UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



TRABAJO FIN DE MÁSTER

# Exploiting subsequence matching in Recommender Systems

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones (i2-TIC)

Autor: Pablo Sánchez Pérez
Tutor: Alejandro Bellogín Kouki
Ponente: Fernando Díez Rubio

FECHA: Junio, 2017

# Resumen

Desde su surgimiento al inicio de la década de los 90, los sistemas de recomendación han experimentado un crecimiento exponencial empleándose en cada vez más aplicaciones debido a la utilidad que tienen para ayudar a los usuarios a elegir artículos en función de sus gustos y necesidades. Actualmente son indispensables en un gran número de empresas que ofrecen su servicio a través de Internet, el medio de intercambio de información más importante que existe. Por esta razón, la continua innovación en estos sistemas resulta imprescindible para poder efectuar recomendaciones que sean capaces de seguir sorprendiendo a los usuarios y mejorar las ya existentes.

En este Trabajo Fin de Máster hemos realizado un estudio e investigación acerca del estado actual de estos sistemas, prestando especial atención a los sistemas de filtrado colaborativo basados en vecinos y los basados en contenido. No obstante, debido a las desventajas que puede tener cada sistema por separado normalmente en aplicaciones reales se emplean combinaciones de varios sistemas, creando recomendadores híbridos. Como apoyo a este estudio, se propone como aspecto novedoso el uso del algoritmo de la subcadena común más larga (LCS) para ser utilizada como medida de similitud entre usuarios, introduciendo además una técnica general y transparente para generar secuencias haciendo uso tanto de información de contenido como de información colaborativa, pudiendo generar recomendadores híbridos de manera sencilla. Complementando a estos nuevos recomendadores, también detallamos otros parámetros auxiliares (confianza, preferencia, normalizaciones y distintas ordenaciones) que tienen como fin mejorar el rendimiento de estos sistemas basados en LCS.

Por otro lado, además de la definición de estos nuevos recomendadores, el trabajo se complementa con resultados experimentales haciendo uso de tres conjuntos de datos conocidos en el área: Movielens, Lastfm y MovieTweetings. Cada uno de ellos estará orientado a explotar un aspecto específico de la generación de secuencias. Los resultados han sido obtenidos haciendo uso de métricas de ranking (Precisión, Recall, MAP o nDCG) y de novedad y diversidad ($\alpha$-nDCG, EPC, EPD, Aggregate diversity, EILD y Gini). Los resultados han tenido como fin comparar el rendimiento de los recomendadores basados en la subsecuencia común más larga frente a otros recomendadores conocidos en el área.

Como resumen, se ha observado que los recomendadores propuestos resultan altamente competitivos en las pruebas realizadas siendo incluso mejores en algunas ocasiones a otros recomendadores conocidos en el área, tanto en términos de métricas de ranking como de novedad y diversidad. No obstante, también se ha observado que, en algunos casos, el uso de recomendadores híbridos basados en la subsecuencia común más larga obtiene unos resultados peores que otras versiones puramente colaborativas. En cualquier caso, consideramos que es una propuesta con potencial para seguir siendo investigada.

**Palabras Clave**

Sistemas de recomendación, subsecuencia común más larga, novedad y diversidad, métricas de similitud.

# Abstract

Since their inception in the early 1990s, recommender systems have experienced exponential growth as they are being used in a large number of applications because of their usefulness in helping users choose items based on their tastes and needs. Nowadays, they are indispensable in many companies that offer their service through the Internet, the most important method for information exchange. For this reason, continuous innovation in these systems is essential to make recommendations that are able to continue surprising users, while improving the existing ones.

In this Master's Thesis, we have studied and researched on the current state of these systems, paying special attention to collaborative filtering based on neighborhood and content-based algorithms. However, due to the disadvantages that each system may have separately, combinations of these systems are often used in real applications, creating hybrid recommenders. To support this study, we propose the use of the longest common subsequence (LCS) algorithm as a novel aspect to be used as a similarity metric between users, also introducing a general and transparent technique to generate sequences using both content and collaborative information, allowing us to generate hybrid recommenders in a simple way. Complementing these new recommendations, we also detail other auxiliary parameters (confidence, preference, normalization functions, and different orderings), whose main goal is to improve the performance of these LCS-based systems.

On the other hand, in addition to the definition of these new recommenders, the study is complemented with experimental results using three well-known datasets in the area: Movielens, Lastfm and MovieTweetings. Each one of them will be oriented to exploit a specific aspect of the sequence generation process. The results have been obtained by using ranking metrics (Precision, Recall, MAP, or nDCG) and novelty and diversity metrics ($\alpha$-nDCG, EPC, EPD, Aggregate diversity, EILD, and Gini). With these experiments, we aimed at comparing the performance of recommenders based on the longest common subsequence against other well-known recommenders in the area.

As a summary, we have observed in the experiments performed that the proposed recommenders are highly competitive, and sometimes they are even better than other recommenders known in the area, both in terms of ranking quality metrics, and novelty and diversity dimensions. However, we have also observed that, in some cases, the use of hybrid recommenders based on the longest common subsequence results in worse performance than other purely collaborative versions. In any case, we believe this is a proposal with enough potential to be worthy of further investigation.

**Keywords**

Recommender systems, Longest common subsequence, novelty and diversity, similarity metrics.

# Acknowledgments

I would like to express my gratitude to my director Dr. Alejandro Bellogín for the support provided throughout the year. His guidance has been indispensable to be able to accomplish this work successfully.

My sincere thanks are also for Dr. Pablo Castells, who allow me to join his team and use his laboratory for the research project. Without these facilities it would have been impossible to make this work.

I thank my master's mates for all the time spent working together finalizing lab assessments, especially Alejandro Catalina. Without his help, it would have been even harder. I am also grateful to other colleagues who although I have not shared classes with them this year, I have also shared very good moments, especially with Guillermo Sarasa and Sergio Sanz.

Last but not the least, I would like to thank my family, especially my parents and my brother, for supporting me throughout writing this thesis and in my life.

# Contents

# List of Figures

# List of Tables

# Acronyms

- **AD**: Aggregate Diversity. Diversity metric.
- **ALS**: Alternating Least Squares. Learning algorithm for MF.
- **EILD**: Expected Intra-List Diversity. Diversity metric.
- **EPC**: Expected Popularity Complement. Novelty metric.
- **EPD**: Expected Profile Distance. Novelty metric.
- **IB**: Item-Based Collaborative Filtering. $k$-NN strategy where similarities are computed between items.
- **IDF**: Inverse Document Frequency.
- **$k$-NN**: $k$ Nearest Neighbors.
- **LCS**: Longest Common Subsequence.
- **LDA**: Latent Dirichlet Allocation.
- **MAE**: Mean Absolute Error. Error metric.
- **MAP**: Mean Average Precision. Ranking evaluation metric.
- **MF**: Matrix factorization.
- **nDCG**: Normalized Discounted Cumulative Gain. Ranking evaluation metric.
- **pLSA**: Probabilistic Latent Semantic Analysis. Matrix factorization technique.
- **RMSE**: Root Mean Square Error. Error metric.
- **RS**: Recommender System.
- **SGD**: Stochastic Gradient Descent. Learning algorithm for MF.
- **TF**: Term Frequency.
- **UB**: User-Based Collaborative Filtering. $k$-NN strategy where similarities are computed between users.
- **VSM**: Vector Space Model. Spatial representation of text documents.

# Chapter 1

# Introduction

## 1.1 Motivation

Recommender Systems (RS) are software tools that allow users finding the items and information they need in a simple and direct way. These items are related to the specific domain of the recommender system, although movies, books, and music are some of the most studied domains in the scientific community. In order to predict the most interesting items for each user in the system, these methods analyze the tastes and interests of the users to make personalized recommendations [36].

Although the rise of the Internet dates back to the early 1970s, the research in these systems has taken place especially in the last 20 years, due to the global spread of information and communication technologies. The antecedents of these systems can be found in the early 1990s, in the Tapestry [20] and Grouplens [35] projects, co-occurring with the rise of the Internet. However, they are especially relevant today as they have now become essential to filter the large amount of data available in the cloud. A large number of important companies offering online services make use of recommendation algorithms to expand their economic activity and improve the user experience. Some examples are Amazon (online store), Youtube (videos) or Netflix (streaming audiovisual content). This last company became very popular in 2006 for a three-year contest with a prize of 1 million dollars to the research group who managed to improve its prediction algorithm by 10% [31]. The team "BellKor's Pragmatic Chaos" ended up winning this contest and putting these technologies in the spotlight.

User recommendations can be obtained by many different ways, depending on how the system works with the data. Traditional approaches like content-based and collaborative-filtering recommenders were the first systems being implemented and deployed in industry, and they still remain the core of many applications. Nevertheless, other recommendation techniques like knowledge-based (which requires specific knowledge about the items and users preferences) or demographic (that makes recommendations analyzing features of the population around the user) have emerged and they are worth of further research [36], even though they are out of the scope of this work. Herein we shall focus on $k$-NN algorithms, where the users closer to the target user – in terms of a specific similarity metric – are used in the recommendation process. This type of technique is very intuitive and simple to implement, which allows for richer explanations and justifications of the recommendations, however it typically suffers from limited coverage and lower accuracy. Nonetheless, it is worth noting that every technique has its own advantages and drawbacks, which motivates the definition of hybrid systems combining the recommendations coming from any number of techniques in order to achieve a better performance than each system individually [10].

In this scenario, some researchers are using the so-called *context* information to train the recommendation algorithms to adapt better to the user needs. In the end, making good recommendations entail the success of the companies, as regular customers will keep on trusting them and new customers looking for new products may emerge. Among the different possibilities to gather contextual information from, temporal data is one of the most interesting contexts to be integrated into the recommendation approaches, due to its facility to be captured (normally, when a user purchases or consumes an item, a timestamp is created, so it is not difficult to find datasets with this information) and it usually discriminates better than other dimensions [1]. Nonetheless, to be able to work with temporal data, the classical recommendation task needs to be redefined, since the users are now represented as a sequence of actions, and the objective is, hence, to predict which action s/he will do next, requiring changes in the algorithms but also in the way the evaluation is performed (e.g., training/test splits should be temporal). Markov Chain methods (and derivations) are widely used to model these stochastic transitions [22], whereas recommenders based on neighborhood have been mostly modified in ad-hoc fashions to integrate temporal information, but no formal approaches for this kind of methods have been proposed so far [11].

Although in the Netflix prize the objective was to achieve a lower error between the predicted ratings and the real ones (more specifically, the goal was reducing the RMSE by a 10%), this evaluation approach has been displaced by the use of ranking evaluation metrics, some of them adapted from the Information Retrieval (IR) field, where these metrics seek to measure the relevance of a ranking of articles provided for the user. However, in the last years, recommender systems evaluation has acknowledged complementary dimensions such as coverage or serendipity, but the two most relevant ones are, at the moment, novelty and diversity [13]. These new dimensions aim to measure other aspects besides accuracy of the recommendations received by the user. A paradigmatic example where these metrics evidence its potential is when a popularity recommender (that recommends the most popular items in the system) is evaluated: despite its simplicity, this (non-personalized) method usually obtains good results in terms of ranking quality metrics such as Precision or nDCG; however, since the recommended items are the most popular in the system, they are well-known to the user (lack of novelty) and the same for every user (low diversity). In this context, the main concern in the community is to obtain a recommender with a good balance between accuracy, novelty, and diversity, since it is generally accepted that outperforming every other recommender in all dimensions is a very difficult problem to solve [21].

Taking all of the above as a starting point, in this Master's Thesis we propose a new similarity metric to be integrated in $k$-NN collaborative-filtering recommenders. This metric exploits the item sequences generated by the users when they interact with the recommender system by means of the Longest Common Subsequence (LCS) algorithm [3]. As this algorithm cannot be applied directly to the recommendation context, a general method to transform users into sequences is formulated. Its generality allows us to work with both collaborative-filtering and content-based information, creating a new hybrid recommender in a simple and direct way with different options to order the items in the sequences. Besides, we will also define some configurable parameters (*confidence filter*, *preference filter*, and *normalization functions*) that can be used to reduce the computational time and increase the recommender performance. To evaluate this approach, we make use of three datasets (MovielensHetRec, Lastfm, and MovieTweetings) that contain different information (pure collaborative-filtering, content-based, and temporal data) in order to see if there are any advantages over the traditional algorithms. We report the results of our proposals

on these real-world datasets (using offline evaluation) comparing them against other state-of-the-art recommenders in order to check the validity of our approach. The results are presented in terms of ranking quality (Precision, Recall, MAP, and nDCG) and novelty and diversity (EPC, EPD, $\alpha$-nDCG, Aggregate diversity, and Gini) evaluation metrics.

In summary, the research proposed in this Master's Thesis aims to answer the following questions:

- Can state-of-the-art similarities be extended to take advantage of user sequences generated from user interactions with the system?
- Is it possible to create a recommender system taking advantage of these sequences? If so, which configurations and parameters work best in this case? In other words, which ones are better aligned to the recommendation problem?
- How can we adapt sequence-aware recommenders to work with temporal information? What advantages and disadvantages can be found when introducing this new context in recommendation?
- What is the performance of these new approaches in terms of both ranking quality and novelty and diversity metrics?

## 1.2 Goals

Taking into account the questions formulated above, the main objective of this work is to investigate if it is possible to define new similarities exploiting the subsequence matching problem between users. This main objective is divided in the following goals:

- Investigate different ways to build user sequences based on preference data, in particular, using the Longest Common Subsequence algorithm.
- Reformulate the traditional similarities to take advantage of the sequences of items consumed by the users.
- Integrate content-based features or any other additional information into the user sequence generation process.
- Test these new approaches and analyze them in terms of ranking quality and novelty and diversity metrics.
- Obtain conclusions about the algorithms based on LCS and how different configurations and parameters affect their performance, using empirical results obtained with real-world datasets.

## 1.3 Document structure

The structure of this Master's Thesis is as follows:

- In the second chapter (State of the Art) we present some important concepts about recommender systems and the main references and studies are introduced. We first categorize the different types of recommenders along with the definition of the recommendation problem, then we describe the most common metrics used to evaluate the performance of the recommender systems.

- In the third chapter (LCS as a similarity metric) we formally define the method proposed to obtain user sequences from the data. Some other additional parameters whose objective is to improve the quality of the recommendations are also described. These explanations are supported by some examples in order to help understanding the approach.
- In the fourth chapter (Experiments) we describe the evaluation methodology we followed in order to test the recommendation algorithms. Furthermore, empirical results of both ranking quality and novelty and diversity metrics of different recommenders based on LCS and other state-of-the-art algorithms are also presented and discussed.
- In the last chapter (Conclusions) we summarize the main contributions of this research, as well as different ideas for future research.
- Finally, in the appendices we present specific details about the implementation used in our work (Appendix A) and additional results regarding another content-based feature (Appendix B) and another dataset (Lastfm, Appendix C) which, for the sake of space and length of Chapter 4, were not included in that chapter.

# Chapter 2

# State of the art

## 2.1 Sequence matching

The Longest Common Subsequence (LCS) problem 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 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. This problem arises in a number of applications, from text editing to molecular sequence comparisons, and has been extensively studied [3]. The standard dynamic programming solution to compute the LCS can be seen in Algorithm 1. It has a complexity of $O(mn)$ for both time and space, where $n$ and $m$ are the length of the two input sequences. If only the length of the LCS is needed, then the algorithm can be adapted to use only linear space. The reason for this is that the computation of each row of matrix $L$ only needs the preceding row (this implementation is presented in Algorithm 2). However, if we want to retrieve the full LCS, the matrix computation is required. In Table 2.1 the reader can see an example of a computation of the LCS for the strings "BACBAD" and "ABAZDC", whose LCS is "ABAD" with a length of 4.

**Algorithm 1** Longest Common Subsequence

```
1:  procedure LCS(x, y)                                   ▷ The LCS of x and y
2:      L[0 ··· m, 0 ··· n] ← 0
3:      for i ← 1, m do
4:          for j ← 1, n do
5:              if x_i = y_j then
6:                  L[i, j] ← L[i − 1, j − 1] + 1          ▷ There is a matching
7:              else
8:                  L[i, j] ← 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 ... x_i and
    y_1 ... y_j
13: end procedure
```

**Algorithm 2** Longest Common Subsequence. Optimizing space.

```
1: procedure LCS_OS(x, y)          ▷ The LCS of x and y computed using linear space
2:     prev[0 ··· n] ← 0
3:     cur[0 ··· n] ← 0
4:     for i ← 1, m do
5:         for j ← 1, n do
6:             if x_i = y_j then
7:                 cur[j] ← prev[j − 1] + 1                    ▷ There is a matching
8:             else
9:                 cur[j] ← max(prev[j], cur[j − 1])
10:            end if
11:        end for
12:        for j ← 1, n do
13:            prev[j] ← cur[j]
14:        end for
15:    end for
16:    return cur[n]▷ cur[n] contains the length of an LCS between x_1 ... x_i and y_1 ... y_j
17: end procedure
```

**Table 2.1**: LCS Example Matrix. The circled number represents the length of the LCS found by the algorithm.

| | 0 | B | A | C | B | A | D |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| B | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| A | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| Z | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| D | 0 | 1 | 2 | 2 | 2 | 3 | 4 |
| C | 0 | 1 | 2 | 3 | 3 | 3 | (4) |

## 2.2 Recommender Systems

Recommender Systems (RS) are tools whose purpose is to make item recommendations to users. Depending on how these recommendations are obtained we distinguish the following systems [36, 9]:

- Content-based (CB): suggestions are done by analyzing the features of the items the user liked in the past. For example, in the movie domain, if a person likes movies of an specific genre or director, the system will recommend more movies of the same genres/directors.

- Collaborative-filtering (CF): recommendations are made by exploiting the users' preferences in order to find other users that have similar tastes. CF techniques can be divided in two groups: $k$-NN and model-based recommenders.

- Demographic: they assume that users with the same demographic profile tend to have common interests. For example, it is not strange to see that people from a given country will have more interests on news about that country.

- Knowledge-based: they normally do inference between user preferences or needs in order to make recommendations. They recommend items to the users trying to solve a specific problem.
- Community-based: they analyze the preferences of friends/relatives of the user in order to make suggestions. They are widely used in some social networks.
- Hybrids: all the systems that have been defined before have advantages and disadvantages (see Table 2.2). To avoid these individual drawbacks, in real applications combinations of them are implemented in order to achieve a global improvement.

Main advantages and drawbacks of each of the aforementioned systems are shown below and summarized in Table 2.2. According to [10] and [5] the most common problems of each system are:

- Overspecialization (OP): recommendations are very similar to other items the user has consumed before. Suggested items are usually already known by the user.
- Cold start (CSP): problem with new users/items. It is difficult to recommend items to new users if there is no previous activity from them, as they are difficult to categorize. In the case of new items, if they have not been rated by many users, recommending them is not a trivial task.
- Gray sheep (GSP): some users have specific or unusual preferences, so finding neighbors in order to make useful recommendations becomes a difficult task.
- Knowledge-engineering (KEP): some special knowledge either by the product or by the user is required.
- Demographic information (DIP): it is necessary to process demographic data where we must take into consideration legal aspects.
- Large-data (LDP): in order to make good recommendations, a large amount of data is required.

And the general advantages:

- Not data-domain knowledge (DDA): it is not necessary a specific knowledge about the data domain as it is transparent for the recommender.
- Improvement over time (ITA): this normally depends on the algorithm that is being used in the system. It will be indicated if this feature is available for the most extended algorithms.
- Implicit-data (IDA): it can make recommendations with data that is not directly indicated by the user but obtained analyzing her/his activity (e.g., server logs or the number of listenings of a song).
- User-sensitive (USA): it is sensitive to the changes of the user's taste over time.
- More features (FA): it can include other features of the products like distribution dates, product value, availability, etc.

Among all, content-based and collaborative-filtering are two of the most widely deployed recommendation approaches. In this work we will make special emphasis on these two types of recommendation systems, showing a more exhaustive description in the next subsections.

**Table 2.2**: Advantages and disadvantages of recommender systems.

| RS | Advantages | Disadvantages |
|---|---|---|
| Content-based | ITA, IDA | OP, LSP, LDP |
| Collaborative-filtering | ITA, IDA, DDA | CSP, LDP, GSP |
| Knowledge-based | USA, FA | KEP |
| Demographic | DDA | DIP, LDP, GSD |
| Community-based | DDA, ITA | CSP, GSP |

### 2.2.1 Problem definition and notation

In order to help the reader throughout this document, we introduce now the following notation for the recommendation task. The set of users in the system will be denoted as $\mathcal{U}$ and the set of items as $\mathcal{I}$. The set of ratings will be $\mathcal{R}$ and the values of possible ratings as $\mathcal{F}$ (normally $\mathcal{F}$=[1,5] or $\mathcal{F}$={I like/I dislike}). When $\mathcal{F}$ is known, we say that the data is explicit and implicit otherwise (e.g., server logs). Normally it is easier to work with explicit data as the provided ratings have a valuable interpretation of the users' interests/preferences. We will also consider that a user $u \in \mathcal{U}$ cannot make more than one rating to a particular item $i \in \mathcal{I}$. The subset of items that has been rated by a user will be denoted as $\mathcal{I}_u$ and the users that have rated an item will be $\mathcal{U}_i$. Furthermore, the items rated by two users $\mathcal{I}_u \cap \mathcal{I}_v$ will be abbreviated as $\mathcal{I}_{uv}$.

