instead of focusing only on similar ratings in the items. We also compare our approach with other well-known techniques, showing comparable or better performance in terms of rating prediction, ranking evaluation, and novelty and diversity metrics. According to the results obtained, we believe there is still a lot of room for improvement, due to its generality and the good performance obtained by this technique.

Keywords: Collaborative Filtering, User Similarity, Longest Common Subsequence, Interaction pattern

1. Introduction

With the exponential growth of the population that have access to the Internet in the last years, the recommender systems need to adapt and innovate in the way

Email address: `alejandro.bellogin@uam.es` (Alejandro Bellogín)

they suggest items to the users. Neighbourhood-based systems continue being a popular approach due to their simplicity and positive performance. Nevertheless, most of these approximations work with users as vectors (obtained from the items they have rated) applying correlations and distances between them such as Pearson correlation and Cosine similarity; those users with higher correlation values or lower distances will be used as neighbours during the recommendation process.

As an alternative, we can further analyse the user profiles, studying different representation models so that novel and more nuanced concepts could be captured by these representations. In this work, we treat the users as *sequences* of interactions, allowing us to operate with more information about the user profiles, and additionally observe the behaviour of the recommender algorithms using different ways to sort the items and capture interaction patterns. We aim to do this by defining and testing new similarity measures on top of classical nearest-neighbourhood recommenders. Then, as the goal is to compare how similar a sequence (a user or item) is to another, we adapt a well-known algorithm that is typically used to compute the *longest common subsequence* (LCS) between character strings.

Hence, in this paper, we present different ways to obtain these sequences from the users' interactions, either using their ratings, items and ratings, or only their items. Besides, as the length of a subsequence between two users is not a bounded value, we propose a normalisation technique to bound LCS results in $[0, 1]$. We have also adapted the LCS algorithm so that not-exact matchings are required between two user sequences. Therefore, the main contributions of this paper are:

- A new algorithm that can be used as a similarity metric to compare two users (also applicable to items) of a recommender system, by transforming them into interaction sequences.

- Different models to represent users as sequences, together with additional configuration parameters that further generalise the LCS algorithm.

- A thorough comparison between popular algorithms used in collaborative filtering approaches and our proposal using rating prediction metrics, ranking evaluation techniques, and novelty and diversity metrics in two different datasets.

This paper is organised as follows: in Section 2 we introduce a more detailed explanation about neighbourhood-based recommender systems and the standard LCS algorithm. Section 3 presents our proposal to integrate the computation of LCS as a similarity metric in Collaborative Filtering. Then, in Section 4 we show the procedures followed to evaluate and compare the proposed metric with other classical similarities used in the literature. Finally, Section 5 presents alternative studies that work with LCS for recommendation, and in Section 6 we summarise the main conclusions obtained in this paper and discuss about potential future work.

2. Background

2.1. Recommender Systems

The aim of Recommender Systems (RS) is to assist users in finding their way through huge databases and catalogues, by filtering and suggesting relevant items taking into account the users' preferences (i.e., tastes, interests, or priorities). Collaborative Filtering (CF) systems can be considered as the earliest and most widely deployed recommendation approach [23], suggesting interesting items to users based on the preferences from "similar" people [27]. Other types of recommendation algorithms include content-based systems – that suggest items similar to those the user preferred or liked in the past and based on content features of the items in the system [24] –, demographic systems – where users are categorised based on their demographic attributes [2] –, social filtering systems – exploiting contacts, interactions, and trust between users [18, 31] –, and hybrid recommenders – where different techniques are combined [7, 13].

Techniques performing CF recommendation are typically classified into two main categories: model-based and memory-based. Model-based approaches build statistical models of user/item interaction patterns to provide automatic predictions [23]; memory-based algorithms, on the other hand, make predictions based on the entire collection of interactions, usually by computing similarities between users or items and taking those similarities into account when producing the recommendations [27]. In this paper, we are going to focus on the second type of CF algorithms, also known as *nearest-neighbour recommender systems* since they exploit those similarities to rank the users/items and use the closest ones to generate recommendations.

More specifically, this is the *standard* definition for a user-based nearest neighbour algorithm (UB):

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} r_{vi} w_{uv}}{\sum_{v \in N(u)} |w_{uv}|} \tag{1}$$

where $N(u)$ is the neighbourhood of user u (the most similar users according to some similarity metric), r_{vi} is the rating given by user v to item i, w_{uv} denotes the similarity between u and v, and $\hat{r}_{ui}$ is the predicted rating according to this formulation. Alternatively, it is possible to predict ratings using an item-based nearest neighbour formulation (IB):

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} r_{uj} w_{ij}}{\sum_{j \in N(i)} |w_{ij}|} \tag{2}$$

It is also possible to incorporate the user or item deviation in the formulation. These techniques have usually obtained good results when error metrics are used, however for ranking-based metrics (like precision) its performance decreases. Because of that, some authors have proposed variations tailored to ranking tasks that can also be applied to implicit feedback datasets [12, 1], where the main difference with respect to the previous formulation is that the summation is not normalised:

$$\hat{r}_{ui} = \sum_{v \in U} \mathbb{1}_{v \in N_K(u)} r_{vi} w_{uv} \tag{3}$$

Note that UB and IB algorithms have two critical parameters: the size of the neighbourhood being considered and the similarity metric. Similarities are usually based on distance and correlation metrics [27]. We present here two of the most popular ones when rating data is available – cosine and Pearson similarities, although other formulations have also present competitive results [6]:

$$\cos(u, v) = \frac{\sum_{i \in I(u,v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2 \sum_{i \in I(v)} r_{vi}^2}} \tag{4}$$

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

where $\bar{r}_u$ denotes the user's average rating, $I(u)$ represents the items rated by user u, and $I(u, v) = I(u) \cap I(v)$ those rated by both users.

2.2. Longest Common Subsequence

The Longest Common Subsequence (LCS) problem arises in a number of applications, from text editing to molecular sequence comparisons, and has been extensively studied [3]. It is specifically defined as follows: given a string x over an alphabet $\Sigma = (\sigma_1, \cdots, \sigma_s)$, a *subsequence* of x is any string w that can be obtained from x deleting zero or more (not necessarily consecutive) symbols. The

Algorithm 1 Longest Common Subsequence

```
 1: procedure LCS($x, y$)                                ▷ The LCS of $x$ and $y$
 2:     $L[0 \cdots m, 0 \cdots n] \leftarrow 0$
 3:     for $i \leftarrow 1, m$ do
 4:         for $j \leftarrow 1, n$ do
 5:             if $x_i = y_j$ then
 6:                 $L[i, j] \leftarrow L[i-1, j-1] + 1$        ▷ There is a matching
 7:             else
 8:                 $L[i, j] \leftarrow \max(L[i, j-1], L[i-1, j])$
 9:             end if
10:         end for
11:     end for
12:     return $L[m, n]$    ▷ $L[i, j]$ contains the length of an LCS between $x_1 \ldots x_i$
    and $i_1 \ldots y_j$
13: end procedure
```

LCS problem for input strings $x = x_1 \cdots x_m$ and $y = y_1 \cdots y_n$ (assuming $m \leq n$) consists of finding a third string $w = w_1 \cdots w_l$ such that w is a subsequence of x and also a subsequence of y, and w is of maximum possible length. In general, such w is not unique.

Computationally, lower bounds for this problem are time $\Omega(n \log n)$ or linear time, according to whether the size of Σ is unbounded or bounded [3]. Time $\Theta(mn)$ is achieved by the dynamic programming algorithm described in Algorithm 1, from [20]. If only the length of an LCS is needed, then this code can be adapted to use only linear space. The basic observation for this is that the computation of each row of matrix L only needs the preceding row. Otherwise, if the complete LCS is required then the whole matrix will be used to produce the desired output by backtracking.

3. An LCS-based similarity metric for Collaborative Filtering

In this paper, we propose an LCS-based similarity metric, where the computation of LCS – actually, its length – will be the same as in the literature for string matching (as described in Section 2.2). Hence, in order to adapt it into the CF domain, we need to define how to represent the users or items properly as sequences (Section 3.1) and, according to that representation, a corresponding function that identifies when two *characters* are the same (Section 3.2). Finally, as explained in Section 2.1, some CF algorithms might be sensitive to the normalisation of the similarity measures; hence, we present some alternatives in Section 3.3. In the rest of the paper, we shall focus on the problem of user similarity, we leave as future work the computation of item similarities using an analogous development.

3.1. Representing Users as Sequences

As introduced in Section 2.2, the LCS problem is defined upon two sequences of symbols (*strings*). Therefore, in order to apply this problem to CF we first need to define what a symbol is in our context, and how the 'strings' will be generated.

Let us take users $u, v \in \mathcal{U}$, the set of items rated/consumed/bought by one particular user shall be defined as $I(u) = \{(i, r) : (u, i, r) \mid r \neq \emptyset\}$ – that is, those items (and their corresponding interaction values) for which any interaction (rate, consume, buy, etc.) has been produced by the user u. Hence, to generate a string/sequence representation of users it makes sense to use the set $I(u)$ of rated items; nonetheless, there are multiple alternatives to transform $I(u)$ into symbols, depending on the considered alphabet Σ. We propose the following transformations:

- Using the item, i.e., $f_i : I(u) \to \Sigma = \mathcal{I}, f_i(x) = x(i)$.
- Using the value of the interaction, i.e., $f_r : I(u) \to \mathcal{R}, f_r(x) = x(r)$.
- Using a combination of the item and the interaction value, i.e., $f_{ir} : I(u) \to \mathcal{I} \times \mathcal{R}, f_{ir}(x) = (x(i), x(r))$.

In practice, if the range of the interactions is bounded (typically for ratings $\mathcal{R} = \{1, \cdots, 5\}$) and there is a mapping between items and their ids, it is possible to define these transformations so the output fits in an integer – for instance, in the third case, we may use the following transformation: $f_{ir}(x) = \tilde{f}_{ir}(x) = x(i) \cdot 10 + x(r)$, assuming $x(r) < 10$.

Note these transformations can also be applied when unbounded interactions are available in the system (document clicks, listening frequencies of an artist, page views, etc.) provided a proper transformation reducing the infinity range of such interactions to a discrete (bounded) domain [10, 28].

Once we have decided the alphabet that will be used when representing the users, we need to specify the order in which the symbols will be arranged – this is important since the LCS algorithm is aware of the order of both strings. In this paper, and as a first approach, we sort each user sequence according to the natural item ordering (i.e., ordered by item id); which actually corresponds to any other

Algorithm 2 Longest Common Subsequence with Rating Data

```
1:  procedure LCS_CF($u, v, f, \delta$)        ▹ 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
7:                  $L[i, j] \leftarrow L[i-1, j-1] + 1$        ▹ There is a $\delta$-matching
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
```

global ordering considered for the whole item collection at the same time, such as sorting the items according to their popularity, their novelty, or any other permutation (a comparison among these circumstances will be presented in Section 3.4). Other alternatives we leave as future work include taking into account the timestamp when the item was interacted with (in a user basis); although it should be noted that building time-based sequences of user preferences is not a trivial question, mostly because most of the available datasets do not contain meaningful timestamps (sometimes the timestamp is directly not available, or some of the profiles are not complete, the users could have most of their ratings in the very same second, or it could also happen that temporal splits of the datasets leave a very unbalanced training-test configuration from which it is very difficult to learn proper patterns [8]).

3.2. Sequence Matching with Rating Data

In the original LCS problem, an exact matching is sought between the two strings, however, when dealing with user data some level of fuzziness is desired, since it is assumed that two users do not have to show an identical behaviour to be considered good predictors of each other. Classical similarity measures (Cosine, Pearson correlation, etc.) address this issue by considering the distance between the values provided by each user (see Equations 4 and 5). Because of this, we propose a variation on the LCS algorithm to relax the matching condition, in such a way that a *matching threshold* δ is used to decide whether two symbols of the sequence are equivalent – i.e., a matching is found. Algorithm 2 presents this variation, including a boolean function `match`(a, b, δ) that outputs `True` if a and b have the same value or their difference is lower than δ.

3.3. Similarity Normalisation

The last component of the similarity metric we propose herein deals with its normalisation. As we described in Section 2.1, the classical formulation of neighbourhood-based recommenders normalises the scores in such a way that the range of the similarity metric does not affect the final result (see Equation 1). However, non-normalised versions (see Equation 3) of these recommenders obtain better performance values when evaluated on top-N recommendation tasks [12, 1], and hence, they are being used more frequently nowadays.

At the same time, the LCS algorithm presented before generates a number ranging from 0 to the length of the alphabet (for instance, for $\Sigma = \mathcal{I}$ the maximum LCS would be $|\mathcal{I}|$). It might be important – depending on the considered alternative of the neighbourhood-based recommender – whether the LCS value is used as such or if it is somehow normalised. Because of this, we propose using our LCS-based similarity metric with and without normalisation (sim_2 and sim_1, respectively). Specifically, we normalise the LCS value between two users (considering any transformation function f and matching threshold δ) using the length of the two sequences involved:

$$\text{sim}_1^{f,\delta}(u, v) = \text{LCS\_CF}(u, v, f, \delta) \tag{6}$$

$$\text{sim}_2^{f,\delta}(u, v) = \frac{\text{sim}_1^{f,\delta}(u, v)^2}{|f(u)| \cdot |f(v)|} \tag{7}$$

In this way, $\text{sim}_1^{f,\delta}(u, v) \in [0, |\Sigma|]$, whereas $\text{sim}_2^{f,\delta}(u, v) \in [0, 1]$.

3.4. Toy Example

In this section, we present a small example to show how the different variations of the LCS-based similarity would be computed. Table 1 presents the interactions between users and items considered for this example, whereas Table 2 shows the similarity values in each case, together with the corresponding transformation f and matching threshold δ.

	×	○	△	□	◇
u	4		5	3	1
v	4	5		4	4

Table 1: Interaction (ratings) data between two users and five items.

f	δ	$f(u)$	$f(v)$	$\text{sim}_1^{f,\delta}(u,v)$	$\text{sim}_2^{f,\delta}(u,v)$
f_i	0	(×, △, □, ◇)	(×, ○, □, ◇)	3	0.56
f_r	0	(4,5,3,1)	(4,5,4,4)	2	0.25
	1			3	0.56
f_{ir}	0	(×4, △5, □3, ◇1)	(×4, ○5, □4, ◇4)	1	0.06
	1			2	0.25

Table 2: Representation and LCS values for different transformation functions and matching thresholds based on data from Table 1.

The values of $\text{sim}_1^{f,\delta}(u,v)$ from Table 2 are computed using Algorithm 2 (see Equation 6). When $f = f_i$, the LCS between u and v is (×, □, ◇), which has a length of 3, the value of $\text{sim}_1^{f,\delta}(u,v)$; then, the normalised similarity $\text{sim}_2^{f,\delta}(u,v)$ is computed as in Equation 7, which uses the length of each transformed user (in this case, $|f(u)| = |f(v)| = 4$), resulting in a similarity value of 0.5625. It is interesting to observe that the similarity computed using this transformation is equivalent to computing the item overlap between the two users, which is the basis for several metrics such as similarities based on Jaccard or item co-occurrence [1, 21].

From Table 2 we first observe that, for a fixed matching threshold, the representation ($f(u)$ and $f(v)$) does not change. We also note that each transformation function f may generate a different representation, which, in any case, will be treated by the LCS algorithm transparently, as long as such algorithm is able to deal with the symbols comprising each representation and is capable to decide if a matching has occurred. Finally, we notice in this example that $\text{sim}_2^{f,\delta}(u,v)$ is always determined by the value of $\text{sim}_1^{f,\delta}(u,v)$ (for fixed users u, v), this is because the proposed representation functions do not alter the size of the user vectors; should we have a function f that depending on the item outputs a different number of components (e.g., the genres a movie belongs to) then $f(u)$ would change across representations and the same value of $\text{sim}_1^{f,\delta}(u,v)$ would be normalised to different values of $\text{sim}_2^{f,\delta}(u,v)$.