In [1] it is presented the fundamental recommendation problem by making use of a function $g(u, i)$ that indicates the usefulness of an item to a user. This utility function is of the form $g : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{D}_g$ and is represented in this way:

$$\forall u \in \mathcal{U}, i'_u = \arg\max_{i \in \mathcal{I}} g(u, i) \tag{2.1}$$

Thus, for each user $u \in \mathcal{U}$, we want to choose items $i \in \mathcal{I}$ maximizing function $g$. However, the target domain $\mathcal{D}_g$ can be defined in different ways depending on the problem we are solving. The most well-known tasks in recommendation are rating prediction and top-N recommendation [19]. Rating prediction aims to determine what rating a user will give to an item that has not been rated by her/him. In this case $\mathcal{D}_g$ would correspond to $\mathcal{F}$, for instance, the interval $[1, 5]$. Nevertheless, sometimes we only know the items that the user has consumed (without using ratings) or we do not want to return a predicted rating but a ranking list, or an utility value representing the user interests. In this case we consider the top-N recommendation task in which we recommend a list of items hypothetically relevant to the user, changing the domain $\mathcal{D}$ to another one that may be less bounded (up to the number of items in the system), or bounded to different intervals depending on the utility function used. There are several ways to evaluate the recommendation based on the aspect that we want to analyze; the most important approaches and metrics are presented in Section 2.3.

### 2.2.2 Content-based recommenders

Content-based recommender systems (CB) analyze the items the user liked in the past and recommend items having similar features. The main process of making recommendations using a content-based approach consists in matching users preferences and interests obtained in the users' record, with the attributes of the items [29]. Since CB systems recommend items analyzing the user profile, the utility function $g$ can be defined as:

$$g(u, i) = sim(UserProfile(u), Content(i)) \tag{2.2}$$

where $Content(i)$ will be the item profile (attributes that characterize item $i$). As CB recommenders tend to use articles represented by text, the content is normally represented by keywords using simple retrieval models like the Vector Space Model (VSM). A VSM is a spatial representation of text documents in which each document $d \in D$ is represented by an $n$-dimensional vector, where $n$ will be the size of the vocabulary of keywords (usually the number of keywords should not be too broad in order to create manageable vectors). Although VSM is normally used with text documents, it can also be used in the context of movies or songs, the keywords could be the genres or specific tags. In this approach, every item is represented as a vector of term weights. Let us denote $T = \{t_1, t_2, \cdots, t_n\}$ as the terms in the system. Each item $j$ can be represented as $\vec{i_j} = \{w_{1j}, w_{2j}, \cdots, w_{nj}\}$, in which each weight $w_{nj}$ represents the degree of association between the item $j$ and the term/keyword $n$ [17]. Recalling Equation 2.2, $UserProfile(u)$ can be represented in the same way. In this case, the keywords or terms belonging to the user can be obtained by adding the terms of all the articles this user has consumed.

However, this definition is incomplete, as we still need a method to compute the weights of the terms and another one to measure the vector similarity. The most extended mechanism for term weighting is TF-IDF (Term Frequency-Inverse Document Frequency) which assumes that unusual terms may be more relevant than usual ones, that terms that appear multiple times in a document are more important than others appearing less times, and that short documents tend to be better for the users than longer ones [17]. The TF-IDF function will be defined as:

$$TF - IDF(t_k, d_j) = TF(t_k, d_j) \cdot IDF(t_k) \tag{2.3}$$

where

$$TF(t_k, d_j) = \frac{f_{k,j}}{\max_z f_{z,j}} \tag{2.4}$$

$$IDF(t_k) = \log \frac{|D|}{n_k} \tag{2.5}$$

In these equations, $TF(t_k, d_j)$ takes into account the frequency of term $k$ by dividing such frequency by the maximum of the frequencies $f_{z,j}$ of all keywords $k_z$ that appear in document $d_j$. The other part of Equation 2.3, $IDF(t_k)$, will penalize keywords that appear in many documents as they do not help in distinguishing between useful and non-useful items. In this case, $|D|$ denotes the total number of documents of the system and $n_k$ is the number of documents where the term $k$ occurs at least once.

In order to bound the weights between [0,1], the TF-IDF function is usually normalized as follows:

$$w_{ki} = \frac{TF - IDF(t_k, d_i)}{\sqrt{\sum_{s=1}^{|T|} TF - IDF(t_s, d_i)^2}} \tag{2.6}$$

Nevertheless, as mentioned before, a similarity measure is needed to find the proximity between two vectors. Cosine similarity is the most widely used similarity measure. We will consider $\vec{w_u}$ the user vector and $\vec{w_i}$ the item vector:

$$cos(\vec{w_u}, \vec{w_i}) = \frac{\sum_{j=1}^{k} w_{uj} w_{ij}}{\sqrt{\sum_{j=1}^{k} w_{uj}^2} \cdot \sqrt{\sum_{j=1}^{k} w_{ij}^2}} \tag{2.7}$$

which represents the utility function defined above. For example, in the context of books, if a user consumes many horror books, the content-based recommendation system will make recommendations of this style, matching the description of such books with important

keywords like "ghosts", "zombies", "curses", and so on. The cosine similarity will be higher with books having such keywords than other ones related to romantic stories.

Moreover, there are other techniques for CB recommendation. One of the most important techniques is Bayesian classifiers. These approaches estimate the posterior probability $P(c \mid d)$ of a document belonging to a specific class $c$ based on the prior probability of the class $P(c)$, the probability of observing the document in class $c$ (i.e., $P(d \mid c)$) and the probability of observing the document $d$ denoted as $P(d)$ [17]. Normally, the classes have two possible values, the user likes the document or the user dislikes it. Applying the Bayes theorem:

$$P(c \mid d) = \frac{P(c)P(d \mid c)}{P(d)} \tag{2.8}$$

The estimation of $P(d \mid c)$ is complicated, thus it is common to use the naïve Bayes classifier, as shown in [30] for book recommendation and in [33] for classifying unrated web pages. With the naïve Bayes classifier, the document is replaced by a vector of keywords over the system vocabulary, $T$ according to our notation. Each component of the vector indicates whether that keyword appeared in the document or not. If we work with binary values, we are using a multivariate Bernoulli approach and if we count how many times the word appeared in the document, we are making use of multinomial naïve Bayes [17]. This second approach can be represented as:

$$P(c_j \mid \vec{i_d}) = P(c_j) \prod_{t_k \in \vec{i_d}} P(t_k \mid c_j)^{N(i_d, t_k)} \tag{2.9}$$

in which $N(i_d, t_k)$ are the number of times that the word $t_k$ appeared in the document $d$ ($\vec{i_d}$). This kind of models are very efficient and easy to implement, although the may obtain worse results than other learning methods as Support Vector Machines (SVM).

### 2.2.3 Nearest neighbors collaborative-filtering recommenders

Unlike CB systems, nearest neighbors collaborative-filtering recommenders (also known as $k$-NN recommenders) estimate the function $g$ by taking into account the opinions (ratings) assigned by similar users of $u$ to item $i$. This kind of recommenders take advantage of the intuitive idea that users who are similar tend to prefer similar items and that similar items are preferred by similar users. They have some important advantages [19]. They are intuitive and simple, as it is an approach easy to understand and implement. Normally the only important parameter is $k$, the number of neighbors to use in the recommendation step. They are not seriously affected by the constant insertion of new users or items and, in addition, their training phase (neighbors computation) is usually less computationally expensive than the other family of collaborative-filtering systems (model-based).

Neighborhood CF recommenders can be divided in two categories depending on how the similarities are computed. If the similarities are between users, the method is called user-based (UB) and if the similarities are between items it is called item-based (IB). We will replace, for the moment, the function $g(u, i)$ with $\hat{r}_{ui}$ (the predicted rating of a user $u$ to an item $i$) as done in [1]. The value of an unknown rating of user $u$ to item $i$ can be represented as an aggregation of the ratings of the top-$k$ neighbors. The most simple way to do so is to compute the mean of the ratings of all neighbors of $u$, however, the most extended approach to predict a rating in the user $k$-NN approach is a weighted sum:

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} r_{vi} w_{uv}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.10}$$

where $w_{uv}$ is the weight (or similarity) between users $u$ and $v$ and $\mathcal{N}_i(u)$ represents user's $u$ neighbors that have rated item $i$. In the same way, we can define the item-based approach prediction as:

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathcal{N}_u(i)} r_{uj} w_{ij}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.11}$$

If we are focusing on predicting a rating to an item, we must take into account that each user has its own personal scale. A rating of 4 over 5 can be the standard rating of a user and an eccentric rating for another one, so it can be useful to normalize the ratings. Two of the most popular normalization schemes are mean-centering and Z-score [35, 23]. The formula of the mean centering normalization (for a user-based approach) is:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_i(u)} (r_{vi} - \bar{r}_v) w_{uv}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.12}$$

And of the Z-score normalization:

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(u)} (r_{vi} - \bar{r}_v) w_{uv} / \sigma_v}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.13}$$

where $\bar{r}_u$ and $\sigma_u$ correspond, respectively, to the user's $u$ average rating and standard deviation, calculated from the observed interactions with the system.

Both formulas can be used for item-based approaches as follows. The mean centering normalization:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} (r_{uj} - \bar{r}_j) w_{ij}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.14}$$

and the Z-score normalization:

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum_{j \in \mathcal{N}_u(i)} (r_{uj} - \bar{r}_j) w_{ij} / \sigma_j}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.15}$$

The Equations 2.10 and 2.11 perform well when predicting the rating that a user will give to an item. However, this it not the case when the system is evaluated with ranking metrics such as Precision, Recall, MRR or nDCG. When evaluating with these metrics, Equation 2.16 performs better when it is not normalized [16, 2]:

$$\hat{r}_{ui} = \sum_{v \in \mathcal{N}_i(u)} r_{vi} w_{uv} \tag{2.16}$$

In this equation, $\hat{r}_{ui}$ represents the score to be used when ranking the item, not the predicted rating. Hence, using this approach, error metrics like MAE or RMSE cannot be calculated (see Section 2.3).

To compute similarities between users, there are three popular approaches: the cosine similarity, Pearson Correlation, and Jaccard index:

$$\cos(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2 \sum_{j \in \mathcal{I}_v} r_{vj}^2}} \tag{2.17}$$

$$\mathrm{PC}(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \overline{r}_u)(r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \overline{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \overline{r}_v)^2}} \tag{2.18}$$

$$\text{Jaccard}(u,v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u \cup \mathcal{I}_v|} \tag{2.19}$$

The reader may notice that the first two metrics are bounded between [-1,1]. However, if the ratings are always positive (i.e., $F \in [1,5]$), cosine similarity is actually bounded between $[0,1]$. The three equations shown above are defined this way for a user-based $k$-NN approach. Equivalent formulations can be derived to compute similarities between items:

$$\cos(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2 \sum_{u \in \mathcal{U}_j} r_{uj}^2}} \tag{2.20}$$

$$\text{PC}(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \overline{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \overline{r}_j)^2}} \tag{2.21}$$

$$\text{Jaccard}(i,j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{|\mathcal{U}_i \cup \mathcal{U}_j|} \tag{2.22}$$

There are more similarities metrics that are used in recommendation, like Adjusted Cosine similarity (AC) [38] or Mean Squared difference (MSD) [39], however the ones shown here are the most widely and typically used.

### 2.2.4 Matrix factorization models

Matrix factorization (MF) techniques, also known as model-based techniques, were the first choice for implementing collaborative-filtering recommenders and due to their accuracy they are the preferred technique for working with the Netflix data (one of the largest available datasets in the community) [27]. These models try to explain the ratings by characterizing both users and items on a number of factors (denoted as $k$) obtained from the rating matrix. Hence, they map the users and items to a joint latent factor space of dimensionality $k$, so user-item interactions are modeled as dot products in that space [28]. Thus, each user $u \in \mathcal{U}$ is associated with a vector $p_u \in \mathbb{R}^k$ and each item $i \in \mathcal{I}$ with a vector $q_i \in \mathbb{R}^k$. Then, the inner product of $q_i^T \cdot p_u$ approximates the rating of user $u$ to item $i$, that is:

$$\hat{r}_{ui} = q_i^T \cdot p_u \tag{2.23}$$

Once this formulation is derived, the important step is learning the factor vectors ($q_i$ and $p_u$). To do so, it is necessary to minimize the regularized squared error:

$$\min_{q*,p*} \sum_{(u,i) \in \mathcal{R}} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{2.24}$$

In this case, $\mathcal{R}$ is equivalent to the set of pairs $(u,i)$ whose rating is known. The $\lambda$ variable is a constant whose purpose is to avoid overfitting (regularization term), as this is computed over the set of known ratings. The minimization of Equation 2.24 can be done by stochastic gradient descent (SGD) or alternating least squares (ALS). For the first case, the system predicts $r_{ui}$ for each rating in the training set. The error is then computed as:

$$e_{ui} = r_{ui} - q_i^T \cdot p_u \tag{2.25}$$

Then the parameters are modified proportionally to $\gamma$ in the opposite direction of the gradient:

$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \tag{2.26}$$

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda q_u) \tag{2.27}$$

The process finishes in a number of iterations, or when the error is less than a threshold. However, sometimes ALS is preferred when the system can effectively parallelize as both $q_i$ and $p_u$ are computed independently, or when the system makes use of implicit data [28].

One important advantage of matrix factorization methods is that we can add a bias in order to improve the predicted rating. The basic bias is defined as:

$$b_{ui} = \mu + b_i + b_u \tag{2.28}$$

where $\mu$ is the mean of all ratings of the system, and $b_i$ and $b_u$ represent the deviation of item $i$ and user $u$ respectively. Then, the rating prediction becomes:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \cdot p_u \tag{2.29}$$

And the minimization becomes in this case:

$$\min_{q*,p*,b*} \sum_{(u,i)\in\mathcal{R}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2) \tag{2.30}$$

In [25], the authors introduce new variables to the matrix factorization problem. More specifically, a set of binary variables indicating the preference of user $u$ to item $i$ denoted as $\pi_{ui}$; these variables take the value 1 if the user has consumed the item (i.e., $r_{ui} > 0$). A confidence variable is also defined as:

$$c_{ui} = 1 + \alpha r_{ui} \tag{2.31}$$

where $\alpha$ is a parameter to be configured. Then, as explained before, we need to compute the vectors $p_u$ and $q_i$ using the new variables $\pi_{ui}$ and $c_{ui}$:

$$p_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u \pi_{u\cdot} \tag{2.32}$$

where $Y$ is a $n \times k$ matrix, with $n$ and $k$ the number of items and factors respectively, $C^u$ is a diagonal matrix where $C^u_{ii} = c_{ui}$ and $\pi_{u\cdot}$ is the vector of user preferences, that is, a vector with the values $\pi_{ui}$ in each dimension for every item $i$. The item vector is computed in a similar way:

$$q_i = (X^T C^i X + \lambda I)^{-1} X^T C^i \pi_{\cdot i} \tag{2.33}$$

These are not the only MF techniques that can be applied in collaborative-filtering RS. Other interesting approaches reported in the literature are the probabilistic Latent Semantic Analysis (pLSA) [24], that uses some hidden variables $Z$ so that the users and items are assumed independent given the variable $Z$. The different values of $z \in Z$ are finite and of size $k$, leading to this model:

$$P(i \mid u) = \sum_z P(i|z)P(z|u) \tag{2.34}$$

Another reported MF technique is the Latent Dirichlet Allocation (LDA) that complements pLSA with a Dirichlet distribution [8].

### 2.2.5 Hybrid recommender systems

Hybrid recommendation takes advantage of the techniques from two or more recommender systems to achieve a higher performance while limiting the potential drawbacks that each system may obtain separately [36]. Although there are hybrids that combine implementations of recommendations of the same type (for example, two content-based techniques), the most interesting ones are those that are able to work with different recommenders. There are different strategies for hybrid recommendation which [10] summarized in the seven methods shown in Table 2.3.

**Table 2.3**: Hybridization methods from [10].

| **Hybrid technique** | **Description** |
|---|---|
| Weighted | Each recommender obtains a score for each candidate item and these scores are combined using a linear formula. |
| Switching | It switches between recommenders depending on the situation. |
| Mixed | Each recommender makes its own recommendations and the final output is a combination of them. |
| Feature combination | Features derived from various sources are combined and sent to the recommendation scheme. |
| Feature augmentation | Similar to feature combination but instead of deriving, the recommenders augment (compute) new features and send them to the final recommendation scheme. |
| Cascade | They normally use a weak and a strong recommender. The weak recommender is only used when breaking ties in the ranking. |
| Meta-Level | A recommender produces a model, which is the input for the second recommender. Similar to feature augmentation but the second recommender does not work with raw data, only with the model provided by the first recommender. |

### 2.2.6 Sequential recommenders

Because in this work we deal with sequential information, we now review the most important techniques that deal with this, not so well-known, recommendation task. In sequential recommendation each user is associated with a sequence of actions $S(u) = (S_1, S_2, S_3, .., S_n)$ (following the notation introduced in [22], that is similar to the one proposed in [34]). These actions can be the items the user has consumed, or the places that s/he has visited, depending on the system domain. The main goal here is to estimate the next element on the sequence, that is, $S_{n+1}$, and recommend suitable items for that sequence. This process differs from the generic recommendation task in that they rely only on the latest user events to estimate future actions. In contrast, the general recommendation problem learns what the user typically likes in a global way.

Because of this, classical collaborative-filtering approaches may not be the best techniques to make sequential recommendations. For capturing these sequential patterns, a common approach is to use an $L$-Markov chain model, where $L$ denotes the number of previous actions that we will consider to make the recommendation. An $L$-order Markov chain for the user sequence can be defined as:

$$P(S_t \mid S_{t-1}, S_{t-2}, .., S_{t-L}) \tag{2.35}$$

The most simple approach is to consider a first order ($L = 1$) Markov chain in which the probability of choosing item $j$ given the actual item $i$ at the next step, $p(j \mid i)$, is obtained by using maximum likelihood estimation in the item-to-item transition matrix. For the basic Markov chain models, this transition is the same for all users. However, [34]

proposed a Factorizing Personalized Markov Chain (FPMC) in which each user has its own transition matrix, dealing to a global representation of a tensor. Besides, they make use of factorization techniques and combine them with the Markov chain method, dealing to a probability proportional to the sum of the inner products of the vector representations of user $u$, item $i$, and the last item the user has consumed, item $l$:

$$p_u(i \mid l) = q_i^T \cdot p_u + q_l^T \cdot p_u + q_i^T \cdot q_l \tag{2.36}$$

where $p_u$ and $q_i$ are the projections of users and items on the $k$-dimensional latent space (matrix factorization component). Note that this formula is not exactly equivalent to the one in [34], as in that case, they work with every item in the last basket of products consumed by the user.

Another interesting approach is the one proposed in [22]. In this case, instead of using a first-order Markov chain, the authors generalize this method to an $L$-order making a weighted sum for the short and long term dynamics of the user preferences. In that paper, the authors combine Markov chains with Factored Item Similarity Models (FISM) into an approach called Fossil (Factorised Sequential Prediction with Item Similarity Models). According to the reported results, this approach outperforms many sequential state-of-the-art algorithms. In this context, FISM is a method used to decompose the matrix $W$ of the Sparse Linear Methods (SLIM) shown in [32]. In SLIM, the recommendation score of a user to an item is computed as:

$$\hat{a}_{ij} = a_i^T \cdot w_j \tag{2.37}$$

where $a_i^T$ is the rating history of $u$ on all items, considering the value 0 in those items that have not been seen and $w_j$ is a column vector of aggregated coefficients. FISM approximates the item-to-item similarity matrix $W$ as:

$$W = P \cdot Q^T \tag{2.38}$$

where $P$ and $Q$ are two matrices of size $|\mathcal{I}| \times k$, with $k$ the number of latent dimensions.

Finally, there are some articles where the LCS technique has been used as a pattern finding algorithm. In [40], the authors created a recommendation system using the LCS algorithm. They stored the information from the transactions of previous users and, based on that history, a list of recommended products is displayed. Based on this information, they used LCS to predict and recommend users' future requests. However, to the best of our knowledge, no prior work has used this technique to create a sequential recommender or as a similarity metric, something that we will explore in the rest of this work.