Tables 3 and 4 evidence the comment made at the end of Section 3.2: sorting the items by different criteria has no effect in the LCS computation (as long as this

f	δ	$f(u)$	$f(v)$	$\text{sim}_1^{f,\delta}(u,v)$	$\text{sim}_2^{f,\delta}(u,v)$
f_i	0	(×, □, ◇, △)	(×, □, ◇,○)	3	0.56
f_r	0	(4,3,1,5)	(4,4,4,5)	2	0.25
	1			3	0.56
f_{ir}	0	(×4, □3, ◇1,△5)	(×4, □4, ◇4,○5)	1	0.06
	1			2	0.25

Table 3: Representation and LCS values for different transformation functions and matching thresholds based on data from Table 1. Items ordered by popularity (number of ratings).

f	δ	$f(u)$	$f(v)$	$\text{sim}_1^{f,\delta}(u,v)$	$\text{sim}_2^{f,\delta}(u,v)$
f_i	0	(△, ×, □, ◇)	(○, ×, □, ◇)	3	0.56
f_r	0	(5,4,3,1)	(5,4,4,4)	2	0.25
	1			3	0.56
f_{ir}	0	(△5, ×4, □3, ◇1)	(○5, ×4, □4, ◇4)	1	0.06
	1			2	0.25

Table 4: Representation and LCS values for different transformation functions and matching thresholds based on data from Table 1. Items ordered by inverse popularity.

sorting is made in the same way for all the items rated by the users, since that is equivalent to renaming the symbols for each item). We observe that the sequences built in the three cases (Tables 2, 3, and 4) are different but the similarity value (the length of the Longest Common Subsequence) is the same. These similarity metrics with different item orderings, when incorporated into a user-based nearest neighbour algorithm (Equation 1), produce the same recommendations because the neighbours found for each user are the same, since the similarity between users do not depend on how the items are ordered.

4. Empirical Evaluation

We now present several experiments that allow us to analyse the behaviour of the LCS-based similarity metric under different scenarios (Section 4.2). Prior to that, in Section 4.1, we introduce the datasets used in our experiments and the evaluation methodology followed. We then summarise the main results and conclusions obtained (Section 4.3) and end this section with a discussion (Section 4.4) of the obtained results.

Table 5: Parameters used for the different baselines in each framework. The JMSDSimilarity is not part of any of the frameworks, but it was implemented on top of them (as explained in Section 4.1.1).

Framework	Recommender class	Other classes	Parameters
MF			
RankSys	MFRecommender	PLSAFactorizer	$k = 50$; 100 iterations
Mahout	SVDRecommender	SVDPlusPlusFactorizer	$k = 50$; 100 iterations
UB1			
RankSys	UserNeighborhoodRecommender	TopKUserNeighborhood VectorJaccardUserSimilarity	$k \in [5, 100]$; $q = 1$; dense = `True`
Mahout	GenericUserBasedRecommender	NearestNUserNeighborhood PearsonCorrelationSimilarity	$k \in [5, 100]$
UB2			
RankSys	UserNeighborhoodRecommender	TopKUserNeighborhood VectorCosineUserSimilarity	$k \in [5, 100]$; $q = 1$; dense = `True`; $\alpha = 0.5$
Mahout	GenericUserBasedRecommender	NearestNUserNeighborhood UncenteredCosineSimilarity	$k \in [5, 100]$
UB3			
RankSys	UserNeighborhoodRecommender	TopKUserNeighborhood JMSDSimilarity*	$k \in [5, 100]$; $q = 1$;
Mahout	GenericUserBasedRecommender	NearestNUserNeighborhood JMSDSimilarity*	$k \in [5, 100]$
IB1			
RankSys	ItemNeighborhoodRecommender	TopKItemNeighborhood VectorJaccardItemSimilarity	$k \in [5, 100]$; $q = 1$; dense = `True`
Mahout	GenericItemBasedRecommender	PearsonCorrelationSimilarity	
IB2			
RankSys	ItemNeighborhoodRecommender	TopKItemNeighborhood VectorCosineItemSimilarity	$k \in [5, 100]$; $q = 1$; dense = `True`; $\alpha = 0.5$
Mahout	GenericItemBasedRecommender	UncenteredCosineSimilarity	

4.1. Experimental Setup

4.1.1. Baselines

In this work, since we propose a user-based similarity metric, our main goal is to compare it against other similarity metrics in a user-based nearest-neighbour scenario; however, we shall also show the performance of other representative baselines, such as matrix factorisation, most popular items, and item-based nearest-neighbour recommenders [23, 27]. To make this comparison easier to reproduce, we have integrated our similarity metric into two recommendation frameworks (on top of the similarity interfaces provided by each of them) and used their implementations of user-based nearest-neighbour. Specifically, we have used Mahout[1] and RankSys[2], where we have also implemented another similarity metric as baseline, denoted as JMSD [6]:

$$\begin{aligned} \mathrm{JMSD}(u,v) &= \mathrm{Jaccard}(u,v) \times (1 - \mathrm{MSD}(u,v)) = \\ &= \frac{\|I(u) \cap I(v)\|}{\|I(u) \cup I(v)\|}\left(1 - \frac{\sum_{i \in I(u) \cap I(v)} (r^s_{ui} - r^s_{vi})^2}{\|I(u) \cap I(v)\|}\right) \end{aligned} \tag{8}$$

where r^s corresponds to a normalised rating in the $[0, 1]$ interval.

Interestingly, we have found very different results depending on the framework being used (something already observed in the literature [29]), and hence, the results from both frameworks will be reported separately. One possible reason for this is the fact that RankSys does not normalise the output of the CF algorithm – as proposed in [12] as an optimisation for ranking-oriented algorithms.

The notation and settings for the different baselines are included in Table 5.

Table 6: Statistics about the datasets used in the experiments.

Dataset	#users	#items	#ratings	Density
Lastfm	1,892	17,632	92,834	0.28%
MovieLens	2,113	10,197	855,598	3.97%

4.1.2. Datasets

In the evaluation of the recommendation methods, we have used two publicly available datasets from two different domains: movies (MovieLens-HetRec)

[1]See `https://mahout.apache.org/`.

[2]See `http://ranksys.org/`.

and music (Lastfm-HetRec)[3]. Table 6 shows the basic characteristics about these datasets. We performed a 5-fold cross-validation evaluation, where 80% of the data is retained to train the recommenders, and the rest is used for the evaluation.

Furthermore, since the ratings in MovieLens are made on a 5-star scale with half-star increments (from 0.5 stars to 5.0 stars), every transformation function will take such rating and multiply it by 10 (instead of what was shown in Section 3.1), and then the item id will be multiplied by 100. Prior to this transformation, for the Lastfm dataset, we transform the frequency numbers to ratings in a similar way as shown in [10] and [30]. Different from other Lastfm datasets, this version stores the aggregate frequency of a user towards an item (an artist), and hence it is not possible to obtain or exploit any temporal information regarding such interaction, nonetheless this information allows pure collaborative filtering algorithms – like the one proposed in this paper – to work as usual. It is important to note that, if the raw frequencies are being used instead, algorithms where a Normal distribution is assumed for the ratings are not recommended, but approaches where a Poisson distribution is exploited are more appropriate, like the one presented in [16].

4.1.3. Evaluation

We have applied both rating- and ranking-based evaluation metrics; for the computation of the latter, we followed the methodology called *TrainItems* in [29], where every item in the system is considered as a candidate to be part of a user's ranking – except those already seen by such user in the training set. We then applied standard evaluation metrics, either rating-based: mean absolute error (MAE) and root mean squared error (RMSE) [17]; or ranking-based at different cutoffs: precision, recall, normalised discounted cumulative gain (nDCG), and mean average precision (MAP) [4].

We also report alternative evaluation dimensions besides accuracy and precision, such as novelty and diversity. For this, we computed the following metrics, all of them available in the RankSys framework and presented in detail in [9, 35], where the document features correspond to the genres in MovieLens and tags in Lastfm:

- α-nDCG: a diversity-aware ranking metric where the score of retrieved documents is penalised if they share features with documents ranked higher in the list.

[3]Both datasets are available at `http://grouplens.org/datasets/hetrec-2011/`.

- Aggregate diversity (AD): it accounts for the total number of items that the system recommends.
- EILD (expected intra-list diversity): rank-sensitive and rank-aware expected intra-list diversity metric.
- EFD (expected free discovery): expected ICF (inverse collection frequency) of (relevant and seen) recommended items.
- 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.
- ERR-IA (intent-aware expected reciprocal rank): generalisation of α-nDCG where the document features are not assumed to be equally probable.
- Gini (or sales diversity in [36]): it takes into account not only whether items are recommended to someone, but to how many people and how even or unevenly distributed.

Among these metrics, EFD, EPC, and EPD aims to measure the novelty of the recommendation list, whereas the rest are diversity-oriented metrics. In the reported results, the larger these values, the more diversity or novelty is being measured for that recommender.

4.2. Results

We present now the conducted experimentation, along with the obtained results, in order to validate our contributions and answer the following research questions: (i) Which instantiation (combination of transformation, normalisation, and matching threshold) of a similarity based on the Longest Common Subsequence (LCS) algorithm is better aligned with the CF problem? (ii) How does this similarity perform in terms of prediction accuracy (rating prediction task)? (iii) How effective is it in terms of precision-based metrics (item ranking task)? And (iv) what is the impact on beyond-accuracy metrics (diversity and novelty) when this similarity is used?

Table 7: Performance (nDCG@5) results in the Lastfm dataset using RankSys framework for different neighbourhood-based recommendation algorithms. Best value for each k in bold.

Recommender	5	10	20	30	40	50	60	70	80	90	100
f_i + 0 + sim$_1$	0.184	0.211	0.220	0.219	0.219	0.217	0.215	0.212	0.211	0.209	0.207
f_i + 0 + sim$_2$	**0.185**	**0.212**	**0.223**	**0.224**	**0.224**	**0.223**	**0.222**	**0.220**	**0.219**	**0.218**	**0.217**
f_{ir} + 0 + sim$_1$	0.164	0.187	0.198	0.199	0.199	0.199	0.198	0.197	0.195	0.193	0.192
f_{ir} + 0 + sim$_2$	0.161	0.186	0.200	0.203	0.203	0.203	0.203	0.203	0.202	0.202	0.202
f_{ir} + 10 + sim$_1$	0.183	0.205	0.217	0.216	0.217	0.214	0.211	0.209	0.207	0.205	0.204
f_{ir} + 10 + sim$_2$	0.184	0.207	0.219	0.221	0.222	0.220	0.220	0.218	0.216	0.215	0.214
f_r + 0 + sim$_1$	0.033	0.046	0.060	0.067	0.070	0.074	0.076	0.077	0.078	0.079	0.079
f_r + 0 + sim$_2$	0.033	0.046	0.060	0.067	0.071	0.074	0.076	0.077	0.079	0.079	0.080
f_r + 10 + sim$_1$	0.030	0.044	0.058	0.065	0.069	0.072	0.073	0.074	0.075	0.076	0.076
f_r + 10 + sim$_2$	0.030	0.044	0.058	0.065	0.070	0.072	0.074	0.075	0.076	0.076	0.077

Table 8: Performance (nDCG@5) results in the Lastfm dataset using Mahout framework for different neighbourhood-based recommendation algorithms. Best value for each k in bold.

Recommender	5	10	20	30	40	50	60	70	80	90	100
f_i + 0 + sim$_1$	0.156	**0.127**	0.093	**0.074**	0.063	**0.056**	**0.048**	0.043	**0.039**	**0.035**	**0.033**
f_i + 0 + sim$_2$	**0.158**	**0.127**	**0.094**	**0.074**	**0.064**	**0.056**	**0.048**	**0.044**	**0.039**	**0.035**	0.032
f_{ir} + 0 + sim$_1$	0.137	0.107	0.073	0.056	0.046	0.040	0.034	0.030	0.026	0.024	0.021
f_{ir} + 0 + sim$_2$	0.138	0.108	0.073	0.056	0.046	0.040	0.034	0.030	0.026	0.023	0.020
f_{ir} + 10 + sim$_1$	0.153	0.119	0.086	0.070	0.057	0.049	0.043	0.038	0.035	0.032	0.031
f_{ir} + 10 + sim$_2$	0.155	0.122	0.089	0.070	0.057	0.049	0.044	0.039	0.034	0.031	0.030
f_r + 0 + sim$_1$	0.033	0.026	0.013	0.007	0.004	0.003	0.002	0.002	0.001	0.001	0.001
f_r + 0 + sim$_2$	0.033	0.026	0.014	0.007	0.004	0.003	0.002	0.002	0.001	0.001	0.001
f_r + 10 + sim$_1$	0.033	0.028	0.015	0.007	0.004	0.003	0.002	0.002	0.001	0.001	0.001
f_r + 10 + sim$_2$	0.032	0.028	0.015	0.007	0.004	0.003	0.002	0.002	0.001	0.001	0.001

Table 9: Performance (nDCG@5) results in the MovieLens dataset using RankSys framework for different neighbourhood-based recommendation algorithms. Best value for each k in bold.

Recommender	5	10	20	30	40	50	60	70	80	90	100
f_i + 0 + sim_1	0.083	0.090	0.112	0.132	0.143	0.152	0.161	0.166	0.169	0.174	0.177
f_i + 0 + sim_2	0.143	0.164	0.188	0.202	0.210	0.213	0.217	0.218	0.221	0.221	0.221
f_{ir} + 0 + sim_1	0.099	0.114	0.143	0.162	0.172	0.181	0.185	0.190	0.192	0.194	0.196
f_{ir} + 0 + sim_2	0.143	**0.171**	**0.194**	0.206	0.213	0.217	0.219	0.222	0.223	0.224	0.225
f_{ir} + 10 + sim_1	0.097	0.107	0.132	0.149	0.161	0.168	0.174	0.181	0.185	0.189	0.191
f_{ir} + 10 + sim_2	**0.146**	0.169	**0.194**	**0.207**	**0.216**	**0.220**	**0.224**	**0.226**	**0.227**	**0.229**	**0.230**
f_r + 0 + sim_1	0.049	0.065	0.085	0.098	0.107	0.114	0.120	0.126	0.132	0.137	0.141
f_r + 0 + sim_2	0.094	0.114	0.137	0.148	0.155	0.159	0.163	0.165	0.166	0.168	0.169
f_r + 10 + sim_1	0.069	0.085	0.108	0.119	0.127	0.133	0.138	0.142	0.146	0.149	0.152
f_r + 10 + sim_2	0.084	0.108	0.128	0.141	0.148	0.154	0.157	0.160	0.164	0.165	0.167

Table 10: Performance (nDCG@5) results in the MovieLens dataset using Mahout framework for different neighbourhood-based recommendation algorithms. Best value for each k in bold.

Recommender	5	10	20	30	40	50	60	70	80	90	100
f_i + 0 + sim_1	0.028	0.006	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000
f_i + 0 + sim_2	0.078	0.035	0.013	0.007	0.004	0.003	0.002	0.002	**0.002**	0.001	0.001
f_{ir} + 0 + sim_1	0.031	0.015	0.005	0.003	0.001	0.001	0.000	0.000	0.000	0.000	0.000
f_{ir} + 0 + sim_2	0.086	0.045	0.017	0.009	**0.006**	0.004	0.003	0.002	**0.002**	0.001	0.001
f_{ir} + 10 + sim_1	0.032	0.015	0.005	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000
f_{ir} + 10 + sim_2	**0.087**	**0.047**	**0.020**	**0.011**	**0.006**	**0.004**	**0.004**	**0.003**	**0.002**	**0.002**	**0.002**
f_r + 0 + sim_1	0.013	0.005	0.002	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000
f_r + 0 + sim_2	0.051	0.027	0.010	0.006	0.003	0.002	0.002	0.001	0.001	0.001	0.001
f_r + 10 + sim_1	0.019	0.004	0.001	0.001	0.001	0.001	0.001	0.000	0.000	0.000	0.000
f_r + 10 + sim_2	0.049	0.027	0.010	0.006	0.004	0.003	0.003	0.002	**0.002**	**0.002**	**0.002**

4.2.1. Sensitivity Analysis

In this section, we analyse how the different parameters and configuration settings of the proposed metric affect its performance. For this, we shall compare the performance of a user-based nearest-neighbour recommender using some instantiations of the LCS-based similarity metric, where we test the three transformation functions f presented in Section 3.1, two values for the matching threshold δ (either 0 – exact matching – or 10 – allowing matching ratings within the range of one integer), and the normalised and raw values of the LCS algorithm (sim_2 and sim_1 respectively).