## 2.3 Evaluation

Evaluation is a necessary step in analyzing the effectiveness of a recommendation system. However, there is no single way to evaluate a recommendation technique, since there are several aspects that can be analyzed. In some cases, we are only interested in predicting ratings and computing the error between these predictions and the real ratings (error metrics), but sometimes it is useful to provide a list of potentially relevant items to the user and check their real relevance (ranking quality) or analyze how novel and diverse the recommended items are as users tend to value better recommendations of different characteristics (novelty and diversity metrics). Due to this variety of metrics, there is typically no algorithm that outperforms every other method in all possible situations [21]. In this section we present these three evaluation approaches, which are currently the ones most popular to evaluate recommendation techniques.

### 2.3.1 Error metrics

A way to test how good a recommender system performs is to compare ratings returned by the system against the ones in a test split of a dataset. The two most popular metrics to do this are Mean Absolute Error (MAE, Equation 2.39) and Root Mean Square Error (RMSE, Equation 2.40) [19]:

$$\text{MAE} = \frac{1}{|\mathcal{R}_{test}|} \sum_{r_{ui} \in \mathcal{R}_{test}} |g(u,i) - r_{ui}| \tag{2.39}$$

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{R}_{test}|} \sum_{r_{ui} \in \mathcal{R}_{test}} (g(u,i) - r_{ui})^2} \tag{2.40}$$

In both metrics, $\mathcal{R}_{test}$ denotes the ratings in the test set, $g(u,i)$ is the predicted rating, and $r_{ui}$ denotes the real rating (note that as the difference is squared or absolute, these values can be exchanged). RMSE tends to penalize more large errors than MAE [21]. Higher values indicate that the recommender predicts ratings farther apart from real ones and, hence, they are bad predictions. Nevertheless, even though this kind of evaluation was one of the first metrics used in recommendation and the one used in the Netflix prize [31], it is incomplete, as these metrics can only be applied to observed ratings, while real-world problems may take more benefit from solving the ranking-oriented evaluation problem [41].

### 2.3.2 Ranking quality evaluation

Sometimes ratings are not available because we only have a list of items that the user likes to test our recommender. In this case the evaluation is done by comparing the list returned by the recommender and the test list of the user. Hopefully, this list is created by the most useful articles for the user, ordered according to a specific ranking that can be obtained by maximizing the function $g$ presented in Equation 2.1. Some of the most important metrics are Precision (Equation 2.41), Recall (Equation 2.42), Average Precision (Equation 2.43) and nDCG (Equations 2.44 and 2.45). Normally all of them are computed for each user of the system, so the result for the whole system is obtained by taking the average of those per-user metric values:

$$\text{Precision} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Retrieved}|} \tag{2.41}$$

$$\text{Recall} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Relevant}|} \tag{2.42}$$

$$\text{AP} = \frac{1}{|\text{Relevant}|} \sum_{\{k:d_k \in \text{Relevant}\}} P@k \tag{2.43}$$

$$\text{nDCG} = \frac{\text{DCG@}p}{\text{IDCG@}p} \tag{2.44}$$

$$\text{DCG@}p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{2.45}$$

In AP, $d_k$ denotes the document at position $k$ in the returned list and $rel_i$ in nDCG and DCG denotes the real relevance of item $i$ (normally from 1 to 5, no relevant items are treated as 0). IDCG is computed in the same way as DCG but with the list of relevant documents ordered by descending relevance. In all of them, instead of evaluating the full list of recommended items, a top-$N$ sublist is normally evaluated, represented as @$p$, where this $p$ is called *cutoff*. This is because users often prefer short instead of long lists. All these metrics are bounded in $[0,1]$, where 1 is the maximum value (and the optimal one) that can be achieved and 0 the lowest one.

### 2.3.3 Novelty and diversity

As mentioned before, ranking evaluation is the most extended way to analyze the performance of recommender systems. However, this evaluation is "incomplete" as it does not take into account the novelty or diversity of the items. It is relevant to show that although novelty and diversity are related, they are not the same. Novelty refers to how different the recommended items are to the ones that the user has previously consumed, whereas diversity, on the other hand, analyzes how different to each other are the recommended items [42]. These dimensions of recommendation should be as fundamental as the evaluations presented in previous subsections. The most important metrics that are used to measure novelty and diversity according to [13, 42] are:

**Average Intra-List Distance**

One of the first diversity metrics proposed. It is computed as the average pairwise distance of the items in the set of recommended items $R$:

$$\text{ILD} = \frac{1}{|R|(|R|-1)} \sum_{i \in R} \sum_{j \in R} d(i,j) \tag{2.46}$$

where $d(i,j)$ is a distance measure that can be specified.

**$\alpha$-nDCG**

This metric was proposed in [15]. It is a diversity metric extension of the nDCG metric shown in Equation 2.44. The idea of this metric is to penalize the score of retrieved documents if they share features with documents ranked higher in the list (the higher the value of $\alpha$, the higher the penalization). As nDCG, it computes the division between DCG and IDCG. In this case, the gain vector is computed as:

$$G[k] = \sum_{i=1}^{m} J(d_k, i)(1-\alpha)^{r_i, k-1} \tag{2.47}$$

$$r_{i,k-1} = \sum_{j=1}^{k-1} J(d_j, k) \tag{2.48}$$

And DCG formula is:

$$\text{DCG}[k] = \sum_{j=1}^{k} \frac{G[j]}{\log_2(1+j)} \tag{2.49}$$

where $J(d_k, i) = 1$ if $d$ contains the "keyword" or "nugget" $i$ and $J(d_k, i) = 0$ otherwise. In general, for each item, the value of DCG will be smaller whenever more articles contain the same keywords.

**Aggregate diversity**

A diversity metric which counts the total number of items that the system recommends:

$$\text{AggDiv} = \left| \bigcup_{u \in \mathcal{U}} R_u \right| \tag{2.50}$$

**Gini Index**

A diversity metric that shows how unequally different are the items chosen by a particular system $s$:

$$\text{Gini} = 1 - \frac{1}{|\mathcal{I}| - 1} \sum_{k=1}^{|\mathcal{I}|} (2k - |\mathcal{I}| - 1) p(i_k \mid s) \tag{2.51}$$

$$p(i \mid s) = \frac{|\{u \in \mathcal{U} | i \in R_u\}|}{\sum_{j \in \mathcal{I}} |\{u \in \mathcal{U} | j \in R_u\}|} \tag{2.52}$$

where $p(i_k \mid s)$ is the probability of the $k$-th least recommended item being drawn from the recommendation list. In this case, we will use the complementary of the Gini Index proposed in [13], as defined in [43].

**Other metrics**

To expand knowledge about more metrics of novelty and diversity, we look for this definition obtained from [42], in which the authors define a general recommendation metric scheme affected by the relevance over a ranked list as:

$$m(R \mid \theta) = C \sum_{i_n \in R} \text{disc}(n) p(rel \mid i_n, u) \text{nov}(i_n \mid \theta) \tag{2.53}$$

where $\theta$ is a contextual variable that represents the element on which item discovery depends (for example time intervals or a group of users). $C$ is a normalizing constant to avoid biases in the metric (normally, $C = 1/\sum_{i_n \in R} \text{disc}(n)$), $\text{disc}(n)$ represents a discount function, which serves to model the fact that items placed in a lower position in the ranking are less likely to be seen (for example, $\text{disc}(n) = 1/\log_2 n$). The term $p(rel \mid i_n, u)$ represents the probability of being relevant given the item and the user, and $\text{nov}(i_n \mid \theta)$ is the function to be modeled depending on the metric we are using. Thus, for Expected Popularity Complement (EPC) we have that $\text{nov}(i_n \mid \theta) = 1 - p(seen \mid i, \theta)$, that means the complement of the probability that the item was seen given the context and the item. This leads to this equation:

$$EPC = C \sum_{i_k \in R} \text{disc}(k) p(rel \mid i_k, u)(1 - p(\text{seen} \mid i_k)) \tag{2.54}$$

For other metrics like expected inverse popularity (EIP), the term $\text{nov}(i_n \mid \theta)$ was defined as $-\log_2 p(\text{seen} \mid i, \theta)$ whereas for Expected Free Discovery (EFD), $\text{nov}(i_n \mid \theta) = -\log_2 p(i \mid \text{seen}, \theta)$. The other functions remain the same. However, in the same article, the authors also described a distance-based novelty model. For example, the Expected Profile Distance (EPD) metric can be expressed as:

$$EPD = C \cdot \frac{\sum_{i_k \in R, j \in \mathcal{I}_u} disc(k) p(rel \mid i_n, u) p(rel \mid j, u) d(i_k, j)}{\sum_{j \in \mathcal{I}_u} p(rel \mid j, u)} \tag{2.55}$$

where $d(i_k, j)$ is the complementary of any possible similarity metric like Pearson or Cosine. Besides, they also defined a diversity metric similar to ILD, shown in Equation 2.46 with rank discount and relevance weighting (Expected Intra-List Diversity).

## 2.4 Datasets

In this section, we describe the different datasets that have been used in the experiments reported later. All of them are publicly available datasets and vary from the movie (MovielensHetRec, MovieTweetings) to music (Lastfm) domain. Due to the intrinsic nature of the data, in most cases (except in Lastfm) it was not necessary a previous pre-processing step. Specific details of each dataset are shown below:

**Table 2.4**: Statistics about the datasets used in the experiments.

| Dataset | Users | Items | Ratings | Density |
|---|---|---|---|---|
| MovielensHetRec | 2,113 | 10,197 | 855,598 | 3.97% |
| LastFm | 1,892 | 17,632 | 92,834 | 0.28% |
| MovieTweetings | 45,324 | 26,087 | 541,304 | 0.045% |

**MovielensHetRec**

The MovielensHetRec dataset[1] is a subset of the Movielens10M dataset in which only users with both ratings and tags have been kept [12]. Each movie of the dataset has a lot of information associated with it (movie title in English and Spanish, countries of origin of the movies, etc), but we will only use genres and directors as complementary information of the ratings. Ratings in this dataset are made on a 5-star scale with half-star increments (from 0.5 stars to 5.0 stars). This dataset includes timestamps for each interaction between a user and an item.

**LastFm**

The Lastfm dataset can also be obtained in the same url as the previous one. This dataset is the only one among those reported that does not contain explicit data, but only implicit data (frequencies of the listened artists instead of ratings). We transformed the frequency numbers to ratings (implicit to explicit data) in a similar way as defined in [14]. Thus, the ratings can be obtained in this way:

$$\tilde{r}_{ui} \sim \left\lceil 5 \cdot \frac{\mathcal{F}_{ui}}{\max \mathcal{F}_u} \right\rceil$$

The transformation is done by dividing the number of listenings of the specific artist ($\mathcal{F}_{ui}$) and the maximum number of listenings of that user ($\max \mathcal{F}_u$). The factor 5 allows us to create ratings between 1 and 5.

This dataset, on the other hand, does not contain any temporal information, since only aggregated information for each (user, artist) pair is provided, instead of information at track-level, where a timestamp would be available.

**MovieTweetings**

The MovieTweetings dataset[2] consists of ratings on movies that were contained in well-structured tweets on Twitter. Ratings go between 1 to 10 with no half-scales. Although there are many snapshots, we decided to use the latest dataset. We will take advantage of the timestamps from this dataset in order to make recommendations. At the time that we were working with this data, the largest (newest) timestamp was 1476136977.

[1] `https://grouplens.org/datasets/hetrec-2011/`

[2] `https://github.com/sidooms/MovieTweetings`

# Chapter 3

# LCS as a similarity metric

As mentioned before, one of the purposes of this Master's Thesis is to analyze the performance of LCS as a similarity metric against other important state-of-the-art recommenders. In fact, as we shall show in this section, this algorithm can be generalized to work in a purely collaborative-filtering approach or in a hybrid content-based and collaborative-filtering system taking advantage of different sequence orderings.

Firstly, we show the formulation used to transform users into generic sequences (Section 3.1). Afterwards, some extra parameters are introduced in order to improve the performance of the algorithm in terms of both accuracy metrics and execution time (Section 3.2). Subsequently, we introduce four possible normalization functions in order to bound the LCS-based similarity (Section 3.3) and, then, we discuss different ways to create sequences, including a time-based ordering of the items, whose purpose is to integrate the temporal dimension into the similarity metric (Section 3.4). We end this chapter with a toy example to better understand how this new approach works in detail (Section 3.5) and a discussion about which specific instantiations of an LCS-based similarity metric are equivalent to more classical similarity metrics (Section 3.6).

## 3.1 Representing users as sequences

In Section 2.1 we presented the original LCS procedure to work with strings as sequences (Algorithm 1). In a $k$-NN recommender, instead of two strings, we receive two users or two items and the longest common subsequence between them will be considered as their similarity. Hence, in order to adapt this algorithm into the recommendation domain, it is necessary to define how to represent the users or items properly as sequences and, according to that representation, we need a function that identifies when two characters are the same.

In the rest of the chapter, we focus on the case of user similarity, that is, $\text{sim}(u, v)$, although this formulation can be adapted for an item similarity in an analogous way. Hence, we need to generate sequences from the information related to a user, and apply the LCS algorithm to such sequences. After the sequences are generated, an adaptation of Algorithm 1 is needed so that we can compute the LCS between them.

Our adaptation of the LCS algorithm as a similarity metric between users can be seen in Algorithm 3. As the reader may observe, there are two relevant modifications to the original approach: a function that transforms users $u$ and $v$ to sequences $x$ and $y$, denoted as $f$, and a $\delta$-matching that allows us to configure when two symbols of the alphabet are considered equal. This second modification is introduced to allow considering two users as equals even when a small rating difference is found, because of this we changed the original matching condition of the LCS algorithm by deciding that two symbols are

**Algorithm 3** Longest Common Subsequence for Recommender Systems

```
1: procedure LCS_RecSys(u, v, f, δ)          ▷ The LCS of users u and v applying transformation f
2:     (x, y) ← (f(u), f(v))                  ▷ String x contains m symbols
3:     L[0 ··· m, 0 ··· n] ← 0
4:     for i ← 1, m do
5:         for j ← 1, n do
6:             if match(x_i, y_j, δ) then
7:                 L[i, j] ← L[i − 1, j − 1] + 1          ▷ There is a δ-matching
8:             else
9:                 L[i, j] ← 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
```

equivalent when their difference is below the variable $\delta$. Nevertheless, this is not the only possible configuration. We can expand this matching threshold by exploiting the semantic information of the items (for the domain of movies or books, we can consider that two articles are the same if they share any genre).

However, the most critical component in this approach is the transformation function $f$. Depending on the information that we are using to represent users, we can obtain different sequences and hence disparate results. Assuming the user can be described as a set of items and ratings (or any other implicit numeric data, like click counts, access frequencies, or binary interactions), we describe the following steps to generate a sequence in a formal and generic way:

1. **Extend the associated information about the items interacted by the user.** Formally, we need a function that returns a set of elements associated to every item. That is, a function of the form: $e : \mathcal{I} \times \mathcal{R} \to \mathcal{I} \times \mathcal{T}^k$, 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. Note that a pure CF approach is derived from this formulation if we use the identity function in this step: $e_{ir}(i, r) = (i, \{i, r\})$. Nevertheless, 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 or text features in news recommendation.

2. **Represent the tuples created as interpretable symbols by the LCS algorithm.** Here, 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. The reason why we use the set of integer numbers instead of strings or other space is that they are computationally more affordable. As a simple example, associated to the function $e_{ir}$ we define the function $t_{ir}(i, r) = 10 \cdot \mathrm{id}(i) + r$ in such a way that it is also possible to recover the original elements of the tuple (the item identifier and the corresponding rating of the interaction) given its output (bijective function). The factor of 10 that multiplies the id allows us to separate the item id and the rating while combining them into a single symbol; obviously, this factor depends on the rating interval. Thus,

if ratings are, for example, between 1 and 20, the transformation function should be $t_{ir}(i, r) = 100 \cdot \mathrm{id}(i) + r$ in order to make that recovery possible.

3. **Arrange the symbols into a sequence.** In string matching, the ordering of the sequence is important, and it is an aspect that the LCS algorithm is able to exploit. As a first approach, this step can be simplified to just 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, by item popularity. In this way, the sequence arranging function proposed will take several pairs of items and tuples generated as described before and will output a sequence of symbols, prepared to be processed by the LCS 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 get an intuitive idea about this process, let us present an example considering a dataset based on movies, which usually have some content-based information associated like actors, directors, or genres (which can be extended to work with tags or, for example, the most important keywords from the synopsis). We have the film *Avatar*, with id 10, and a user $u$ who has rated it with a 4. 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 10 has three genres: Adventure (id 1), Sci-Fi (id 6) and War (id 15). According to the definition of $e_{gr}$, this function leads to the tuple $(10, \{\{\text{Adventure}, 4\}, \{\text{Sci-Fi}, 4\}, \{\text{War}, 4\}\})$. After that, we would represent these tuples as useful symbols for LCS using a reasonable $t_{gr}$ function. By using a similar one to $t_{ir}$, we could transform each genre into its id and using that value in combination with its associated rating, creating the tuple $(10, \{14, 64, 154\})$. To generate the sequence corresponding to this user, we take the tuples associated to the only item this user has rated: $(14, 64, 154)$.

However, if the user has also rated *Goodfellas* (with a rating value of 5 and whose id is 15), then the output is slightly different. This movie has Drama (id 2) and Crime (id 7) as genres. The tuple related to this second movie would be (following the same steps as before, i.e., using $t_{gr} \circ e_{gr}$): $(15, \{25, 75\})$. The final step would produce the sequence $(14, 64, 154, 25, 75)$, since the id of *Avatar* is lower than the one for *Goodfellas*. Note that if $e_{ir}$ and $t_{ir}$ functions are used (pure collaborative-filtering information), then each item will only generate one tuple and the final generated sequence will be shorter: $(104, 155)$.

From now on, the sequence generation function $f$ will be denoted in the same way as their respective transformation function $t$; for instance, $f_{ir}$ corresponds to the transformation $t_{ir}$ where items and ratings are considered for the sequence generation, and similarly for $f_{gr}$. These sequences (generated either using content-based or collaborative information) will be then 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 a standard way. As we shall see, the way these sequences are generated has a critical impact in the final performance of the recommendation algorithm. Recalling the taxonomy shown in Table 2.3, the reader may see that when making use of content-based information, this recommender becomes a hybrid system with a feature combination scheme.

**Table 3.1**: Summary of parameters described.

| Parameter | Notation | Description | Effects |
|---|---|---|---|
| Preference | $\gamma$ | It only considers items that have been rated with a rating higher or equal than the preference value | Computation time reduction<br>Coverage reduction |
| Confidence | $\tau$ | It only considers neighbors whose similarity is higher or equal than the confidence value | Quality of neighbors improvement<br>Coverage reduction |
| Threshold | $\delta$ | It allows to consider two items equals if their rating difference is lower or equal than the threshold | Quality of neighbors reduction<br>Coverage improvement |

## 3.2 Preference and confidence

As shown previously, the LCS algorithm has a complexity of $O(mn)$, with $m$ and $n$ being the length of the sequences to be compared. When receiving very large sequences, computing LCS between all users may become too expensive in terms of computational cost. However, the length of both sequences can be reduced if the less important items are filtered. We introduce a parameter, denoted as $\gamma$ (we will name it *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.

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 this case, it would be enough by having an extended function $e^{\gamma}$ that only outputs values whenever the associated rating is above the $\gamma$ threshold; hence, the corresponding $f^{\gamma} = s \circ t \circ e^{\gamma}$ would work as explained in the previous section. In both cases, Algorithm 3 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 $\delta$, with a value $\leq \delta$). We have added this parameter in the model, where we denoted it as *confidence filter* and is represented by $\tau$. This parameter imposes a harder constraint on the potential neighbor, reducing the number of possible neighbors that a user may have. This parameter is equivalent to the *threshold filtering* defined in [19].

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. Table 3.1 summarizes the main effects (positive and negative) of these parameters.

## 3.3 Normalization functions

The LCS algorithm obtains a value in the interval $[0, \min(|f(u)|, |f(v)|)]$ when calculated for two users $u$ and $v$. 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 utility function $g(u, i)$ [19]. Following the same rationale, we propose four normalizations for our LCS-based similarity metric:

$$\text{sim}_1^{f,\delta}(u, v) = \text{LCS\_RecSys}(u, v, f, \delta) \tag{3.1}$$

$$\text{sim}_2^{f,\delta}(u, v) = \frac{\text{sim}_1^{f,\delta}(u, v)^2}{|f(u)| \cdot |f(v)|} \tag{3.2}$$

$$\text{sim}_3^{f,\delta}(u, v) = \frac{2 \cdot \text{sim}_1^{f,\delta}(u, v)}{|f(u)| + |f(v)|} \tag{3.3}$$

$$\text{sim}_4^{f,\delta}(u, v) = \frac{\text{sim}_1^{f,\delta}(u, v)}{\max\left(|f(u)|, |f(v)|\right)} \tag{3.4}$$

$$\text{sim}_5^{f,\delta}(u, v) = \frac{\text{sim}_1^{f,\delta}(u, v)}{\min\left(|f(u)|, |f(v)|\right)} \tag{3.5}$$

Except for Equation 3.1, that produces the LCS-based similarity with no normalization, that is, as calculated by the Algorithm 3, the other equations present different normalizations of Equation 3.1, producing values in the $[0, 1]$ interval. 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 [18] to compare the output of the LCS algorithm, in a similar way as the one proposed here.