Tables 7 and 8 show the performance evolution when more neighbours are considered in RankSys and Mahout frameworks, respectively, in the Lastfm dataset. First we note that, in both cases, f_r as transformation function obtains the worst performance. This was expected, since defining two users as similar only based on how many times they used the same rating value (not taking into account the actual item such rating was applied to) does not make too much sense. However, taking into account the results that will be presented in the next section, it is interesting to observe that, in Mahout, this simple mechanism performs better than the item-based recommenders and UB2 (user-based kNN with Cosine similarity), and in some cases (smaller k's) it also outperforms UB1.

The best performing transformation function in Lastfm, for both frameworks, is f_i (although f_{ir} with $\delta = 10$ performs at the same level at some neighbourhood sizes); besides, it outperforms any other alternative in Mahout, whereas in RankSys it obtains a similar (sometimes better) performance to that of the user-based baselines.

On the other hand, for the MovieLens dataset (Tables 9 and 10) the best performing transformation function is f_{ir}, and this result is consistent for both frameworks. Again, like in Lastfm, the worst performing transformation function is f_r, although it produces competitive results when enough neighbours are considered.

From these tables, we observe that the most discriminating aspect in our proposed metric is the transformation function f: the matching threshold provides some improvement but in a much smaller range than the transformation function (the largest one with f_{ir} in RankSys for Lastfm with a 9.5% performance improvement); however, the use of normalised similarities typically improves the performance – more significantly with the RankSys framework, where, as noted before, the recommender does not normalise the final score – although its effect depends on the dataset: in MovieLens the improvements are as high as a 50% (RankSys and $f_{ir} + 10$), whereas in Lastfm there are several configurations where

Table 11: Configuration of recommendation algorithms that achieve best performance according to nDCG@5 in Lastfm (left) and MovieLens (right) datasets.

Recommender	Lastfm		MovieLens	
	RankSys	Mahout	RankSys	Mahout
UB1	30	5	100	20
UB2	30	5	90	20
UB3	20	5	100	5
IB1	100	(none)	5	(none)
IB2	100	(none)	5	(none)
LCS1 (f_i + 0 + sim$_2$)	40	5	100	5
LCS2 (f_{ir} + 10 + sim$_2$)	60	5	100	5

Table 12: Rating prediction performance for best recommenders using Mahout framework (see Table 11 for parameters) in Lastfm (left) and MovieLens (right) datasets.

(a) Lastfm

Recommender	MAE	RMSE
MF	1.199	1.568
UB1	1.166	1.449
UB2	1.218	1.492
UB3	1.018	1.298
IB1	1.483	1.910
IB2	1.172	1.409
LCS1	1.025	1.302

(b) MovieLens

Recommender	MAE	RMSE
MF	0.564	0.763
UB1	0.815	1.067
UB2	0.806	1.061
UB3	0.658	0.877
IB1	0.875	1.226
IB2	0.761	0.982
LCS2	0.636	0.841

the performance remains unchanged.

Based on these results, we can answer the first research question (which instantiation of the proposed metric is better aligned with the CF problem). As we discussed above, the f_i transformation function allows for larger improvements than any other of the tested transformations in Lastfm and f_{ir} in MovieLens; on the other hand, a larger matching threshold seems to provide some performance improvements, however it may introduce some noise; and finally, normalising the similarity metric does have an effect, but it depends on the dataset and the actual implementation of the user-based nearest-neighbour recommender (as we conclude from the very different results obtained from comparing Mahout and RankSys frameworks). Hence, we summarise these observations by presenting f_i + 0 + sim$_2$ as the optimal instantiation of the LCS-based metric for Lastfm and f_{ir} + 10 + sim$_2$ for MovieLens.

4.2.2. Rating Prediction Task

In this section we compare the performance of the proposed metric against other baselines in terms of rating prediction accuracy. This task was very important in the seminal papers of the area, but its interest has decreased after the appreciation that accuracy (as measured by error metrics like MAE and RMSE) does not correlate with user satisfaction at the same level as ranking-based metrics [26, 11]. Nonetheless, for the sake of comparison with classical papers, we present in Table 12 the results obtained for the best instantiations of each recommender in Mahout. Note that we only present results for this framework because RankSys does not normalise the predicted score, and hence, it does not produce scores in the range of ratings. The actual parameters used for the baselines are included in Table 11 – optimised, like the LCS-based recommenders, for nDCG@5 across the parameters' range presented in Table 5 –, where LCS1 denotes the optimal recommender in Lastfm and LCS2 the optimal one in MovieLens.

We observe that LCS1 shows a very good predictive accuracy in Lastfm, and very competitive results in MovieLens. This difference might be attributed to the different source of ratings in each dataset: whereas ratings in MovieLens come from real users, ratings in Lastfm have been artificially produced from implicit interactions by the users (see Section 4.1). In any case, both LCS-based similarities outperform classical nearest-neighbour recommenders in terms of rating prediction accuracy (UB1, UB2, IB1, and IB2) and remain very close to a newer baseline (UB3).

Based on these results, we can answer the second research question (how does this similarity perform in terms of prediction accuracy): both MAE and RMSE are lower for the best LCS-based recommender, and hence, **a similarity based on LCS allows to outperform classical nearest-neighbour algorithms in terms of prediction accuracy**.

4.2.3. Item Ranking Task

In Figures 1 and 2 we present results from the item ranking task in Lastfm and MovieLens, respectively. Here, in contrast with the previously presented task, the goal is to include as many relevant items in the top positions of the ranking presented to the user. Throughout the paper we will focus on rankings composed of 5 items, even though our experiments evidence similar behaviour for other cutoffs.

We observe in these figures that the LCS-based recommenders obtain better or as good results as the other nearest-neighbour recommendation algorithms. In the case of the Mahout framework, the improvements are quite significant; however,

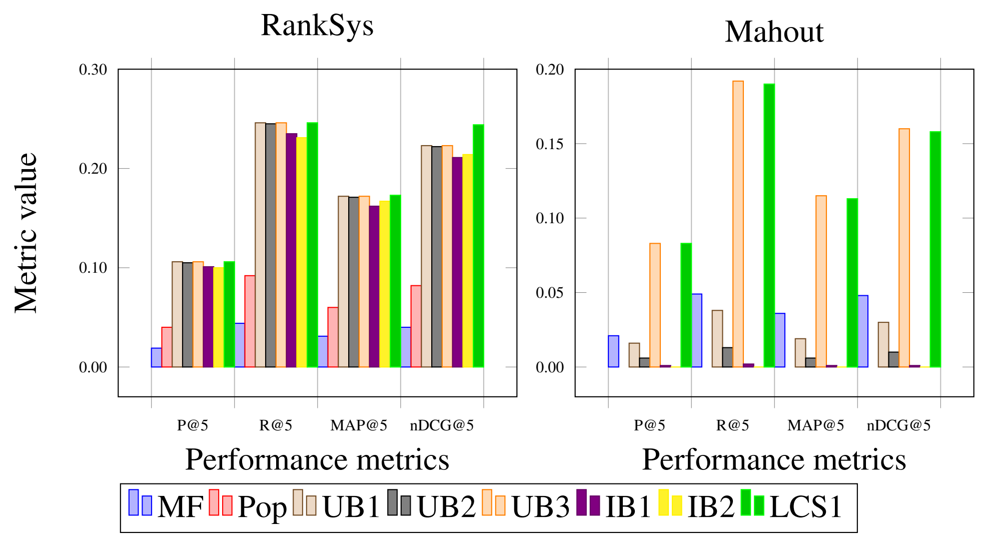

Figure 1: Performance results in the Lastfm dataset for RankSys (left) and Mahout (right) frameworks.