## 3.4 Sequence ordering

As mentioned in the third step of the sequence generation, there are different orderings that could be taken into account when computing the LCS-based similarity approach. The most elementary ordering that can be applied to generate the sequences is an order based on the id, either ascending or descending. This type of ordering is global, in the sense that it is the same for all users. Because of that, any other global ordering would produce equivalent sequences, for instance, one where the items are sorted according to their popularity. Nonetheless, it might be possible to produce orderings making use of temporal information – for example, exploiting the rating timestamp. In this case, we would sort the items consumed by the user in ascending order as they were consumed (older articles at the beginning and newer articles at the end of the sequence). This approach has the following advantages over a global ordering:

- The LCS-based similarity will not only maximize similarities of users that have rated similar items in the same way, but it will also add a temporal factor by matching users having similar patterns while rating items.
- Recommendations are expected to be more personalized, as the neighbors are selected by matching temporal patterns.
- As the ordering is independent of the transformation function, we are able to match temporal patterns not only by item id but also with content-based data, making a hybrid collaborative-filtering and content-based technique that exploits temporal information. The same can be said about the confidence and preference parameters.


**Figure 3.1**: Example of different user's rating history.

In order to clarify the previous explanation, let us consider the example shown in Figure 3.1. In this case, we have the target user $u$ and the different neighbors $v_1, v_2, v_3$. For the sake of simplicity, we will not consider ratings, that is, all items are liked for each user that have rated it; furthermore, the user history is ordered by timestamp. If we consider a simple collaborative-filtering LCS approach, we can see that the similarities with respect to user $u$ are 3, 4, and 3 for $v_1$, $v_2$, and $v_3$ respectively (see Section 3.5 for more details on these derivations). However, if we consider a classic $k$-NN recommender, we see that all neighbors have rated the same number of articles with respect to user $u$, 4 in this case. In such case, the classic $k$-NN algorithm will obtain the same predicted score for movie *Avatar* ($i_2$) and *Rogue One: A Star Wars Story* ($i_{13}$). However, when time is considered in the ordering function for the LCS-based similarity, the latter movie will receive a higher score since user $v_2$ has a higher similarity than the other neighbors.

## 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 3.2 shows the items consumed by the first user, $u_1$, and Table 3.3 those consumed by the second user, $u_2$. In these tables, some content features (genres and directors) are also present, together with their corresponding ids, to understand how the transformation function generates the sequences in each situation. Table 3.4 shows the sequences obtained for each variation of the transformation function $f$, together with the result computed by the LCS-based similarity function. Here, $f_{ir}$ and $f_{gr}$ denote the functions presented in Section 3.1, and following a similar notation, $f_{dr}$ represents sequences built using directors and ratings, whereas we denote as $f_i$ when only item ids are considered.

The first thing we notice in Table 3.4 is that the matching threshold does not affect the

**Table 3.2**: Toy example. Items consumed by user $u_1$.

| Movie (id) | Director (id) | Genres (ids) | User's Rating |
|---|---|---|---|
| The Wild Bunch (M1) | Sam Peckinpah (D1) | Western (G1)<br>Robbery (G2) | 5 |
| Seven Samurais (M2) | Akira Kurosawa (D2) | Action (G3)<br>Drama (G4)<br>Adventure (G5) | 4 |
| The Iron Cross (M3) | Sam Peckinpah (D1) | War (G6) | 3 |
| Gladiator (M4) | Riddley Scott (D3) | Action (G3)<br>Drama (G4)<br>Adventure (G5) | 4 |

**Table 3.3**: Items consumed by user $u_2$.

| Movie (id) | Director (id) | Genres (ids) | User's Rating |
|---|---|---|---|
| Seven Samurais (M2) | Akira Kurosawa (D2) | Action (G3)<br>Drama (G4)<br>Adventure (G5) | 5 |
| Gladiator (M4) | Riddley Scott (D3) | Action (G3)<br>Drama (G4)<br>Adventure (G5) | 2 |
| Alien (M5) | Riddley Scott (D3) | Sci-Fi (G7)<br>Terror (G8) | 5 |
| The Magnificent Seven (M8) | John Sturges (D4) | Western (G1)<br>Adventure (G5) | 4 |

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 $\gamma$ 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. 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.

## 3.6 Relation with other metrics

It is interesting to observe that the similarity computed using a transformation based on items – i.e., $f = f_i$ – is equivalent to computing the item overlap between the two users ($\mathcal{I}_{uv} = \mathcal{I}_u \cap \mathcal{I}_v$), which is the basis for several metrics such as similarities based on Jaccard or item co-occurrence. Furthermore, when the normalization function $\text{sim}_2$ is applied, such instantiation is almost equivalent to the Jaccard similarity metric and ranking-equivalent to a binary Cosine. The explanation goes as follows: $\text{sim}_2^{f_i,0}(u,v) = |\mathcal{I}_{uv}|^2/(|\mathcal{I}_u| \cdot |\mathcal{I}_v|) \propto_u |\mathcal{I}_{uv}|^2/|\mathcal{I}_v|$, where in the last step we make use of a ranking-equivalent transformation

**Table 3.4**: Toy example. User representation as sequences and LCS-based similarity for different transformation functions, matching thresholds ($\delta$), and preference ($\gamma$) and confidence ($\tau$) filters.

| $f$ | $\delta$ | $\gamma$ | $\tau$ | $f(u_1)$ | $f(u_2)$ | $\text{sim}_1^{f,\delta}(u_1,u_2)$ |
|---|---|---|---|---|---|---|
| $f_i$ | 0 | 0 | 0 | $(1,2,3,4)$ | $(2,4,5,8)$ | 2 |
| $f_{ir}$ | 0 | 0 | 0 | $(15,24,33,44)$ | $(25,42,55,84)$ | 0 |
| | 1 | 0 | 0 | | | 1 |
| | 1 | 4 | 0 | $(15,24,44)$ | $(25,55,84)$ | 1 |
| $f_{dr}$ | 0 | 0 | 0 | $(15,24,13,34)$ | $(25,32,35,44)$ | 0 |
| | 1 | 0 | 0 | | | 2 |
| | 1 | 0 | 3 | | | 0 |
| | 0 | 5 | 0 | $(15)$ | $(25,35)$ | 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 | 0 | 0 | | | 4 |
| | 1 | 4 | 0 | $(15,25,34,44,54,34,44,54)$ | $(35,45,55,75,85,14,54)$ | 4 |
| | 1 | 4 | 2 | | | 4 |

(by removing a term that only depends on user $u$); on the other hand, Jaccard similarity between users $u$ and $v$ is computed as $|\mathcal{I}_{uv}|/|\mathcal{I}_u \cup \mathcal{I}_v|$, this similarity is then used to rank the potential users as neighbors, so, let us suppose we are computing the neighbors of user $u$, then $|\mathcal{I}_u \cup \mathcal{I}_v| = |\mathcal{I}_u| + |\mathcal{I}_v| - |\mathcal{I}_{uv}| \propto_u |\mathcal{I}_v| - |\mathcal{I}_{uv}|$; at the same time, binary Cosine (where the interactions between users and items are either 1 or 0) is defined as follows:

$$\cos\text{Bin}(u,v) = \frac{\sum_{i\in\mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i\in\mathcal{I}_u} r_{ui}^2 \sum_{i\in\mathcal{I}_v} r_{vi}^2}} = \frac{\sum_{i\in\mathcal{I}_{uv}} 1}{\sqrt{\sum_{i\in\mathcal{I}_u} 1 \sum_{i\in\mathcal{I}_v} 1}} = \frac{|\mathcal{I}_{uv}|}{\sqrt{|\mathcal{I}_u| \cdot |\mathcal{I}_v|}} \tag{3.6}$$

Therefore, by taking $\sqrt{\text{sim}_2^{f_i,0}(u,v)}$ we find an equivalence with $\cos\text{Bin}(u,v)$. 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 could be integrated under the same formulation.

In fact, if other modifications are allowed, we can even instantiate the Pearson correlation coefficient. Let us consider the sequences generated by the identity function (one symbol per item), natural ordering, exact matching, and no ratings concatenated to the symbol. If we modify Algorithm 3 so that it receives an auxiliary array including the ratings consumed by the user corresponding to each sequence, whenever we find a match, besides increasing the value of LCS by 1, we can accumulate the value $(r_{xi} - \overline{r_x})$ for each user $x$. By definition ($f = f_i$, $\delta = 0$, and natural ordering), this value will only be updated when both users have rated the same item. Once we have finished computing the similarity, we need to perform a new normalization function considering the deviation of each user (see Equation 2.18). In this way, although with a lot of manual tuning, we are able to compute Pearson correlation between two users using a modified version of the proposed approach based on LCS as user similarity.

# Chapter 4

# Experiments

In this chapter we show the results obtained when using the proposed LCS-based similarity metric. We have experimented with three different datasets, analyzing both ranking-based accuracy metrics and novelty and diversity as additional evaluation dimensions. Besides the performance of the proposed similarity, we also show the results obtained by other state-of-the-art recommenders in order to make a comparison between them and the LCS recommenders.

This chapter is organized as follows: firstly, we describe the experimental setup of the experiments, the metrics and configuration of the experiments as well as the different baselines used in order to compare our approach against them (Sections 4.1.1 and 4.1.2). Secondly, we show some results using both pure collaborative-filtering and hybrid content-based and collaborative-filtering approaches, in order to justify the validity of LCS as a similarity metric (Section 4.2). Next, the parameters of our approach (preference, confidence, and normalizations) are analyzed in order to show their performance in LCS-based recommenders (Section 4.3). Then, we illustrate the impact of generating sequences ordered by timestamp (Section 4.4). For all previous cases we analyze the performance in terms of ranking evaluation (mainly through the nDCG metric) and in terms of novelty and diversity (Section 4.5). Finally, we compare our results with other state-of-the-art algorithms and discuss all the results obtained (Section 4.6).

## 4.1 Experimental setup

To analyze the performance of LCS as a similarity metric for both pure collaborative-filtering and content-based LCS similarity approaches, we will work in the first place with the MovielensHetRec dataset, explained in Section 2.4 (in [12] the reader may find a more detailed description of this dataset). As the ratings in this case are in half scale from 0.5 to 5, instead of multiplying the id of the item by 10 as shown in Section 3.1, we will multiply it by 100. For example, for the pure collaborative-filtering approach ($f_{ir}$), if a user has rated the item whose id is 15 with a 4.5, the final representation will be 1545.

We have also experimented with the Lastfm dataset, described in the same section as MovielensHetRec. This second dataset, despite containing content information (tags), will only be used to test the pure collaborative-filtering approach, as the association between items and tags is very sparse. Besides, in this case, and as explained before, a explicit transformation of the data is necessary (see Section 2.4). As we shall see, the following section includes several experiments for the MovielensHetRec dataset; to not clutter this section with results from a different dataset, all results related to Lastfm dataset will be presented in Appendix C.

The last experiments (related to the temporal ordering) will use the MovieTweetings dataset. The main reason for including this dataset (although the domain is the same as in MovielensHetRec, that is, movies) is because the timestamps in this case are more realistic, allowing us to define a meaningful temporal split (the details of this split will be explained later).

### 4.1.1 Evaluation Methodology

In this document, the performance of the different experiments is reported using ranking evaluation metrics (Precision, Recall, MAP, and nDCG, with special emphasis on the last one). For novelty and diversity metrics, we will use EPC and EPD to measure novelty and Aggregate diversity, $\alpha$-nDCG, EILD, and Gini to determine the diversity of recommendations. Unless stated otherwise, all the metrics will work only with the first five items returned by the recommender (i.e., using a cutoff of 5). Furthermore, the results have been obtained by taking into account only the items that are in the test split and have been rated with a 5. The reason behind this is that recommending items that are in the test set but with a low rating should not be considered as good recommendations (this configuration may produce lower results in the mentioned metrics but, in general, the overall comparisons do not differ). Besides, for the MovielensHetRec and the Lastfm datasets, we have 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. For the MovieTweetings dataset, instead of a 5-fold split, we use a temporal split where 80% of the oldest ratings are used for the training set and the rest for test. In all the experiments, we have worked with the raw data (without a pre-processing step), except for the Lastfm dataset, where we have transformed the artist listenings to a explicit rating, as described in Section 2.4. Furthermore, as described in [6] and as a typical methodology in the field, the item rankings for each user will only be composed by items the user has not previously seen, that is, not found in the training split of that user.

For the computation of some novelty and diversity metrics, the RankSys framework requires an auxiliary file specifying the features of each item. These features, for both the MovielensHetRec and MovieTweetings datasets have been the genres of the movies, whereas for the Lastfm dataset, we used the different tags associated to the items. Even though some of the novelty and diversity metrics mentioned in Section 2.3.3 can use a discount function and a relevance model, we have decided to take the most simple approach and ignoring them. In such case, EILD with no discount and no relevance model is equivalent to ILD [42]. Finally, for the metrics that use a distance metric (EILD and EPD), we have used the cosine similarity as a basis for a distance model between items.

All the recommenders have been implemented making use of the RankSys framework[1]. At some point, we also considered making use of the Apache Mahout library[2], but after some preliminary results we found they varied too much with respect to those obtained by RankSys, always producing worse results, as we have shown in [7] (also discussed in Appendix A). In part, this may be due to Mahout rankings being obtained by ordering the items by descending predicted rating (the predicted rating was bounded between the minimum and the maximum value of the ratings), producing more ties between the scores of several items. On the other hand, RankSys produces a ranking based on scores that do not correspond to predicted ratings, hence, they are not bounded. This behavior makes ties more difficult to occur and it was observed that tends to perform better [16], although

[1] `https://github.com/RankSys/RankSys`
[2] `http://mahout.apache.org/`

it prevents from computing error metrics like MAE or RMSE. More details about the libraries and the implementations are presented in Appendix A.

### 4.1.2 Baselines

The most important baselines on which we will base our comparisons are the following:

1. Pop: "PopularityRecommender" from the RankSys framework. The items with more ratings (most popular) will be recommended to users.

2. MF: a matrix factorization recommender from the RankSys framework. Specifically, it refers to "MFRecommender" with "HKVFactorizer", proposed in [25] since it was the best performing factorization method in that library; other methods based on LDA were also tested but their performance was usually lower. This factorization method has also been described in Section 2.2.4.

3. IB: a pure collaborative-filtering recommender using a rating-based similarity between items (cosine or Jaccard). Obtained from the RankSys framework ("ItemNeighborhoodRecommender"). Note that in the RankSys framework two types of item similarity are defined: a binary one (where ratings are ignored and only information about whether an interaction exists is considered) and another one where ratings are considered in the similarity computation (vector-based variations). Some of the implementations that correspond to those defined in Chapter 2 are the binary approaches (for Jaccard), whereas other implementations correspond to the vector-based ones (for cosine).

4. UB: a pure collaborative-filtering recommender using a rating-based similarity between users (cosine or Jaccard). Obtained from RankSys framework ("UserNeighborhoodRecommender"). As in the IB recommender, RankSys supports two types of user similarities: binary and vector-based.

5. PureCB: a pure content-based recommender using the approach explained in Section 2.2.2. However, instead of the TF-IDF approach, this recommender uses a Vector Space Model (VSM) with binary weights as the performance with this approach was better. Thus, the coordinates of the feature vectors will have a 1 in the feature position 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 have been tested as features.

6. CBCF: a hybrid recommender system. In this case, the classical collaborative-filtering formulation is used (Equation 2.10), but making use of content-based similarities (instead of using rating-based ones). These similarities are generated using genres and directors between users by transforming them into a VSM and then computing the cosine similarity. This recommendation approach is the same approximation taken in the Fab system described in [4] (a meta-level hybridization scheme).

7. Fossil: FactOrized Sequential prediction with Item SImilarity ModeLs. This recommender was proposed in [22] and it is a sequential recommender that combines similarity-based methods (like FISM) and Markov Chains, as described in Section 2.2.6.

8. MC: a Markov Chain recommender. It uses a first-order Markov Chain to make recommendations [34].

**Table 4.1**: Configuration of the baselines used in each dataset.

| | **Parameters** | | |
|---|---|---|---|
| **Baseline** | **MovielensHetRec** | **Lastfm** | **MovieTweetings** |
| **UB1** | VectorJaccardSimilarity | | |
| | k=100 | k=30 | k=50 |
| **UB2** | VectorCosineSimilarity | | |
| | k=90 | k=30 | k=100 |
| **IB1** | VectorJaccardSimilarity | | |
| | k=5 | k=30 | k=10 |
| **IB2** | VectorCosineSimilarity | | |
| | k=5 | k=30 | k=5 |
| **CB** | Binary representation<br>Directors as CB information | Not used | |
| **CBCF** | Binary representation<br>k=100<br>Directors as CB information | | |
| **Popularity** | None | | |
| **MF** | HKV factorizer<br>50 factors<br>$\lambda = 0.1$ and $\alpha = 1$ | | |
| **Fossil** | Not used | | L(Markov Chain Order)=1<br>K=10 (Latent Feature Dimension)<br>bias=100 |
| **MC** | | | K=10 (Latent Feature Dimension) |

Note that the PureCB and the CBCF will only be used with the MovielensHetRec dataset, as it is the only one with content-based information. For the last two baselines, we have used the code provided by the authors. However, we made some modifications because the original code did not retrieve a ranking, instead it only considered the last consumed item as a test item (only one item in test). The rest of the code remains the same[3]. These two baselines will only be used in the MovieTweetings experiments. The different parameters of each of the baselines with respect to each specific dataset are presented in Table 4.1, where some of the parameters have been optimized for each dataset (such as the neighborhood size and the similarity metric) whereas the default values of other parameters were considered in some methods (this was the case for Fossil and MC baselines).

[3]The original code can be obtained from `https://drive.google.com/file/d/0B9Ck8jw-TZUEeEhSWXU2WWloc0k/view`.

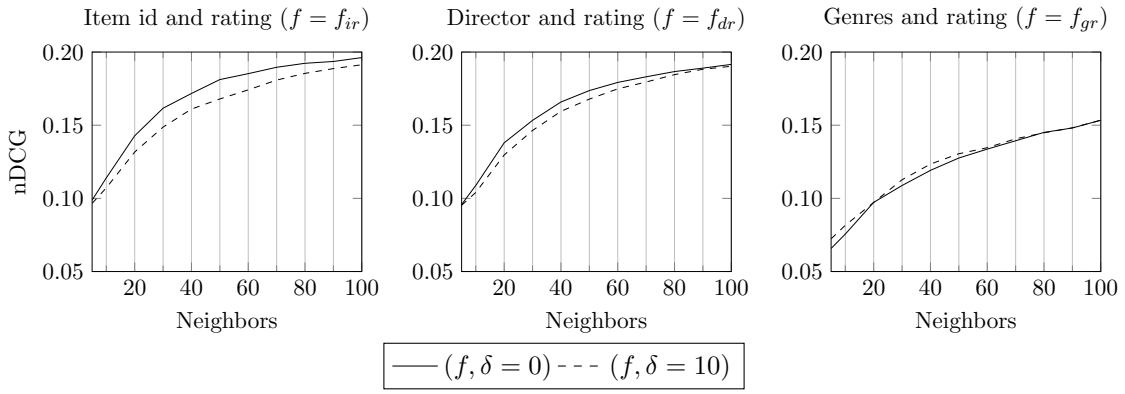


**Figure 4.1**: Results of LCS-based similarity for MovielensHetRec dataset. Transformations based on items (pure collaborative-filtering), directors, and genres using $\delta = 0$ and $\delta = 10$.

## 4.2 LCS as a similarity metric

In this section we show the results using LCS as a similarity metric without any configuration parameter, only experimenting with transformation functions, in the MovielensHetRec dataset. In this case, we denote as $f_{ir}$ (item-rating) 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. We also report two content-based approaches, denoted as $f_{dr}$ (directors-rating) and $f_{gr}$ (genres-rating) as transformation functions. The results of these recommenders are first analyzed in terms of nDCG@5, that is, taking into account only the first five items that the recommender suggests to each user. Later (in Section 4.5) other evaluation dimensions such as diversity and novelty are discussed.