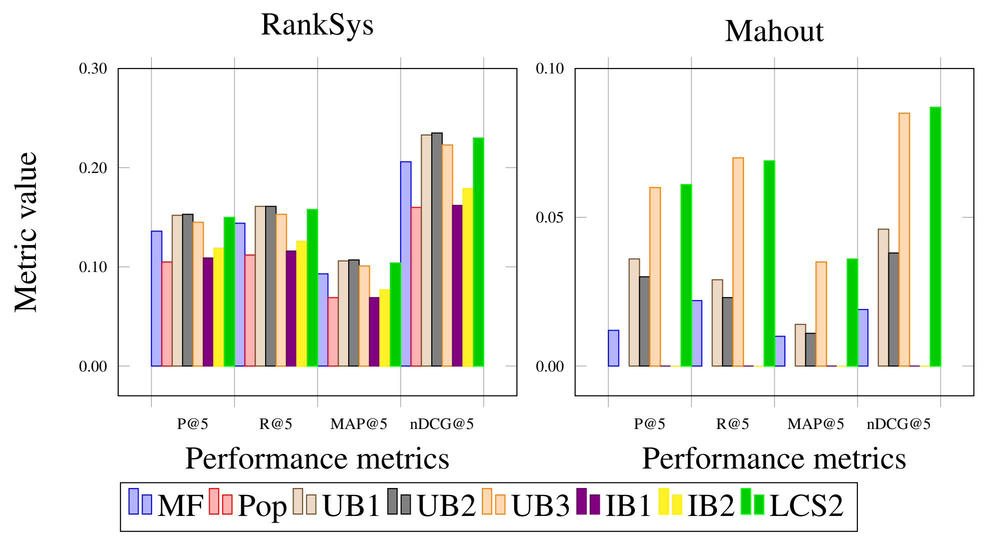

Figure 2: Performance results in the MovieLens dataset for RankSys (left) and Mahout (right) frameworks.

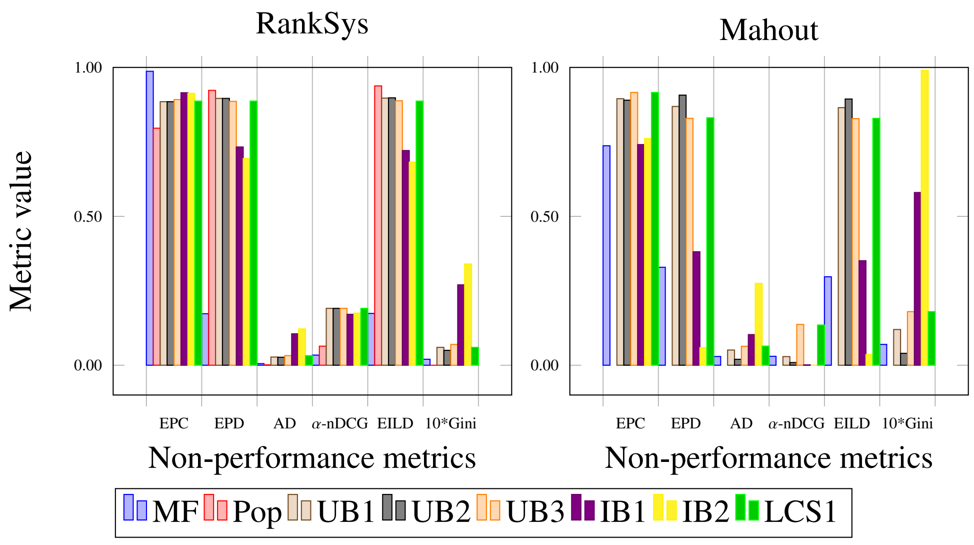

Figure 3: Novelty (EPC, EPD) and diversity (AD, α-nDCG, EILD, and Gini) results in the Lastfm dataset for RankSys (left) and Mahout (right) frameworks. Gini results are presented multiplied by a factor of 10 for better visualisation.

when the RankSys framework is used, some metrics obtain values very close to other baselines – for instance, MAP@5 in Lastfm. In general, the order of the baselines depends on the framework and dataset: MF is the best baseline in Lastfm using Mahout whereas UB1 outperforms the rest of the baselines in the other cases.

Hence, based on these results we can answer the third research question (how effective is it in terms of precision-based metrics) by summarising **the performance of similarities based on LCS as very competitive**, especially in some cases (depending on the dataset and specific implementation) where they outperform significantly other baselines.

Now, as alternative evaluation dimensions besides accuracy, in Figures 3 and 4 we show the results obtained for the non-performance metrics that measure the novelty and diversity of the recommendation lists. For the sake of space and clarity, we do not present results for all the metrics introduced in Section 4.1. Regarding the novelty metrics, we have discarded EFD because it is very similar to EPC, while EPC is bounded in $[0, 1]$. On the other hand, the diversity metrics tend to measure complementary concepts, except for ERR-IA and α-nDCG, which are very close to each other – actually, the former is an extension of the latter –; however, we decided to not present results of ERR-IA because they were not too discriminative (in the sense that the measurements were almost always too low).

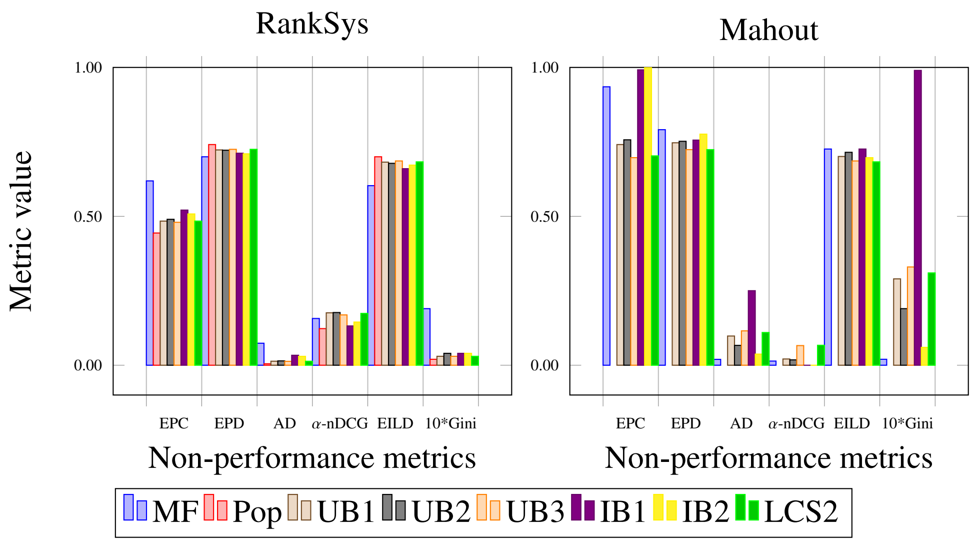

Figure 4: Novelty (EPC, EPD) and diversity (AD, α-nDCG, EILD, and Gini) results in the MovieLens dataset for RankSys (left) and Mahout (right) frameworks. Gini results are presented multiplied by a factor of 10 for better visualisation.

Figures 3 and 4 show that the LCS-based recommenders usually obtain results very similar to the other user-based nearest-neighbour recommender systems. This is interesting, especially when the Mahout framework is being used, since the performance in terms of precision in those cases is much higher, which results in a recommender with good performance and, at the same time, novel and diverse recommendations. This is not the case, for instance, of the item-based nearest-neighbour recommenders, that achieve high diversity values but their performance is suboptimal.

Therefore, we can finally answer the fourth research question (what is the impact on beyond-accuracy metrics when this similarity is used): **an LCS-based similarity metric does not change the novelty and diversity of user-based nearest-neighbour recommender systems**, which, together with the previous results, shows that this type of similarity produces recommendations with a good balance of accuracy and diversity/novelty.

4.3. Summary

The reported experiments provide empirical evidence of the usefulness of the proposed approach. The analysis of the results revealed that it is possible to use the Longest Common Subsequence (LCS) algorithm as a similarity metric for Collaborative Filtering. Moreover, its performance is comparable (and sometimes better) to that of classical algorithms in the field, both in terms of rating prediction accuracy and item ranking precision, without suffering from the well-known diversity-accuracy tradeoff.