Firstly, in Figure 4.1 we show the results obtained when we use the three different transformation functions. For each transformation, we report results with two $\delta$-matching thresholds: exact matching ($\delta = 0$), with a continuous line and matchings allowing a difference of $\pm 1$ in the rating value (using $\delta = 10$ as explained before due to the presence of half-star ratings in this dataset) with dashed lines. The same scheme is maintained for the rest of the experiments, even though not all the datasets have half scales, which means that, in order to avoid confusion, $\delta = 10$ will always indicate that the similarities are generated allowing a difference of $\pm 1$ in the ratings.

It is interesting to observe that the performance is worse when non-exact matchings ($\delta > 0$) are used. However, this behavior changes depending on the specific configurations used regarding the other parameters available in the model, as we shall see in other experiments later. In this case, 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 see that when integrating content-based information in our LCS-based similarity metric, it performs equally or even worse than applying a pure collaborative-filtering recommender. In this case, the worst recommender is always the one that uses genre information, because of this, the results using genres will be ignored in the rest of the chapter and moved instead to Appendix B . Nonetheless, a possible reason for these low results could be that genres in films are usually very subjective: even if two movies have the same genres, they could be totally different.

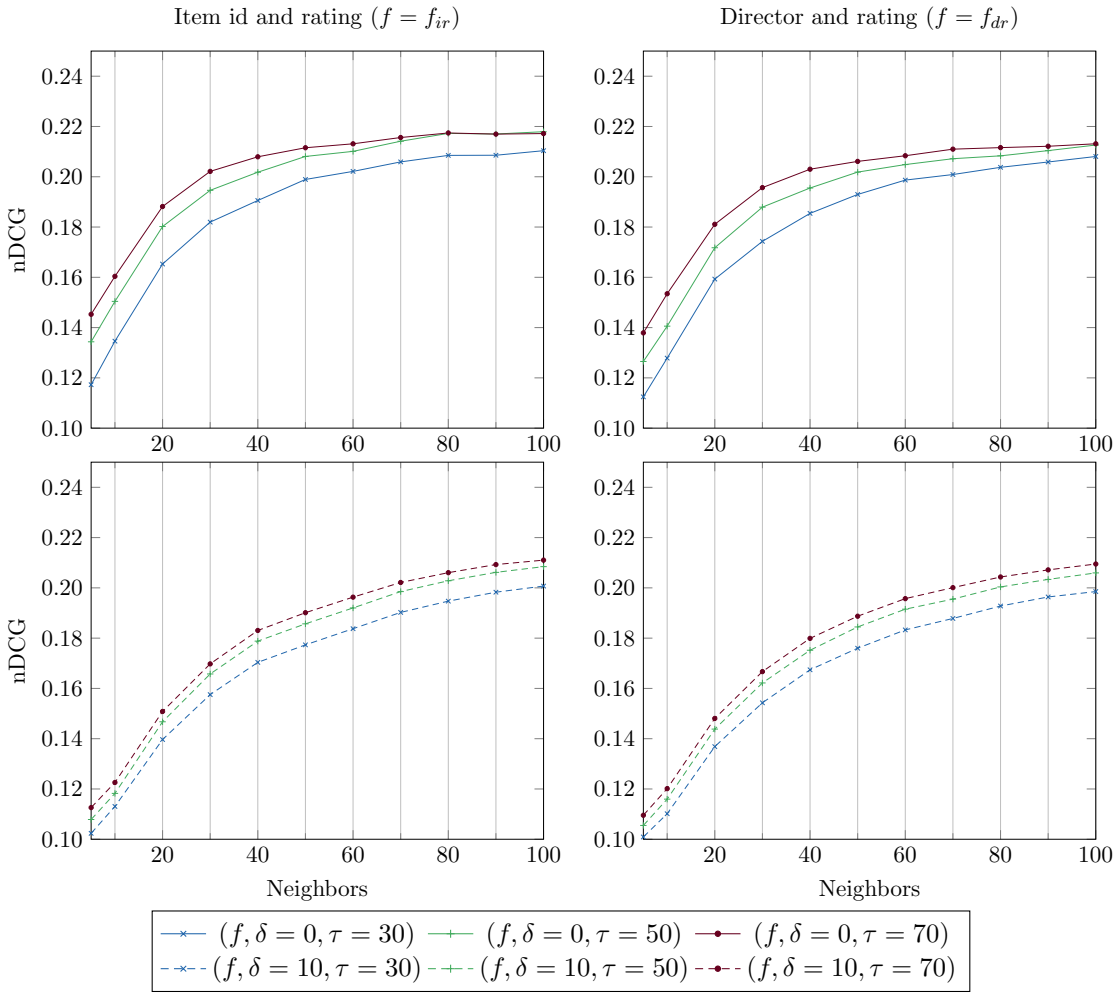


**Figure 4.2**: Results of LCS-based similarity for MovielensHetRec dataset. Different values of confidence filter parameter $\tau$ using $\delta = 0$ and $\delta = 10$.

## 4.3 Sensitivity to confidence, preference and normalization parameters

In this section, we show the results with preference and confidence parameters only for the MovielensHetRec dataset. Appendix C includes the effect of these parameters in the Lastfm dataset with a complementary study of the effect produced in the coverage of user recommendations.

### 4.3.1 Sensitivity to the *confidence filter* parameter

Let us start with the *confidence filter* parameter $\tau$. According to the previous explanation, this parameter works 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 these experiments, we show three different values of confidence (although we have analyzed the behavior for more values): 30, 50, and 70; smaller and larger values had almost no effect, due to sparsity reasons. Nevertheless, the value of this parameter is totally dependent on the dataset and should be tuned in consequence.

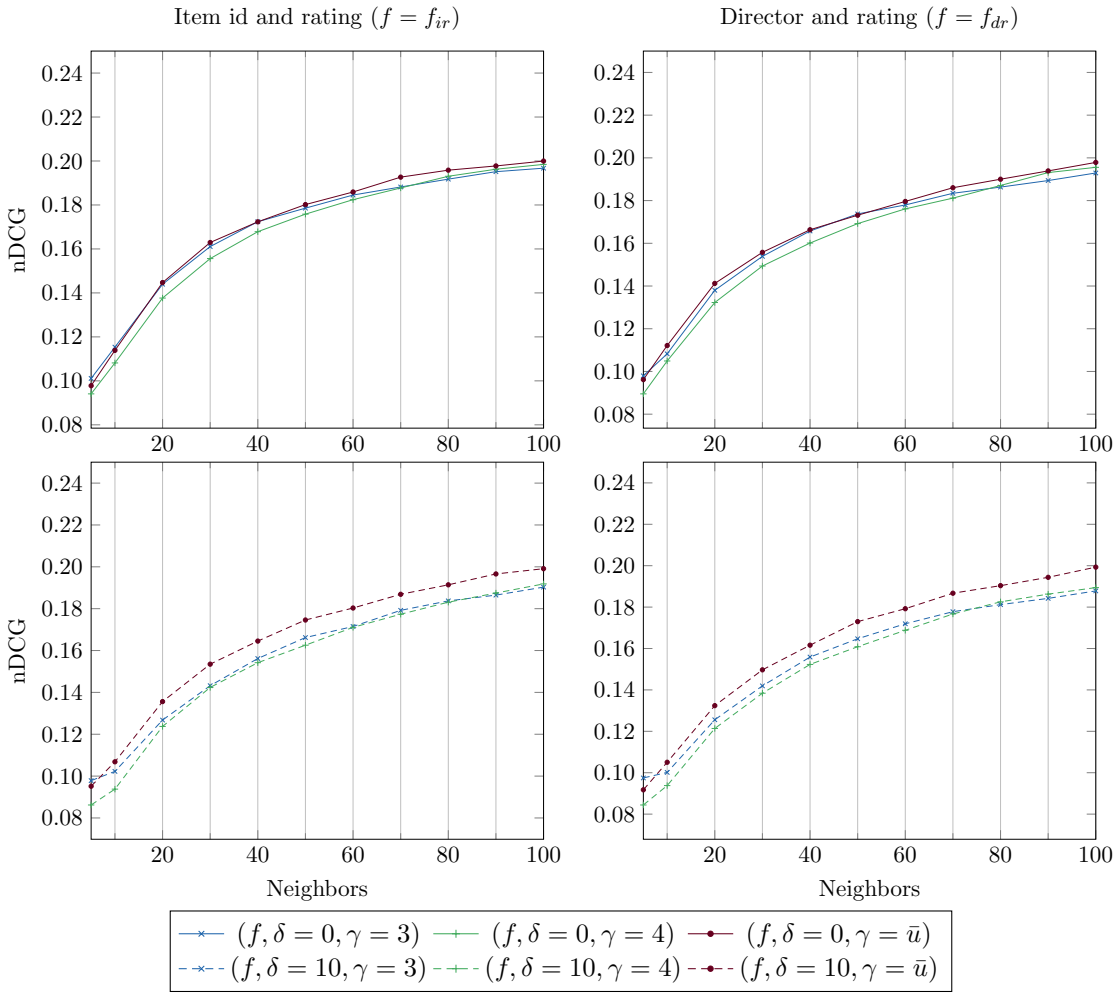


**Figure 4.3**: Results of LCS-based similarity for MovielensHetRec dataset. Different values of preference filter parameter using $\delta = 0$ and $\delta = 10$.

Figure 4.2 shows the performance of the recommenders for the directors and item transformations using the three different values of confidence mentioned before. As in previous experiments, the same recommenders with a $\delta$-matching threshold of 10 are also included. We observe that, when using different values of the confidence filter, we obtain better results. Therefore, we can conclude that a posterior neighbor filtering improves the performance obtained by the LCS-based approaches.

Nevertheless, there are also some similarities with the previous experiment, as the best results are still obtained for the pure collaborative-filtering representation of the user sequences ($f = f_{ir}$) and, at the same time, the exact matching still produces higher improvements than the non-exact matching ($\delta = 10$).

### 4.3.2 Sensitivity to the *preference filter* parameter

We now analyze the *preference filter* parameter, denoted by $\gamma$. This parameter allows us to consider in the user sequences only those items having a rating value higher than a specific value determined by $\gamma$. In these experiments we have considered the following three possibilities: considering ratings higher or equal than 3 ($\gamma = 3$), higher or equal than

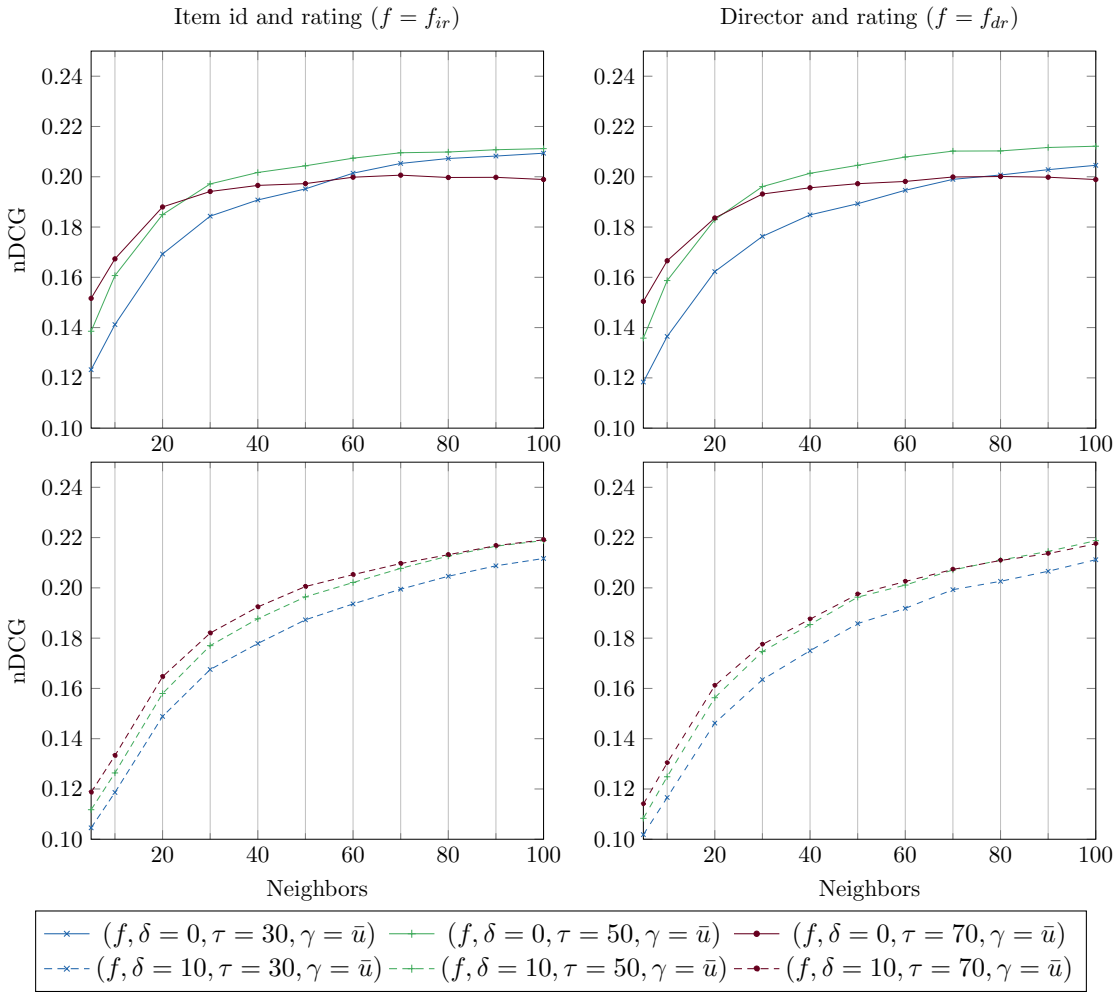


**Figure 4.4**: Results of LCS-based similarity for MovielensHetRec dataset. Combination of different values of preference and confidence filtering using $\delta = 0$ and $\delta = 10$.

4 ($\gamma = 4$), and higher or equal than the user mean ($\gamma = \overline{u}$). In Figure 4.3 we show the results for this experiment.

Although there is some improvement with respect to the basic model without preference filter ($\gamma = 0$ depicted in Figure 4.1), the change in performance is lower than the one achieved with the confidence parameter. This is a very interesting observation, since when using this parameter the similarities are computed with less data, therefore, its efficiency improves with no expenses in performance, according to the results obtained. Furthermore, we can see that the best value for the preference parameter 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.

### 4.3.3 Performance when combining both *confidence filter* and *preference filter* parameters

For the next experiment, we combine both preference and confidence parameters and use them together to see how they affect the recommmender performance. In this case, we

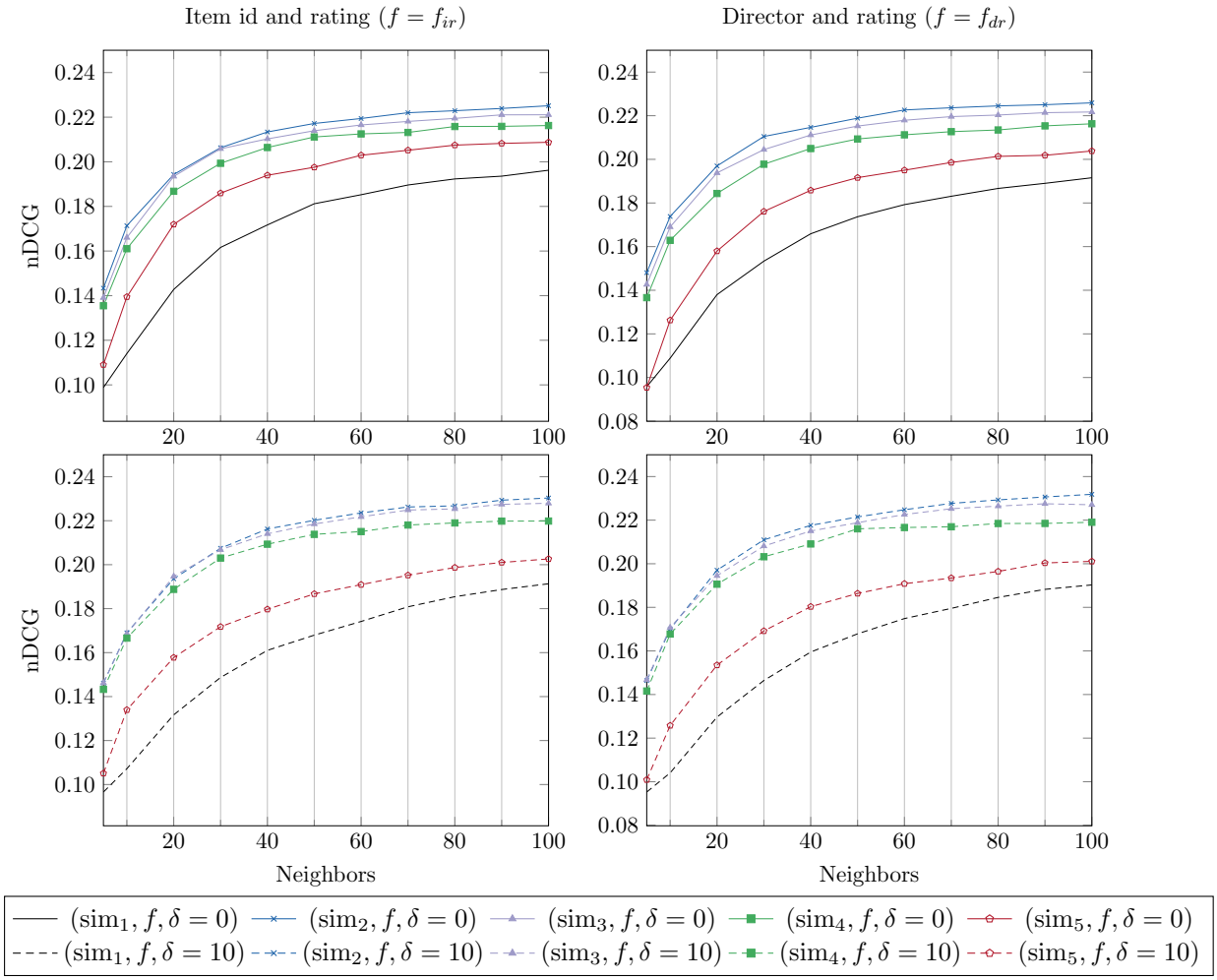


**Figure 4.5**: Results of LCS-based similarity for MovielensHetRec dataset. Different normalization functions using $\delta = 0$ and $\delta = 10$.

are filtering neighbors (according to some confidence value) using only the items that have been rated with a rating higher than the preference value. We restrict this analysis to $\gamma = \overline{u}$ since it has shown better properties to model the user in a generic way.

When using these two parameters at the same time, a general improvement is achieved, as shown in Figure 4.4. Unlike in the rest of experiments, a better performance is found here when non-exact matchings are 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. Besides, in this figure it is relevant to show 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 can be explained as these two parameters reduce the number of possible neighbors. Hence, we need to make use of the $\delta$ threshold in order to palliate this effect, and avoid the dramatically reduction of potential neighbors that prevents an improvement in the recommender.

### 4.3.4 Sensitivity to normalization functions

We now analyze the proposed *similarity normalization* functions. Firstly, in Figure 4.5 we show the performance of the different normalization functions proposed in Section 3.3 when no confidence ($\tau = 0$) and preference ($\gamma = 0$) parameters are used. The objective of this experiment is to compare the non-normalized values ($\text{sim}_1$) against the four proposed normalizations.

From the figure we observe this parameter has a huge effect in performance. Non-normalized recommenders obtain the worst results in every situation. A possible explanation of 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 $\text{sim}_1$ function.

However, we see important differences in the different normalization functions. Interestingly, the best two normalization functions are $\text{sim}_2$ and $\text{sim}_3$, which are the only ones that consider the length of both users when normalizing the original value (the other two only consider the maximum or the minimum length, losing information about one of the sequences). This is a clear indicator that this information is critical and should be used by any similarity normalization function based on LCS. In fact, the worst normalization is the $\text{sim}_5$, that is, the one that divides by the minimum number. This is an expected result. If we denote the length of sequences of user $u$ and two neighbors $n_1$ and $n_2$ as: $|s_u|, |s_{n_1}|$ and $|s_{n_2}|$ respectively, if we have this situation: $|s_u| << |s_{n_1}| << |s_{n_2}|$ with the same similarities between them, we will consider that both neighbors are equal when the first neighbor should be more important as it has the same similarity but with a shorter sequence.

Considering the results from this experiment, we conclude that the previous results are far from optimal in the case of MovielensHetRec. As we have seen, the LCS model improves its performance the most when using normalization functions (more specifically, when using $\text{sim}_2$ and $\text{sim}_3$) and all the results reported at this point (except the last ones) were obtained using no normalization, i.e., $\text{sim}_1$. Therefore, in Figure 4.6 we show the effect of a combination of all the parameters in the model: taking into account preference and confidence filters and the normalization functions. In every case, the best value of each parameter is used (denoted, as mentioned in the caption, with the symbol $*$). From these results, we can make the following observations:

1. The performance improvement between normalization ($\text{sim}_*$) and no normalization ($\text{sim}_1$) is still significant, reproducing the behavior found in Figure 4.5.

2. We obtain better results when non-exact matchings ($\delta = 10$) are allowed.

3. The optimal configurations are different depending on whether exact matchings are allowed: on the one hand, with $\delta = 0$ the best results are obtained by using either the best confidence or preference filter value (very close to each other when using ratings and directors), on the other hand, for $\delta = 10$ the best configuration is consistently found for a combination of the best confidence and preference filter values.

According to these results, we can achieve up to 25% of improvement over the same recommender without any of these variables, where the normalization function plays a critical role in this improvement.

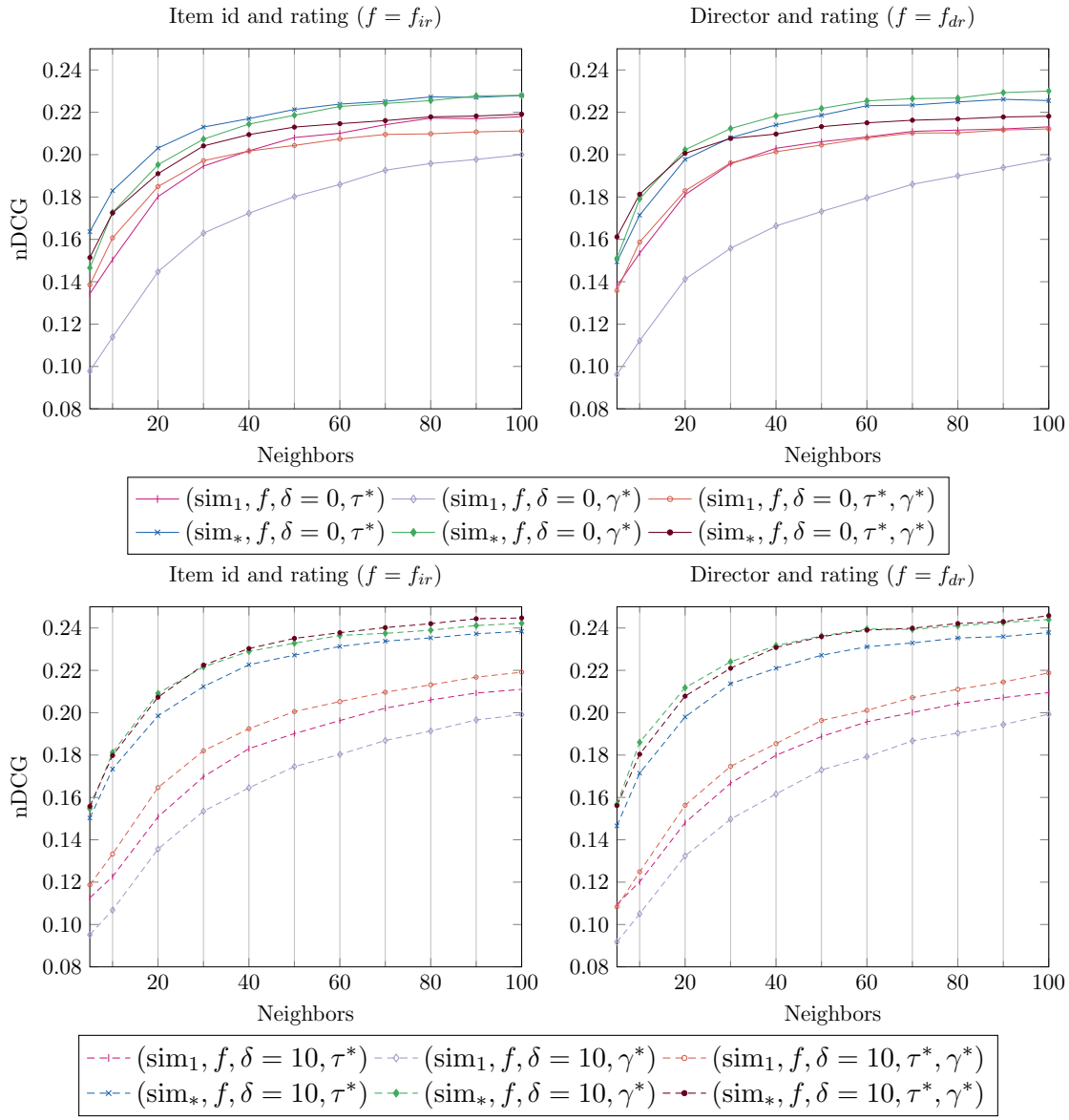


**Figure 4.6**: Results with the best confidence and preference filters and normalizations for MovielensHetRec dataset. Top row shows results using $\delta = 0$, bottom row using $\delta = 10$. The $*$ symbol denotes the best value among the previously reported ones is being used in the combination.

## 4.4 Temporal ordering

In this section we analyze the effect of ordering the user sequences according to the item interaction timestamp, that is, how different functions $s$ (as defined in Section 3.1) affect the proposed model. Figure 4.7 shows the results for the MovielensHetRec using a timestamp ordering (from now on, $s_T$, different from the the ordering function used so far that only takes into account the item id, that we will denote as $s_i$) for both ids and directors and the five normalization functions. We observe the same behavior as in the previous experiments, since all the normalizations obtain a better performance than using no normalization.

However, we found no improvement over the recommenders that ordered the sequences

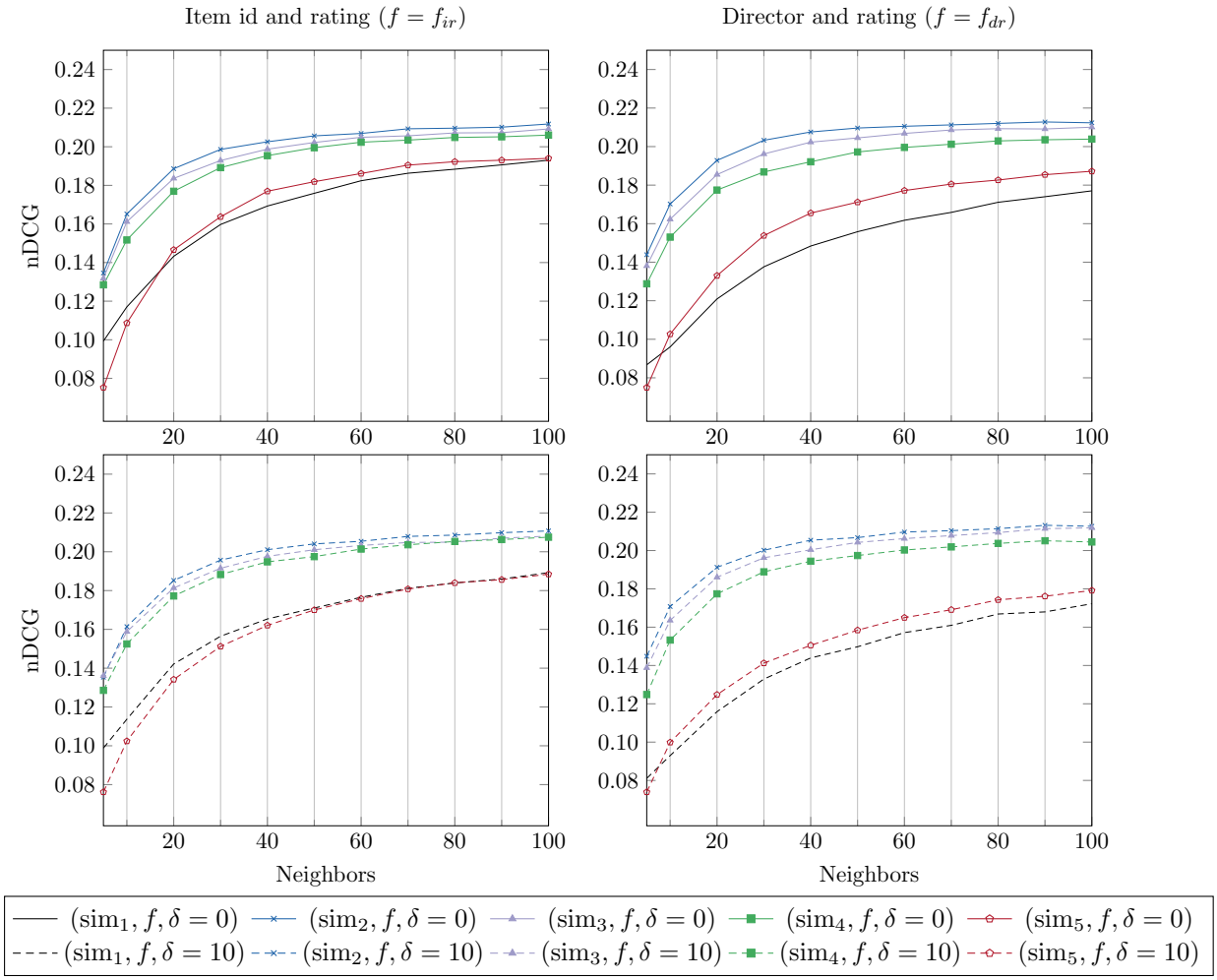


**Figure 4.7**: Results of LCS-based similarity for MovielensHetRec dataset. Different normalization functions using $\delta = 0$ and $\delta = 10$, and sequences ordered by timestamp ($s_T$).

by id (Figure 4.5). In fact, time-aware sequences obtain slightly worse results. A possible explanation for this behavior is that we are performing a 5-cross validation for this dataset, so the splits are independent of time. Furthermore, building time-based sequences of user preferences is not a trivial question, mostly because most of the available datasets – as in this case – 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 [11].

To address this issue, we experiment with a different dataset where interactions (and their timestamps) are realistic. Figure 4.8 shows the results for the MovieTweetings dataset. In the left side of the figure we present the results when sequences are ordered based on time (using $s_T$) and in the right side the ones ordered by item id ($s_i$ is used). In this case, we performed a global temporal split, where 80% of the oldest ratings are put into the training set and the rest into test. In this dataset we do not have any content-based information, so only pure collaborative-filtering recommenders were obtained. There are several aspects worth noting in these results. Firstly, we see that the performance in terms

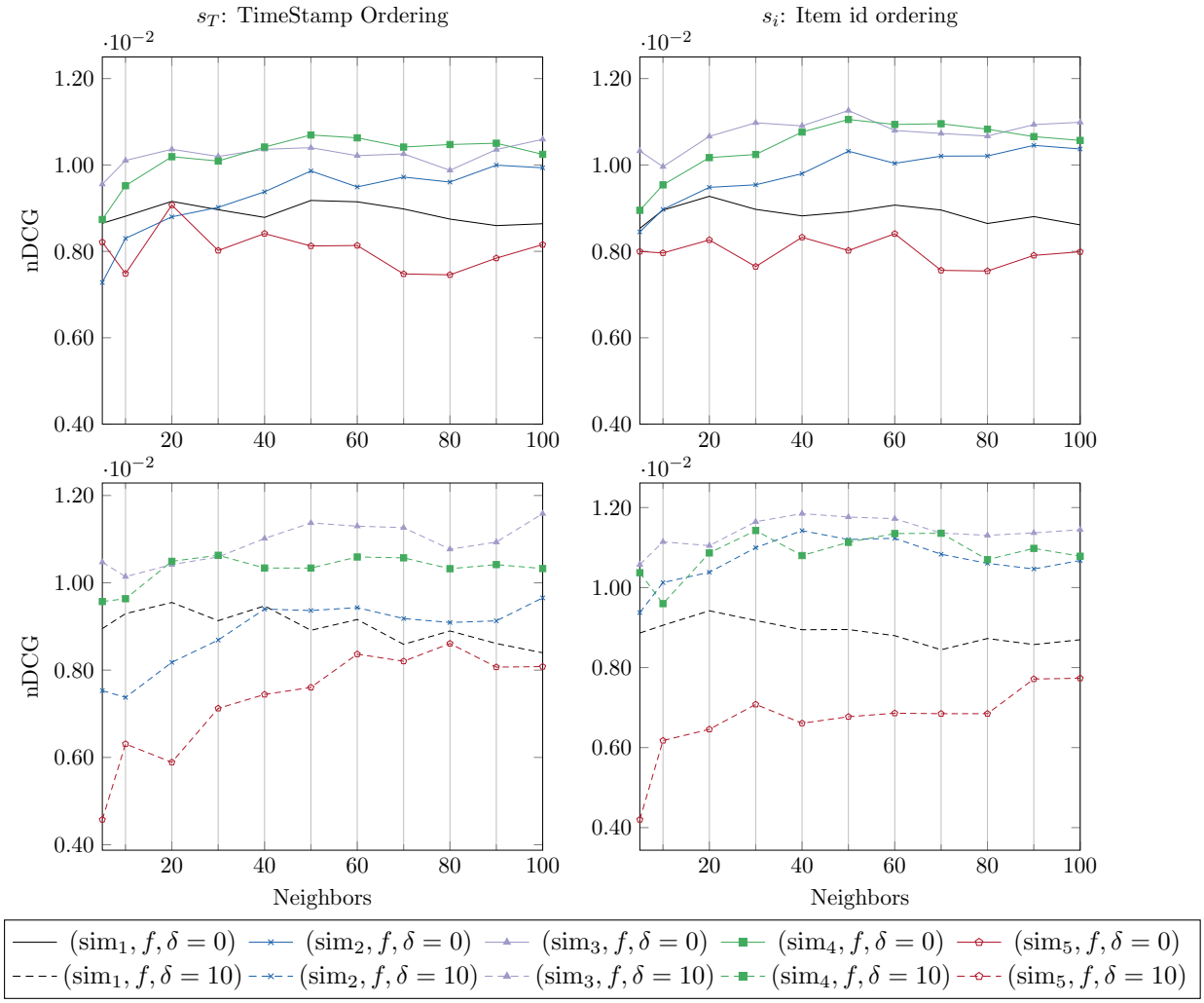


**Figure 4.8**: Results of LCS-based similarity for MovieTweetings dataset using a global temporal split. Different normalization functions using $\delta = 0$ and $\delta = 10$. Ordering by timestamp ($s_T$) and item id ($s_i$).

of nDCG@5 is much lower than in any of the other datasets analyzed. This is due to the sparsity of the data. We have 21 times the number of users with more than the double of items, having only half million of ratings (see Table 2.4). As a comparison, note that in MovielensHetRec we have 300,000 more ratings with less items and users, producing a more dense user-item rating matrix.

In such context, it is very difficult to produce recommendations of items appearing in the test set. Furthermore, we can also observe that the plots of nDCG@5 vs neighbor show a less clear trend. In the previous results, a larger neighborhood would generally lead to a performance improvement. Here, no configuration is stable or monotonically increasing or decreasing, making more difficult to select an optimal value for the number of neighbors. This might be produced because there are some neighbors that have the same similarity than others, so the neighbor selection in these cases is arbitrary.

In any case, we observe, as in previous experiments, that normalization functions obtain higher results than basic configurations, although in this case, the optimal normalization function is $\text{sim}_3$, followed, in some cases, by the optimal one in the other dataset, $\text{sim}_2$. We

**Table 4.2**: Performance of some of the most representative configurations of the proposed approach in MovielensHetRec dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, $\alpha$-nDCG, EILD, and Gini) at cutoff 5. The configuration for each recommender is denoted as $(\text{sim}, f, \delta, \tau, \gamma)$, that is: normalization function, transformation function, threshold for $\delta$-matching, confidence filter, preference filter. The neighborhood size in every case is $k = 100$.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| $(\text{sim}_1, f_{ir}, 0, 0, 0)$ | 0.196 | 0.130 | 0.135 | 0.086 | 0.495 | 0.736 | 10.10% | 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 | 9.83% | 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** | 9.95% | 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 | 9.79% | 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 | 10.30% | 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 | 10.00% | 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 | 35.62% | 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 | **35.98%** | 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 | 12.61% | 0.170 | 0.691 | 0.003 |
| $(\text{sim}_2, f_{dr}, 10, 0, 0)$ | 0.232 | 0.152 | **0.159** | **0.105** | 0.484 | 0.725 | 12.70% | **0.175** | 0.683 | 0.003 |
| $(\text{sim}_1, f_{ir}, 10, 70, \bar{u})$ | 0.219 | 0.172 | 0.114 | 0.075 | 0.288 | 0.370 | 29.75% | 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 | 28.77% | 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 | 29.41% | 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 | 25.88% | 0.150 | 0.538 | 0.005 |

can conclude, therefore, that normalization functions are still useful even in very sparse datasets, and that the optimal configuration of the proposed approaches is a process that strongly depends on dataset characteristics.

## 4.5 Impact on beyond-accuracy metrics

As pointed out in the state of the art (Section 2.3), it is interesting to consider other dimensions of evaluation like novelty and diversity [42]. A trivial example of this tradeoff is the behavior of the Popularity recommender, that suggests the items that have been rated by more users, dealing to unpersonalized recommendations. Although this recommender usually shows a relatively high effectiveness in terms of ranking evaluation, these recommendations obviously lack both diversity and novelty.

In Table 4.2 we show some of the most representative configurations of the LCS-based similarity for the MovielensHetRec dataset that have been analyzed in the previous sections. Together with novelty (EPC and EPD) and diversity (AD as a percentage, $\alpha$-nDCG, EILD, and Gini) metrics, we also show results for other ranking evaluation metrics (nDCG, Precision, Recall, and MAP). We must take into account that these metrics are obtained from the RankSys framework so some of them do not correspond exactly to the definitions shown in Section 2.3.3. For instance, the Gini metric in this framework is the complementary of the real Gini, and the Aggregate diversity is computed in a different way, whose results do not match its definition when coverage is not complete; because of this, we report our own implementation of this metric. The most remarkable aspect about these results is that it is hard to find a configuration that optimizes accuracy, novelty, and diversity at the same time.

Firstly, we observe that the confidence filter $\tau$ affects negatively the novelty of the recommendations. This may be expected as users with more elements in common with the target user are preferred instead of other users with a lower similarity, leaving aside some items that could be more novel. At the same time, this parameter seems to encourage more diverse recommendations. This is evidenced by the highest value achieved in AD and Gini for a configuration using $\tau = 50$, especially when compared against the same configuration

**Table 4.3**: Performance of some of the most representative configurations of the proposed approach in MovielensHetRec dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, $\alpha$-nDCG, EILD, and Gini) at cutoff 5. The configuration for each recommender is denoted as $(\text{sim}, f, \delta)$ using $s_T$ (ordering sequences by timestamp), that is: normalization function, transformation function, and threshold for $\delta$-matching. The neighborhood size in every case is $k = 100$.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| $(\text{sim}_1, f_{dr}, 0)$ | 0.177 | 0.120 | 0.120 | 0.074 | 0.500 | **0.742** | 9.37% | 0.130 | 0.702 | 0.003 |
| $(\text{sim}_2, f_{dr}, 0)$ | 0.212 | 0.137 | 0.144 | 0.096 | 0.468 | 0.732 | 12.62% | 0.160 | 0.691 | 0.002 |
| $(\text{sim}_1, f_{dr}, 10)$ | 0.172 | 0.117 | 0.117 | 0.072 | **0.506** | 0.741 | 9.19% | 0.126 | **0.702** | 0.003 |
| $(\text{sim}_2, f_{dr}, 10)$ | **0.213** | 0.137 | **0.145** | **0.096** | 0.470 | 0.729 | 12.86% | 0.161 | 0.691 | 0.003 |
| $(\text{sim}_1, f_{ir}, 0)$ | 0.193 | 0.129 | 0.133 | 0.084 | 0.485 | 0.738 | 10.21% | 0.144 | 0.695 | **0.003** |
| $(\text{sim}_2, f_{ir}, 0)$ | 0.212 | **0.137** | 0.145 | 0.096 | 0.466 | 0.732 | 13.54% | **0.161** | 0.699 | 0.002 |
| $(\text{sim}_1, f_{ir}, 10)$ | 0.189 | 0.126 | 0.131 | 0.082 | 0.489 | 0.737 | 10.20% | 0.142 | 0.693 | 0.003 |
| $(\text{sim}_2, f_{ir}, 10)$ | 0.211 | 0.136 | 0.144 | 0.096 | 0.469 | 0.729 | **13.76%** | 0.160 | 0.697 | 0.003 |

using $\tau = 0$, where AD is only 10.10% and Gini has a value of 0.003.

Secondly, in the previous results of nDCG we discussed that the preference filter does not affect significantly the quality of recommendations. For novelty and diversity, the same behavior is observed. This is not strictly a negative aspect, as making use of this parameter reduces the computation time as the sequences generated are shorter and, hence, more efficient.

Another important aspect to consider is that normalization functions, although producing an improvement over all accuracy metrics, they are not affecting in the same way the novelty and diversity metrics. There are hardly any changes between the results with and without normalization. This is a very interesting aspect, as we can conclude that by changing the normalization function we can improve the accuracy of the recommendations without altering the observed novelty and diversity dimension of the recommended items.