The proposed LCS-based user similarity has three main components (transformation function, matching threshold, and normalisation) enabling a wide range of instantiations. Two of them have shown particularly good results: $f_i + 0 + \mathrm{sim}_2$ and $f_{ir} + 10 + \mathrm{sim}_2$. Even though the results for the transformation f_r are much lower than for the other transformations, it is remarkable that this approach – which only takes into account the patterns in rating values selected by the users when assessing their preferences – obtains competitive results when enough neighbours are considered. That is, whereas transformations f_i and f_{ir} produce high-quality neighbourhoods when these are small, f_r needs to consider a large amount of neighbours to obtain a comparable recommendation accuracy.

The other two components of the proposed similarity (matching threshold and normalisation) have different effects which depend on the size of the neighbourhoods, the nature and characteristics of the datasets, and how the recommendation algorithms are actually implemented. Whereas normalising the similarity metric usually improves the accuracy with respect to the algorithms using the not-normalised version, there are some situations where it has no effect. On the other hand, the sensitivity to the matching threshold seems to largely depend on how the ratings are distributed in the dataset, together with how much noise/uncertainty we are capable to handle in the system, since we would be matching users that have not expressed their preferences following the exact same behaviour.

4.4. Discussion

As already mentioned, the f_i transformation is actually equivalent to the item overlap between the users ($I(u, v) = I(u) \cap I(v)$), which is the basis for other user similarities like Jaccard or Cosine. In fact, due to the normalisation applied in sim_2, such instantiation is almost equivalent to the Jaccard similarity metric and ranking-equivalent to a binary Cosine, which would explain why UB1 in RankSys (that uses the Jaccard similarity) and the LCS-based recommender have a performance so close to each other. The explanation goes as follows: $\mathrm{sim}_2^{f_i,0}(u, v) = |I(u, v)|^2/(|I(u)| \cdot |I(v)|) \propto_u |I(u, v)|^2/|I(v)|$, where in the last step we make use of a ranking-equivalent transformation (by removing a term that only depends on user u); on the other hand, Jaccard similarity between users u and v is computed as $|I(u) \cap I(v)|/|I(u) \cup I(v)|$, this similarity is then used to rank the potential users as neighbours, so, let us suppose we are computing the neighbours of user u, then $|I(u) \cup I(v)| = |I(u)| + |I(v)| - |I(u) \cap I(v)| \propto_u |I(v)| - |I(u) \cap I(v)|$; at the same time, binary Cosine (where the interactions between users and items are either 1 or 0) is defined as follows:

$$\cos \mathrm{Bin}(u, v) = \frac{\sum_{i \in I(u,v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2 \sum_{i \in I(v)} r_{vi}^2}} = \frac{\sum_{i \in I(u,v)} 1}{\sqrt{\sum_{i \in I(u)} 1 \sum_{i \in I(v)} 1}} = \frac{|I(u, v)|}{\sqrt{|I(u)| \cdot |I(v)|}}$$

Therefore, by taking $\sqrt{\mathrm{sim}_2^{f_i,0}(u, v)}$ we find an equivalence with $\cos \mathrm{Bin}(u, v)$. Even though we have not used this similarity in our experiments, it evidences the generality of the proposed LCS-based similarity metric, and opens up further developments where other metrics (such as Pearson correlation) could be integrated under the same formulation.

Regarding the implementation of the recommendation techniques, we have obtained very different results in terms of absolute values depending on whether RankSys or Mahout frameworks were used. The main reason for this, as already observed in [29], is that it is very difficult to obtain completely objective evaluations unless the whole process (data splitting, recommendation training, candidate item selection for evaluation, evaluation metrics) is controlled. In our case, these frameworks were only used for the recommendation step, and hence, the rest of the evaluation pipeline remained controlled. However, we did observe differences in the algorithms (the most important one is probably that RankSys does not normalise the predicted score) so we decided to present the results from both frameworks, since we could not decide which baselines were closer to the state-of-the-art: Mahout implementations are more similar to the seminal algorithms, however RankSys produces recommendations of higher quality, following up-to-day optimisations [12]. Furthermore, considering our approach was implemented on top of the similarity interfaces defined in each framework, its results are only comparable in terms of the baselines produced by each framework, since it would not be fair to compare a recommendation method using our similarity inside the RankSys framework with the baselines from Mahout.

Finally, from the reported experiments we observe the baselines show a very different behaviour depending on the dataset: for instance, MF performs better in MovieLens than in Lastfm. This is also evidenced in the different instantiations of the LCS-based recommender: the optimal configurations for Lastfm and Movielens are different, including the neighbourhood size (see Table 11). We hypothesise this is due to the different nature the ratings in each dataset come from: whereas in MovieLens these ratings come from actual interactions between real users and the system, in Lastfm they have been generated by a monotonic transformation from implicit interactions between the users and the items in the system (number of listenings to each artist). Such transformation, even though it respects the order of the item preferences in a user basis, does not control for other factors typical in rating-based datasets, such as very skewed rating value histograms and the missing data not-at-random observation [25, 14]. Because of this, we aim to extend the LCS-based similarity measure to also deal with implicit feedback; for such extension we believe that, in contrast to the standard metrics, the basics of the proposed similarity would remain the same, we will only need to find proper transformation and matching functions so that the sequences derived from the implicit data can be processed by the LCS algorithm.

5. Related Work

Although LCS is a popular algorithm in science applications like molecular biology and file comparison [3], it is less known in recommendation, as not many researchers have integrated this algorithm in their frameworks – either as similarity metric, like we propose in this work, or in any other way. There are, however, some papers where it is used as a pattern finding algorithm. In [32], the authors have shown that this algorithm can be used in e-commerce applications by helping users choosing the items they need or like, achieving good results in precision, recall and F1 metrics. Furthermore, in [22] the authors show the utility of LCS in an online Web Usage Mining system, obtaining a best case accuracy of 73%. Nevertheless, using this algorithm as a similarity measure in rating-based systems is a novel approach – to the best of our knowledge – as other papers have used it mostly to obtain user patterns from activity logs.

Besides these examples, capturing as much as possible from the interaction patterns shown by the users has been investigated in the past. Recently, some authors have exploited Markov chains [34], also combined with matrix factorisation algorithms and similarity metrics [19] to analyse sequence patterns obtained from users. These approaches proved to be competitive, especially under sparse conditions.

6. Conclusions and Future Work

In this paper, we have proposed a new similarity metric between users of a Collaborative Filtering recommendation system that exploits the Longest Common Subsequence (LCS) algorithm, including three transformation functions to represent the users as sequences, a matching threshold to allow not-exact matchings in the user representations, and two normalisations to be applied to the output of the LCS algorithm. An analogy between an existing similarity metric and one of the possible instantiations of the proposed metric has been found: representing the users as items and using the presented normalisation is equivalent to the binary Cosine metric; this opens up the possibility of extending the current framework so that other, more complex metrics are integrated and generalised. These equivalences would then introduce new perspectives on the Collaborative Filtering problem, aiming to bring new insights and perhaps a better understanding of the recommendation problem, from which further researchers might design new and effective similarity metrics.

The empirical evaluation of our proposal shows competitive results in terms of precision and beyond-accuracy metrics (diversity and novelty); besides, significant improvements have been achieved in the rating prediction task. We have found that, depending on the actual implementation of the recommendation process (we have tested two different recommendation frameworks), an LCS-based similarity could outperform state-of-the-art baselines in two datasets of very different characteristics.

Several directions open up from this point to explore the potential of the proposed similarity metric. We plan to further study how to integrate implicit feedback into the computation of the LCS-based similarity. We will also consider alternative estimations where not only the rating information is used to transform the users into sequences, but content-based data – such as genres, categories, or other item features – are exploited to produce a hybrid recommendation system with minimal modifications of the framework presented herein, since we would only need to define proper transformation and matching functions able to deal with the new information.

We are particularly interested in generating user sequences considering the temporal dimension, so that an LCS-based method could exploit the order in which users consumed a set of items. Building upon the wide range of alternatives we have presented in this paper, we aim to incorporate temporal information in a similar way as it was done for time-aware recommender systems (where most of the formulations come from time-agnostic methods, like the ones introduced here), so that they could capture the user preferences with more fine-grained control. As it has been discussed before, building time-based sequences is not a trivial question, mainly because it is not easy to find datasets with meaningful timestamps, hence, this issue is not only a modelling problem, but also a problem of lack of valuable data to be exploited for recommendation.

Finally, it would be interesting to analyse how our approach compares against other baselines together with its sensitivity to well-known problems in the recommendation systems area, such as the presence of noise in user preferences [5, 33] and the new user and item problem, also known as *cold-start* [15].

Acknowledgements

This work was funded by Ministerio de Economía y Competitividad from the Spanish Government under Project TIN2016-80630-P. The authors also acknowledge the very helpful feedback from three anonymous reviewers.

References

[1] F. Aiolli, Efficient Top-n Recommendation for Very Large Scale Binary Rated Datasets, in: Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, ACM, New York, NY, USA, ISBN 978-1-4503-2409-0, 273–280, URL `http://doi.acm.org/10.1145/2507157.2507189`, 2013.

[2] M. Y. H. Al-Shamri, User profiling approaches for demographic recommender systems, Knowl.-Based Syst. 100 (2016) 175–187.

[3] 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, ISBN 3-540-60648-3, 361–398, URL `http://dl.acm.org/citation.cfm?id=267859.267867`, 1997.

[4] R. A. Baeza-Yates, B. A. Ribeiro-Neto, Modern Information Retrieval - the concepts and technology behind search, Second edition, Pearson Education Ltd., Harlow, England, ISBN 978-0-321-41691-9, URL `http://www.mir2ed.org/`, 2011.

[5] A. Bellogín, A. Said, A. P. de Vries, The Magic Barrier of Recommender Systems - No Magic, Just Ratings, in: UMAP, vol. 8538 of *Lecture Notes in Computer Science*, Springer, 25–36, 2014.

[6] 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.

[7] R. D. Burke, Hybrid Recommender Systems: Survey and Experiments, User Model. User-Adapt. Interact. 12 (4) (2002) 331–370.

[8] 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.

[9] P. Castells, N. J. Hurley, S. Vargas, Novelty and Diversity in Recommender Systems, in: Recommender Systems Handbook, Springer, 881–918, 2015.

[10] Ò. Celma, Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space, Springer, ISBN 978-3-642-13286-5, URL `http://dx.doi.org/10.1007/978-3-642-13287-2`, 2010.

[11] P. Cremonesi, F. Garzotto, S. Negro, A. Papadopoulos, R. Turrin, Comparative Evaluation of Recommender System Quality, in: CHI '11 Extended Abstracts on Human Factors in Computing Systems, CHI EA '11, ACM, New York, NY, USA, ISBN 978-1-4503-0268-5, 1927–1932, URL `http://doi.acm.org/10.1145/1979742.1979896`, 2011.

[12] P. Cremonesi, Y. Koren, R. Turrin, Performance of Recommender Algorithms on Top-n Recommendation Tasks, in: Proceedings of the Fourth ACM Conference on Recommender Systems, RecSys '10, ACM, New York, NY, USA, ISBN 978-1-60558-906-0, 39–46, URL `http://doi.acm.org/10.1145/1864708.1864721`, 2010.

[13] E. Q. da Silva, C. G. Camilo-Junior, L. M. L. Pascoal, T. C. Rosa, An evolutionary approach for combining results of recommender systems techniques based on collaborative filtering, Expert Syst. Appl. 53 (2016) 204–218.

[14] 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.

[15] I. Fernández-Tobías, M. Braunhofer, M. Elahi, F. Ricci, I. Cantador, Alleviating the new user problem in collaborative filtering by exploiting personality information, User Model. User-Adapt. Interact. 26 (2-3) (2016) 221–255.

[16] P. Gopalan, J. M. Hofman, D. M. Blei, Scalable Recommendation with Hierarchical Poisson Factorization, in: UAI, AUAI Press, 326–335, 2015.

[17] A. Gunawardana, G. Shani, Evaluating Recommender Systems, in: Recommender Systems Handbook, Springer, 265–308, 2015.

[18] I. Guy, Social Recommender Systems, in: Recommender Systems Handbook, Springer, 511–543, 2015.

[19] R. He, J. McAuley, Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation, in: ICDM, IEEE, 191–200, 2016.

[20] D. S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequences, Commun. ACM 18 (6) (1975) 341–343, URL `http://doi.acm.org/10.1145/360825.360861`.

[21] T. Hofmann, Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis, in: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, SIGIR '03, ACM, New York, NY, USA, ISBN 1-58113-646-3, 259–266, URL `http://doi.acm.org/10.1145/860435.860483`, 2003.

[22] M. Jalali, N. Mustapha, N. B. Sulaiman, A. Mamat, A Web Usage Mining Approach Based on LCS Algorithm in Online Predicting Recommendation Systems, in: Proceedings of the 2008 12th International Conference Information Visualisation, IV '08, IEEE Computer Society, Washington, DC, USA, ISBN 978-0-7695-3268-4, 302–307, URL `http://dx.doi.org/10.1109/IV.2008.40`, 2008.

[23] Y. Koren, R. M. Bell, Advances in Collaborative Filtering, in: Recommender Systems Handbook, Springer, 77–118, 2015.

[24] P. Lops, M. de Gemmis, G. Semeraro, Content-based Recommender Systems: State of the Art and Trends, in: Recommender Systems Handbook, Springer, 73–105, 2011.

[25] B. M. Marlin, R. S. Zemel, S. T. Roweis, M. Slaney, Collaborative Filtering and the Missing at Random Assumption, in: UAI, AUAI Press, 267–275, 2007.

[26] S. M. McNee, J. Riedl, J. A. Konstan, Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems, in: CHI '06 Extended Abstracts on Human Factors in Computing Systems, CHI EA

'06, ACM, New York, NY, USA, ISBN 1-59593-298-4, 1097–1101, URL http://doi.acm.org/10.1145/1125451.1125659, 2006.

[27] X. Ning, C. Desrosiers, G. Karypis, A Comprehensive Survey of Neighborhood-Based Recommendation Methods, in: Recommender Systems Handbook, Springer, 37–76, 2015.

[28] M. Richardson, E. Dominowska, R. Ragno, Predicting Clicks: Estimating the Click-through Rate for New Ads, in: Proceedings of the 16th International Conference on World Wide Web, WWW '07, ACM, New York, NY, USA, ISBN 978-1-59593-654-7, 521–530, URL http://doi.acm.org/10.1145/1242572.1242643, 2007.

[29] A. Said, A. Bellogín, Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks, in: Proceedings of the 8th ACM Conference on Recommender Systems, RecSys '14, ACM, New York, NY, USA, ISBN 978-1-4503-2668-1, 129–136, URL http://doi.acm.org/10.1145/2645710.2645746, 2014.

[30] D. Sánchez-Moreno, A. B. G. González, M. D. M. Vicente, V. F. L. Batista, M. N. M. García, A collaborative filtering method for music recommendation using playing coefficients for artists and users, Expert Syst. Appl. 66 (2016) 234–244.

[31] Y. Seo, Y. Kim, E. Lee, D. Baik, Personalized recommender system based on friendship strength in social network services, Expert Syst. Appl. 69 (2017) 135–148.

[32] 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, 223–226, 2011.

[33] R. Y. Toledo, J. Castro, L. Martínez-López, A fuzzy model for managing natural noise in recommender systems, Appl. Soft Comput. 40 (2016) 187–198.

[34] T. Tran, D. Phung, S. Venkatesh, Collaborative filtering via sparse Markov random fields, Information Sciences 369 (2016) 221 – 237, ISSN 0020-0255, URL http://www.sciencedirect.com/science/article/pii/S0020025516304455.

[35] S. Vargas, P. Castells, Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems, in: Proceedings of the Fifth ACM Conference on Recommender Systems, RecSys '11, ACM, New York, NY, USA, ISBN 978-1-4503-0683-6, 109–116, URL `http://doi.acm.org/10.1145/2043932.2043955`, 2011.

[36] S. Vargas, P. Castells, Improving Sales Diversity by Recommending Users to Items, in: Proceedings of the 8th ACM Conference on Recommender Systems, RecSys '14, ACM, New York, NY, USA, ISBN 978-1-4503-2668-1, 145–152, URL `http://doi.acm.org/10.1145/2645710.2645744`, 2014.