In Table 4.3 we show the results for some representative configurations of the LCS-based similarity for the MovielensHetRec dataset where the items are ordered by timestamp, omitting the variations with the confidence and preference parameters. As in the previous results, we see that there is not a single configuration that optimizes all metrics at the same time, but we observe that in both cases, with timestamp order and item id order, the best results in ranking evaluation metrics are obtained when using the $\text{sim}_2$ normalization function. However, this is not the case for novelty and diversity, as there are some configurations using the basic approach that achieve a better performance. Furthermore, if we compare Tables 4.2 and 4.3, we see that, in general, the ordering by id performs better that its equivalent time-ordered recommender, mainly in ranking evaluation metrics.

Last but not least, Table 4.4 presents the results for the MovieTweetings dataset. Here, we observe again that the normalizations obtain better results that the equivalent basic recommenders. This confirms that the normalizations are a must have in order to improve the results. However, the timestamp ordering does not produce better results in comparison with the basic order; this is somewhat unexpected: on the one hand, the timestamps in this case are real and the split is temporal, so we would expect the results to show some improvement in performance, since it should favor time-aware recommenders, but, on the other hand, this is the most sparse dataset of the three analyzed. From $45,324$ users, $20,340$ have rated only one item. Those users are in fact not relevant, neither as neighbors nor as target users for recommendations, as their ratings are not mature. Furthermore, only $13,098$ users have more than five ratings, so these results should be taken carefully. Besides, the ratings are in a scale from 1 to 10, so allowing here only a difference of 1 in

**Table 4.4**: Performance LCS-based recommenders in MovieTweetings dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD) and diversity (AD, $\alpha$-nDCG, EILD and Gini). The neighborhood size in every case is k=100. Recommenders labeled with $s_i$ and $s_T$ generated their user sequences by ordering the items either by item id or timestamp, respectively.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| $(\text{sim}_1, f_{ir}, 0, s_i)$ | 0.009 | 0.013 | 0.005 | 0.003 | 0.472 | 0.349 | 24.52% | 0.007 | **0.341** | 0.001 |
| $(\text{sim}_2, f_{ir}, 0, s_i)$ | 0.010 | 0.015 | **0.007** | 0.004 | 0.464 | 0.335 | **43.75%** | 0.007 | 0.316 | 0.009 |
| $(\text{sim}_1, f_{ir}, 10, s_i)$ | 0.009 | 0.013 | 0.005 | 0.003 | 0.474 | 0.348 | 20.37% | 0.007 | 0.341 | 0.001 |
| $(\text{sim}_2, f_{ir}, 10, s_i)$ | **0.011** | **0.016** | 0.007 | 0.004 | 0.462 | 0.332 | 40.02% | **0.008** | 0.310 | 0.008 |
| $(\text{sim}_1, f_{ir}, 0, s_T)$ | 0.009 | 0.013 | 0.005 | 0.003 | 0.472 | **0.349** | 24.43% | 0.007 | 0.341 | 0.001 |
| $(\text{sim}_2, f_{ir}, 0, s_T)$ | 0.010 | 0.014 | 0.007 | **0.004** | 0.453 | 0.327 | 42.40% | 0.007 | 0.304 | **0.011** |
| $(\text{sim}_1, f_{ir}, 10, s_T)$ | 0.008 | 0.013 | 0.005 | 0.003 | **0.474** | 0.349 | 20.40% | 0.007 | 0.338 | 0.001 |
| $(\text{sim}_2, f_{ir}, 10, s_T)$ | 0.010 | 0.015 | 0.006 | 0.004 | 0.451 | 0.326 | 40.00% | 0.007 | 0.299 | 0.009 |

**Table 4.5**: Performance of baselines in MovielensHetRec dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, $\alpha$-nDCG, EILD, and Gini). The best configuration for LCS-based approach is also included for comparison.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| Pop | 0.160 | 0.105 | 0.112 | 0.069 | 0.444 | **0.741** | 7.66% | 0.123 | **0.700** | 0.002 |
| UB1 | 0.233 | 0.152 | 0.161 | 0.106 | 0.484 | 0.723 | 12.94% | 0.177 | 0.682 | 0.003 |
| UB2 | 0.235 | 0.153 | 0.161 | 0.107 | 0.490 | 0.722 | 12.94% | 0.177 | 0.678 | 0.004 |
| IB1 | 0.162 | 0.109 | 0.116 | 0.069 | 0.521 | 0.712 | 65.03% | 0.132 | 0.660 | 0.004 |
| IB2 | 0.179 | 0.119 | 0.126 | 0.077 | 0.508 | 0.710 | **67.04%** | 0.145 | 0.672 | 0.004 |
| PureCB | 0.010 | 0.007 | 0.010 | 0.005 | **0.853** | 0.739 | 53.69% | 0.028 | 0.658 | 0.020 |
| CBCF | 0.254 | 0.165 | 0.180 | 0.120 | 0.504 | 0.722 | 13.06% | 0.192 | 0.666 | 0.004 |
| MF | **0.271** | 0.176 | **0.200** | **0.133** | 0.635 | 0.694 | 36.50% | **0.207** | 0.626 | **0.025** |
| $(\text{sim}_2, f_{dr}, 10, 30, \bar{u})$ | 0.246 | **0.179** | 0.152 | 0.101 | 0.406 | 0.571 | 25.88% | 0.150 | 0.538 | 0.005 |

the rating may cause a loss of performance under certain situations. This can be observed in the results in all ranking evaluation metrics: they are in fact much lower than the ones obtained in the rest of the datasets; in fact, these results that are so close to zero evidences the difficulty of evaluating item rankings on very sparse datasets.

## 4.6 Performance comparison against other algorithms

We have seen in previous sections the performance of different configurations of some LCS recommenders. However, we should also test their performance against other state-of-the-art recommenders, to check its potential as a new similarity metric. Although the baselines will be mostly the same for all the datasets, their parameters (number of neighbors, number of factors, content-based information, ...) will be optimized to the specific dataset that we are analyzing. Firstly, we show the results of the baselines for the MovielensHetRec dataset (Section 4.6.1). Then, we show the results of the baselines for the MovieTweetings dataset, where we have included other baselines obtained from [22] to compare against specific sequential recommenders (Section 4.6.2). The results for baselines in the Lastfm dataset are presented in Appendix C. As mentioned before, the configuration of these baselines is presented in Table 4.1; these parameters were selected according to a preliminary test where the best configurations with respect to the nDCG@5 metric were chosen.

**Table 4.6**: Performance of baselines and some configurations of the proposed approach in Movielens 10M dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, $\alpha$-nDCG, EILD, and Gini).

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| Pop | 0.106 | 0.069 | 0.061 | 0.040 | 0.576 | 0.748 | 7.22% | 0.078 | **0.749** | 0.001 |
| IB2 | 0.175 | 0.120 | 0.121 | 0.071 | 0.749 | 0.718 | **53.55%** | 0.137 | 0.643 | 0.005 |
| UB2 | 0.268 | 0.172 | 0.179 | 0.124 | 0.711 | 0.715 | 23.69% | 0.200 | 0.662 | 0.005 |
| MF | 0.280 | 0.182 | **0.200** | **0.133** | 0.803 | 0.697 | 28.67% | **0.210** | 0.641 | **0.022** |
| PureCB | 0.012 | 0.008 | 0.010 | 0.006 | **0.912** | 0.738 | 53.48% | 0.030 | 0.659 | 0.020 |
| CBCF | 0.140 | 0.098 | 0.080 | 0.052 | 0.640 | **0.750** | 28.99% | 0.097 | 0.705 | 0.003 |
| $(\text{sim}_1, f_{ir}, 0, 0, 0)$ | 0.228 | 0.151 | 0.153 | 0.100 | 0.686 | 0.736 | 12.01% | 0.164 | 0.680 | 0.003 |
| $(\text{sim}_1, f_{dr}, 0, 0, 0)$ | 0.224 | 0.148 | 0.149 | 0.098 | 0.685 | 0.738 | 11.77% | 0.159 | 0.684 | 0.003 |
| $(\text{sim}_2, f_{dr}, 0, 0, 0)$ | 0.152 | 0.099 | 0.106 | 0.065 | 0.636 | 0.745 | 36.68% | 0.113 | 0.717 | 0.003 |
| $(\text{sim}_2, f_{dr}, 10, 30, \bar{u})$ | **0.284** | **0.199** | 0.177 | 0.124 | 0.587 | 0.593 | 22.10% | 0.179 | 0.551 | 0.005 |

### 4.6.1 Performance comparison in MovielensHetRec

Table 4.5 shows the results for each of the baselines and the best LCS-based recommender. Here, the content-based baseline (PureCB) obtains a much worse performance than the hybrid content-based and collaborative-filtering recommender (CBCF), which, surprisingly, obtains the best results in ranking evaluation right after the MF baseline (the best performing recommendation technique in this case). According to these results, content-based techniques produce more novel recommendations (and hence, less popular), as represented by the higher values of EPC. However, it is not easy to produce novel and accurate recommendations: for instance, a random recommender would also obtain very high values of novelty and diversity; this is probably the reason that the PureCB recommender obtains values so high of EPC.

In terms of diversity, MF outperforms the other techniques in terms of Gini, whereas IB is the best recommender according to AD. The Popularity recommender achieves the highest value of EILD, and the CBCF and UB recommenders obtain the second highest values of $\alpha$-nDCG, just behind the MF recommender. As mentioned before, this metric 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.

However, one of the main objectives of this work is to outperform some of the state-of-the-art baselines, especially those based on neighbors. Recalling the results shown in Table 4.2 and comparing with those of Table 4.5, we find that all configurations of the proposed LCS-based user similarity outperforms some of the state-of-the-art baselines, namely, Pop, IB, and PureCB. Although there are no configurations that outperform the MF baseline, for some preliminary test, we were able to beat other MF methods, like the pLSA recommender. Nonetheless, it is not difficult to find some configurations that outperform the UB recommender, which is the real baseline to beat, since the proposed approach uses the same recommendation technique as UB but changing the user similarity metric. More specifically, the best LCS-based configuration outperforms UB in terms of nDCG, precision, AD, and Gini.

Furthermore, we should note that the extremely high results of the hybrid CBCF are strange. Because of this, we investigated on this issue and discovered that the MovielensHetRec dataset is biased to content-based information. According to the definition of the dataset in [12], the dataset only contains users who have provided both ratings and tags. For this reason, we decided to perform some additional experiments using the whole Movielens10M dataset – from which the MovielensHetRec dataset was extracted – and

**Table 4.7**: Performance of baselines in MovieTweetings dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD) and diversity (AD, $\alpha$-nDCG, Gini). The best configuration for LCS-based approach is also included for comparison.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| Fossil | 0.008 | 0.012 | 0.004 | 0.002 | 0.425 | 0.324 | 14.08% | 0.005 | 0.327 | 0.001 |
| MC | **0.013** | **0.021** | **0.008** | **0.004** | 0.428 | 0.323 | 26.61% | **0.011** | 0.312 | 0.002 |
| UB1 | 0.011 | 0.016 | 0.006 | 0.003 | 0.459 | 0.330 | 39.05% | 0.008 | 0.305 | 0.007 |
| UB2 | 0.010 | 0.015 | 0.006 | 0.003 | 0.463 | 0.333 | 35.07% | 0.008 | 0.311 | 0.007 |
| IB1 | 0.009 | 0.015 | 0.006 | 0.003 | 0.484 | 0.331 | **61.10%** | 0.007 | 0.299 | 0.015 |
| IB2 | 0.009 | 0.016 | 0.006 | 0.003 | 0.484 | 0.334 | 43.68% | 0.008 | 0.311 | **0.020** |
| Pop | 0.003 | 0.006 | 0.003 | 0.001 | 0.938 | **0.393** | 1.84% | 0.006 | **0.751** | 0.000 |
| MF | 0.006 | 0.009 | 0.004 | 0.002 | **0.986** | 0.329 | 11.76% | 0.009 | 0.517 | 0.003 |
| $(\text{sim}_2, f_{ir}, 10, s_i)$ | 0.011 | 0.016 | 0.007 | 0.004 | 0.462 | 0.332 | 40.02% | 0.008 | 0.310 | 0.008 |

filtering out the items not included in the MovielensHetRec dataset, to make the comparisons between datasets more fair. The original Movielens10M dataset includes many more ratings than the MovielensHetRec dataset, and hence it should favor collaborative-filtering algorithms. For this experiment, we only report results for one fold, where an $80-20$ training/test split was performed for the users included in the MovielensHetRec dataset, and all the ratings from the remaining users are considered for training.

The results obtained for these experiments are presented in Table 4.6. Here, the performance of the baseline recommenders are different: MF, UB, and IB algorithms outperform the hybrid baseline CBCF, which no longer benefits from having content-based information as much as before. More specifically, we have observed that the number of ratings per item has increased significantly (from 84.6 to 963.5), which clearly favors collaborative-filtering algorithms. Moreover, since the number of users has also changed (from $2,113$ to $69,878$) we hypothesize that the user profile building step (function $UserProfile(u)$ in Section 2.2.2) is now more noisy, producing neighbors of worse quality for CBCF than for UB, evidenced by the worse results obtained for the former, even though the score prediction is computed in the same way once the neighbors are found. Furthermore, we have also included in this table the two most basic configurations of the LCS similarity approach and the best combination (with respect to nDCG) of parameters in MovielensHetRec. In this context, LCS recommenders are also able to beat the hybrid recommender by a large margin, and even the best baseline method (MF) in terms of nDCG and Precision.

In summary, the proposed approaches based on LCS similarity in MovielensHetRec have shown better performance in terms of nDCG and precision than some of the baselines, especially than the UB recommender, evidencing the advantage of using a similarity based on LCS, since this is the only difference with respect to how these two recommenders are being computed.

### 4.6.2 Performance comparison in MovieTweetings dataset

As mentioned before, in this dataset we use some baselines from [22] besides the more classical ones used before. The reason behind this is that, since we are using a temporal split, we think we should also extend the experiments to compare against other baselines that take into account the temporal information. As before, the parameters of these baselines can be seen in Table 4.1.

According to the results presented in Table 4.7, we observe that the LCS-based recommender performs again quite well, although it is not the best one. It performs better than any of the baselines that do not take into account the time component. Besides, it also outperforms the Fossil recommender, the one that obtained the best results in [22]. The

most striking results in this case are those obtained by the Fossil recommender, which was the best recommender in [22]. However, it is relevant to indicate that we have just used the basic configuration (L=1, K=10 and bias=100) so it may be necessary to optimize those parameters, for instance, by performing a grid search. Furthermore, regarding this and the MC baseline, the authors only considered in the original article the users and items that have at least 5 interactions, while here we have not performed any pre-processing step, finally, in that article, the split was made by selecting the last item consumed by every user and including it in the test set, so the objective was to predict such item. Here, the split is performed globally (20% of newer ratings to the test set), so there may be users with more than one item in test and users that do not have any item in it. Nevertheless, with these results, we again confirm the difficulty of creating a recommender that obtains good performance on all the metrics, and, at the same time, we show promising results of our proposed LCS-based approaches.

## 4.7 Discussion

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 taking advantage of both collaborative-filtering and content-based information. Although in some cases the LCS-based configurations obtain poor results (as in the case of generating genres sequences), the results obtained when using directors or the pure collaborative-filtering approach are comparable to, and sometimes even better than, other state-of-the-art recommenders that are used in the area. However, we must take into account that these experiments are still conditioned to the experimental setup (we have only considered the top 5 items in the ranking with a rating $\geq 5$).

We have also shown the effect on recommendations of some other configurable parameters: the *confidence filter*, the *preference filter* and *normalization functions*. Among all, the one that has produced substantial improvements in terms of ranking quality is the normalization, especially the ones that use the length of both sequences, i.e., the second normalization function $\text{sim}_2$ and the third normalization function $\text{sim}_3$. While $\text{sim}_2$ returns the square of the LCS divided by the product of the lengths of both sequences, $\text{sim}_3$ doubles the LCS and divides this value by the sum of the length of both sequences. The worst normalization in all cases has been $\text{sim}_5$, the one that divided by the minimum sequence length of the two users involved. Hence, we can conclude that when bounding a user similarity, we must take into account the information provided by both users.

Moreover, the best configurations are normally achieved when using different combinations of preference, confidence, and normalization functions. In these cases, we were able to find a combination that could outperform UB baselines in terms of ranking evaluation and, sometimes, in case of novelty and diversity metrics, as for AD. At the same time, we found that for both MovielensHetRec and MovieTweetings the sequences generated by making use of the timestamps did not provide an advantage over the sequences ordered by natural ordering. However, we think that there is still some room for improvement if we rethink the classical neighborhood approaches by prioritizing the items that are near the last common item of both users.

Nevertheless, it is relevant to say that the LCS-based recommenders are not the best approaches reported as we could not find a configuration that beats the MF recommender [25] (the best baseline in two of the datasets). This is expected as it is well-known in the literature that these kind of recommenders tend to outperform neighborhood based

approaches. However, there are many MF methods that can be used in order to make recommendations. Here, we have shown the results obtained by the one described in [25], but, as we reported in [7], some configurations of LCS-based recommenders are able to outperform other MF techniques such as pLSA in the MovielensHetRec dataset. At the same time, we have seen that in the MovieTweetings dataset, the UB approaches obtain better results than the MF approaches, although in this case, the results of all recommenders are very low in comparison with the other two datasets.

Finally, an important aspect to take into account is that there is not a single recommender – not LCS-based nor baselines – that outperforms the rest in all the metrics. In general, it is very difficult to build recommenders that obtain good results in both ranking quality metrics and novelty and diversity metrics. In fact, sometimes when using metrics that measure the same dimension like Aggregate Diversity and EILD in the case of diversity, we can end up with recommenders with high values in one of them and very low values in the other. This was observed, for instance, with IB recommenders, which obtain very good results in terms of the AD metric but worse performance for EILD.

# Chapter 5

# Conclusions

## 5.1 Summary and discussion

Recommender Systems are essential in a large number of applications and contexts, especially those related to the Internet. Because of this, it is compulsory to continue innovating in this field by proposing new approaches and algorithms to improve the user's experience, since good recommendations will usually make current users trust the system and, potentially, may increase the number of customers in the future.

In this work, we have made a study on the most important techniques in this field, showing not only the best known algorithms and recommendation approaches but also different ways of evaluating them depending on the aspect that we want to consider (ranking quality vs novelty and diversity). We have concluded that it is generally necessary to find a balance between them, as sometimes better ranking quality results in a poor performance on both novelty and diversity, and vice versa. In addition, we have analyzed the effects of different state-of-the-art algorithms in three datasets from the real world, evidencing that sometimes it is difficult to produce effective recommendations, especially in very sparse datasets. We have also shown the challenge involved in obtaining a recommender that outperforms the rest in all possible metrics.

Nonetheless, the most important contribution of this work is the proposal, development, and evaluation of the Longest Common Subsequence (LCS) algorithm as a user similarity metric for recommendation. We have presented a general method to configure an LCS-based similarity metric to generate different sequences of user preferences by using pure collaborative-filtering and content-based data, together with different orderings. Furthermore, some additional parameters like the *$\delta$-matching*, *confidence filter*, *preference filter*, and *normalization functions* have also been integrated in this general model.

We have obtained substantial improvements in performance for specific configurations of the proposed model, not only in terms of ranking quality but also in novelty and diversity dimensions. The best results have been typically found when all these parameters are used together, however, strong performance improvements were obtained in some cases when only one of them was used, in particular, when different normalization functions were included in the model. Nonetheless, it should be noted that these parameters add another layer of complexity since they should be tuned to the specific dataset to obtain the optimal configuration.

With the purpose of providing empirical support for the proposed model and the different available configurations, we have used three different datasets under specific circumstances. As a result, with the MovielensHetRec dataset we have found that we slightly improve the results when using data from item contents (only when directors are used),

but, in general, our approaches beat most of the baselines, at least in terms of ranking quality. With the Lastfm dataset we have obtained similar results than with the previous dataset, although the user coverage might be severely decreased in some situations; it should be noted that the ratings exploited in this case were artificially transformed from implicit data. Finally, in the MovieTweetings dataset we performed a temporal split to check the behavior when time-aware sequences are incorporated to the model; here we found that a global ordering of the items (i.e., not based on the time the user interaction was produced) obtained better results in terms of ranking metrics that a sequence ordering based on timestamp. Since we believe this result is counter-intuitive, we aim to further extend these experiments with more datasets, focusing on those where the timestamps are realistic, which is not an easy constraint to satisfy.

In conclusion, we have shown the potential of the LCS algorithm to be used in the recommendation context as a similarity metric between users, besides some previous works where it was used as pattern finding algorithm [40, 26]. To the best of our knowledge, the application of this algorithm in this way is novel and, as described before, it shows empirically not only that it produces good recommendations but also that there is still a margin to improve its performance.

## 5.2 Contributions of this work

The specific contributions of the work presented here can be summarized as:

**Adaptation of Longest Common Subsequence** to recommender systems. Four parameters have been included in this new model, leading to several translations of this concept to be used as a user similarity.

**Two external contexts** were integrated successfully in the proposed model besides collaborative ratings: content-based data and temporal information. These extensions validate the generality of the approach.

**Experimental validation** of the proposed methods. Several configurations have been tested, analyzing the sensitivity with respect to their parameters and comparing the results against several state-of-the-art algorithms in terms of ranking quality, diversity, and novelty metrics. The results are particularly positive with respect to the classical user-based nearest neighbor algorithm, since the proposed approach is built on top of this baseline but the performance is larger when configured appropriately.

## 5.3 Future work

The research presented here uncover new problems and several directions for the continuation of this work. Among them, we highlight the following:

**Use of other features and more general datasets:** the model presented here might be further extended with more transformations so that other features (like demographic information, tags, etc.) can also be exploited and considered when building user sequences. Moreover, we think that parameters like *confidence filter* and *preference filter* can be adapted to implicit data, where no ratings are available.

**More transformation functions:** for these experiments in order to create the sequences we have used either collaborative-filtering or content-based information. We can split

the items in time intervals so all the movies that are between two timestamps are arranged with the same symbol and then order the sequences by timestamps. With this approach we may find similarities with users that have the same time-evolution than the destination user. This is only an example of a possible transformation worthy of further investigation; we plan to continue working on other, more principled transformation functions, that may lead to even larger performance improvements.

**Obtaining $n$-subsequences of length $L$:** instead of computing the whole LCS between two users and use this value as a similarity, we may obtain different subsequences of length $L$ and use the number of these sequences as the similarity. For this approach, we will need to define the number of characters necessary to obtain a single sequence ($L$) and see how many subsequences of that length can be found in the users. An advantage of this approach would be identifying short common patterns of behavior between users. Besides, this may help us to make recommendations based on the gaps that we can find between the sequences.

**Other variations of LCS:** in all the experiments performed, we have considered that two "characters" are the same when the ratings of both users are in the range of the threshold parameter $\delta$. However, we can reformulate the LCS algorithm so it directly considers the differences of ratings; in such a case, the computed LCS should be incremented differently depending on whether both users have rated equally the same item or not. We may even consider generating negative values when the ratings are in opposite extremes of the ratings range. As another extension of LCS, we could weight differently each time the algorithm has to advance one position from one sequence to another (gap) or perform single-character modifications of the sequences to obtain a matching, this would relate to the well-known problem of sequence alignment, but reformulated in the recommendation context with users (or items) as sequences.

**Revisiting $k$-NN recommenders:** we believe that offline experiments using temporal splits are the most realistic ones, since they reproduce more faithfully the recommendation scenario. Under these constraints, we understand that a similarity between users that captures the sequence of actions each user made in the system should be valuable. Even though we have not obtained definitive results in the experiments reported so far, we have started experimenting with a reformulation of the $k$-NN problem for time-aware scenarios, and the results are very promising. Under this reformulation, each neighbor contributes a number of candidate items to the target user depending on their last common interaction; in order to do this, the LCS-based similarity defined here would allow us to easily find this last common interaction and, at the same time, would produce a similarity between the users, considering the interaction pattern of both users. In this case, we have obtained some preliminary results that shows the potential of this new approach.

# Bibliography


[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[2] Fabio Aiolli. Efficient top-N recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 273–280, New York, NY, USA, 2013. ACM.

[3] Alberto Apostolico. String editing and longest common subsequences. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 2. Linear Modeling: Background and Application*, pages 361–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[4] Marko Balabanović and Yoav Shoham. Fab: Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997.

[5] Alejandro Bellogín. *Recommender System Performance Evaluation and Prediction: An Information Retrieval Perspective*. PhD thesis, Universidad Autónoma de Madrid, 2012.

[6] Alejandro Bellogín, Pablo Castells, and Iván Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*, pages 333–336. ACM, 2011.

[7] Alejandro Bellogín and Pablo Sánchez. Collaborative filtering based on subsequence matching: A new approach. *Submitted to Information Sciences*, 2017.

[8] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[9] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[10] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[11] Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, 2014.

[12] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011). In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 387–388, New York, NY, USA, 2011. ACM.

[13] Pablo Castells, Neil J. Hurley, and Saúl Vargas. Novelty and diversity in recommender systems. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 881–918. Springer, 2015.

[14] Òscar Celma. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[15] Charles L. A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 659–666. ACM, 2008.

[16] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-N recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[17] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. Semantics-aware content-based recommender systems. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 119–159. Springer, 2015.

[18] F. T. de la Rosa, María Teresa Gómez López, and Rafael M. Gasca. Analysis and visualization of the DX community with information extracted from the web. In *Database and Expert Systems Applications, 16th International Conference, DEXA 2005, Copenhagen, Denmark, August 22-26, 2005, Proceedings*, volume 3588 of *Lecture Notes in Computer Science*, pages 726–735. Springer, 2005.

[19] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 107–144. Springer US, Boston, MA, 2011.

[20] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

[21] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10:2935–2962, 2009.

[22] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 191–200. IEEE, 2016.

[23] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*, pages 230–237. ACM, 1999.

[24] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115, 2004.

[25] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272. IEEE Computer Society, 2008.

[26] Mehrdad Jalali, Norwati Mustapha, Md Nasir Sulaiman, and Ali 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*, pages 302–307. IEEE Computer Society, 2008.

[27] Yehuda Koren and Robert M. Bell. Advances in collaborative filtering. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 77–118. Springer, 2015.

[28] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer Society*, 42(8):30–37, 2009.

[29] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, Boston, MA, 2011.

[30] Raymond J Mooney, Paul N Bennett, and Loriene Roy. Book Recommending Using Text Categorization with Extracted Information. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 70–74, Madison, WI, 1998.

[31] Netflix prize. `http://www.netflixprize.com/rules.html`. Accessed: 2017-01-20.

[32] Xia Ning and George Karypis. SLIM: sparse linear methods for top-N recommender systems. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 497–506. IEEE Computer Society, 2011.

[33] Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[34] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 811–820. ACM, 2010.

[35] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[36] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: Introduction and challenges. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender Systems Handbook*, pages 1–34. Springer, 2015.

[37] Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 129–136. ACM, 2014.

[38] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 285–295. ACM, 2001.

[39] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Human Factors in Computing Systems, CHI '95 Conference Proceedings, Denver, Colorado, USA, May 7-11, 1995.*, pages 210–217. ACM/Addison-Wesley, 1995.

[40] YS Sneha, G Mahadevan, and M Madhura 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*, volume 2, pages 223–226, April 2011.

[41] Harald Steck. Evaluation of recommendations: rating-prediction and ranking. In *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 213–220. ACM, 2013.

[42] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*, pages 109–116. ACM, 2011.

[43] Saúl Vargas and Pablo Castells. Improving sales diversity by recommending users to items. In *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, pages 145–152. ACM, 2014.

# Appendices

# Appendix A

# Implementation details

As the main objective of this work is to verify if the application of LCS algorithm can be useful to make recommendations, we have implemented a framework that allows to operate with different variations of LCS-based recommenders that have been presented and discussed throughout this document. Moreover, to increase reproducibility and make comparisons against other state-of-the-art algorithms easier, we used external libraries and integrated them into our framework. More specifically, we created "LCS4RecSys" (LCS for Recommender Systems[1]), a framework that allows to execute our LCS-based similarities in user-based recommenders from Mahout[2] and RankSys[3] libraries. The third external library we have used is RiVal[4], but it has only been used to obtain results of recommendation metrics in terms of ranking evaluation. Our project has been implemented in Java as all the libraries are programmed in that language. When integrating the LCS-based recommenders both in RankSys and Mahout, we have followed their corresponding class hierarchy. More details about these libraries and the metrics implemented in them are described in this appendix.

## A.1 Comparing RankSys and Mahout

Both frameworks RiVal and RankSys have been developed by people from the Information Retrieval Group of the EPS-UAM, whereas Mahout is an Apache Software Foundation project. Although some of the recommendation algorithms implemented in RankSys and Mahout are very similar, we discovered that the results obtained by them differs too much depending on the library we use, mostly due to intrinsic differences of implementation. Because of this, the results between these libraries could not be compared directly (a similar conclusion but with different libraries is reported in [37]).

As a consequence, running the same configuration of an LCS-based recomender under both libraries would produce different recommendations, with different performance results. Some specific details about the libraries have already been discussed: RankSys does not normalize the score predicted for a user towards an item. This means that, instead of working with a bounded value when the recommender predicts a rating, it works using Equation 2.16; hence, these recommendations are not suitable for error metrics as now the predicted score does not correspond to a rating. Mahout, on the other hand, normalizes the score and error metrics can be computed but, in our experience, it obtains worse results in

[1] `https://bitbucket.org/PabloSanchezP/lcs4recsys/`
[2] `https://mahout.apache.org/`
[3] `https://github.com/RankSys/RankSys`
[4] `https://github.com/recommenders/rival`

ranking evaluation than the other framework, in part because there are many scores with the same value. Additionally, Mahout does not allow recommendations coming from only one neighbor.

Nonetheless, the main reason for us to choose RankSys over Mahout was the better performance results obtained consistently in all the metrics when using RankSys algorithms. This, together with much higher efficiency in terms of computation offered by this library helped us to decide towards this library instead of Mahout.

## A.2 Evaluation using the libraries

As mentioned before, we have used two libraries to evaluate the performance of the recommender systems. With RiVal, we have obtained results in terms of Precision, Recall, nDCG and MAP. For the different novelty and diversity metrics, we used the implementations included in the RankSys library, except for Aggregate diversity. This metric, although defined in that library, does not follow exactly the definition presented in Equation 2.50. Because of this, we implemented our own version following the class hierarchy defined in RankSys in the same way as done for the recommenders.

Furthermore, the novelty and diversity metrics defined by RankSys make use of auxiliary models of relevance, discount, or distance between items in order to evaluate the recommenders. As these models can take different values, we decided to execute the metrics with the most simple configurations, leaving for future work extending these results by running more complex formulations of these metrics. Hence, we have not taken into account any ranking discount model (we used the "NoDiscountModel") and no relevance model ("NoRelevanceModel" was used). For the metrics requiring a distance metric to compute distances over the items, we have made use of the cosine distance (instantiated as "CosineFeatureItemDistanceModel").

# Appendix B

# Performance of LCS using genres

In this appendix we extend the results of the MovielensHetRec dataset presented in Chapter 4, by making use of the genres information. As mentioned previously, the performance for the hybrid LCS recommender when using this information tends to be lower than the rest of approaches. However, it is interesting to observe the effect of the rest of the parameters when using this information. We use the same notation and experiment with the same configurations as the ones presented in Chapter 4, that is: $f = f_{ir}$, $f = f_{dr}$, and $f = f_{gr}$, which represent user sequences generated considering the item id, the item directors, and the item genres, respectively, also using ratings in the three cases.

Figures B.1, B.2, and B.3 show the results of three different configurations to generate sequences: only using the confidence filter, only using the preference filter, or using a combination of both. In the three cases we observe that, when using genres, the results obtained by the nDCG metric are in fact much lower than the pure collaborative-filtering and the directors hybrid recommender. At first, it is reasonable to think that the use of content-based information is a poor strategy, but as we showed with the performance of the rest of the baselines, it depends on how it is used (CBCF was one of the best baselines for this dataset, see Table 4.5). In this case, we are generating sequences of genres; however, two movies can be very different even if they have the same genres. Consider for example the movies *Godzilla*, *I am legend*, and *Alien*, they have exactly the same genres (in the MovielensHetRec dataset) but they are clearly very different from each other.

Furthermore, consider again the effects of normalization functions in Figures B.4 and B.5. Here, we show the effect of the normalization functions and their combination with preference and confidence filters. In this case, we see that even if the results obtained are slightly worse than the other two approaches, this approach can still be competitive against them. This is a very interesting result, because we may use this information to generate sequences for the users that have rated very few items. Finally, in Figure B.6 we present the results of the recommenders that generate sequences ordered by timestamp. In this case, we note that our approach based on genres, despite having a lower performance, can still be competitive when using certain normalization functions.

Finally, in Table B.1 we show the same results as in Table 4.2 but with some illustrative configurations using sequences generated by movie genres. As we can see here, the genres approach always obtain the worst performance if we compare it against the rest of equivalent configurations using directors and ids. However, in terms of novelty and diversity metrics, this situation is not found. Even if using genres reduces the AD, these approaches obtain relatively high results in EILD, showing that even if they recommend the same items to many users, they are different from the rest of recommended items. In fact, for both EILD and EPD, one of the genres configurations obtain the best results.

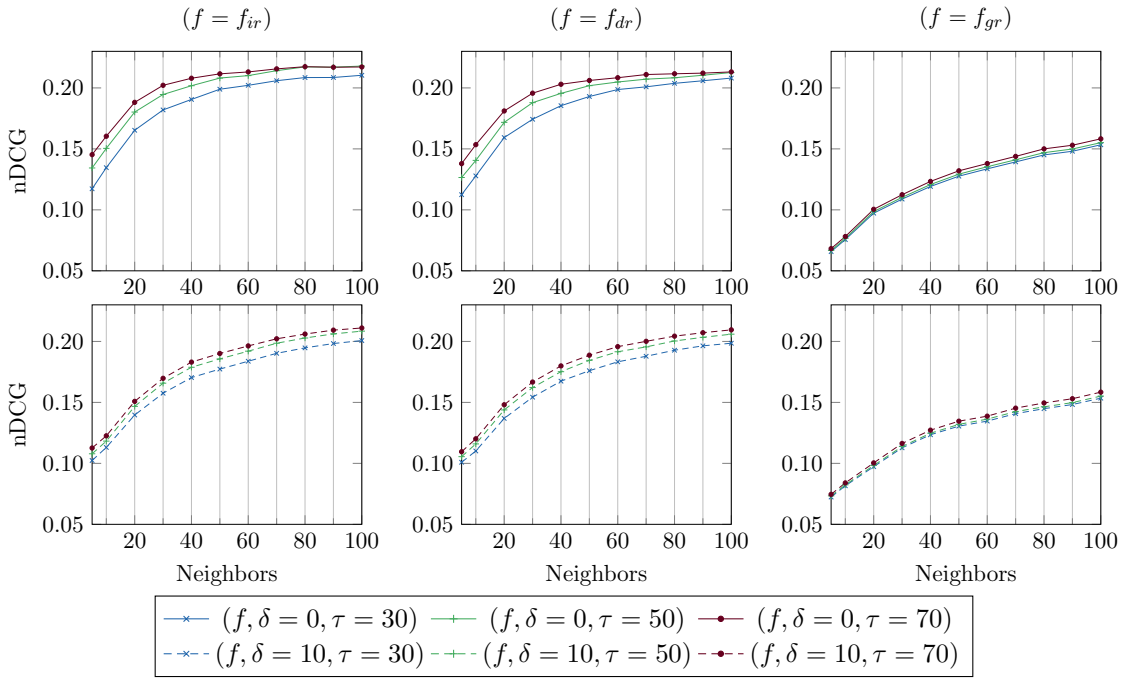


**Figure B.1**: Results including genres of LCS-based similarity for MovielensHetRec dataset. Different values of confidence filter parameter $\tau$ using $\delta = 0$ and $\delta = 10$.

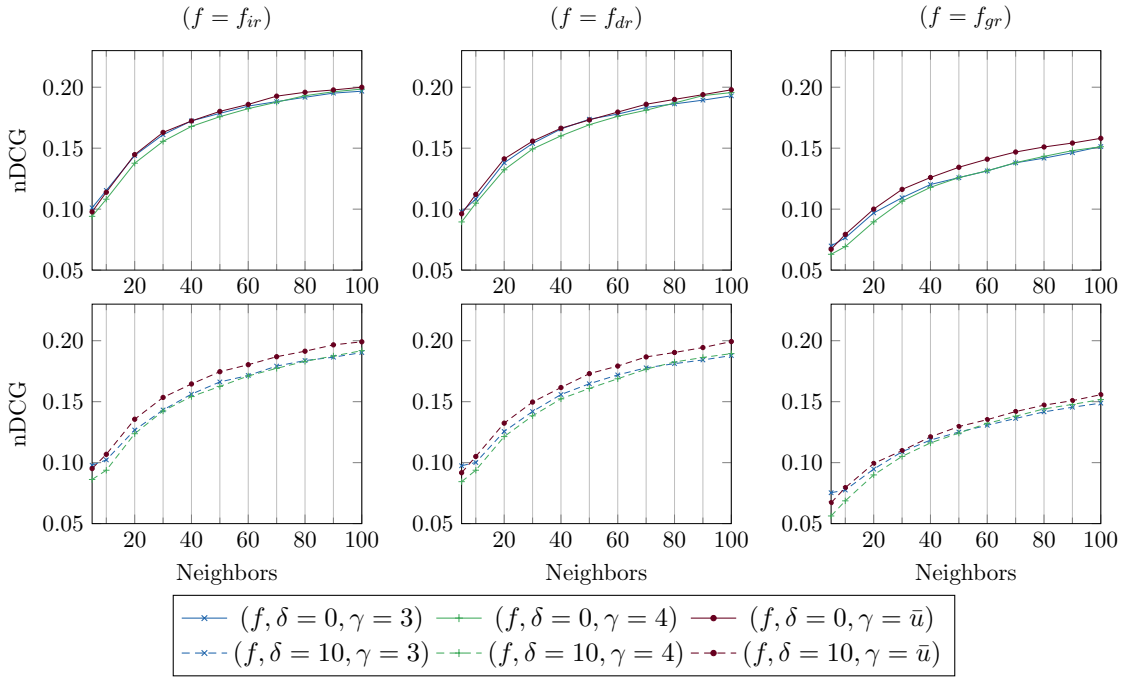


**Figure B.2**: Results including genres of LCS-based similarity for MovielensHetRec dataset. Different values of preference filter parameter using $\delta = 0$ and $\delta = 10$.

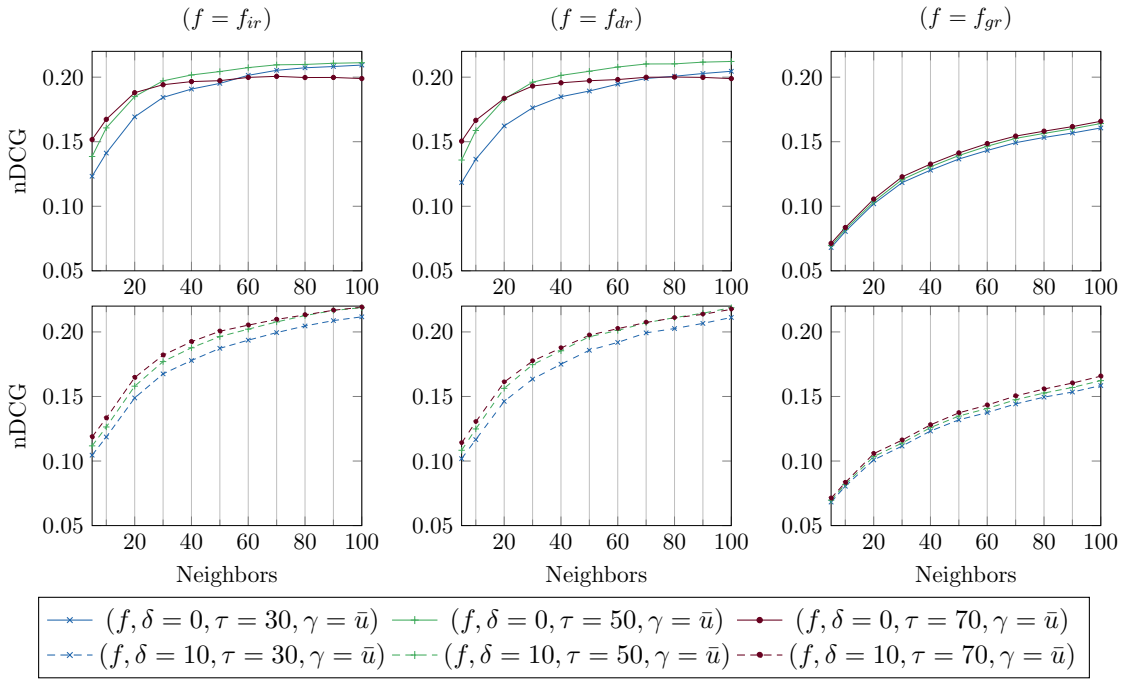


**Figure B.3**: Results including genres of LCS-based similarity for MovielensHetRec dataset. Combination of different values of preference and confidence filtering using $\delta = 0$ and $\delta = 10$.

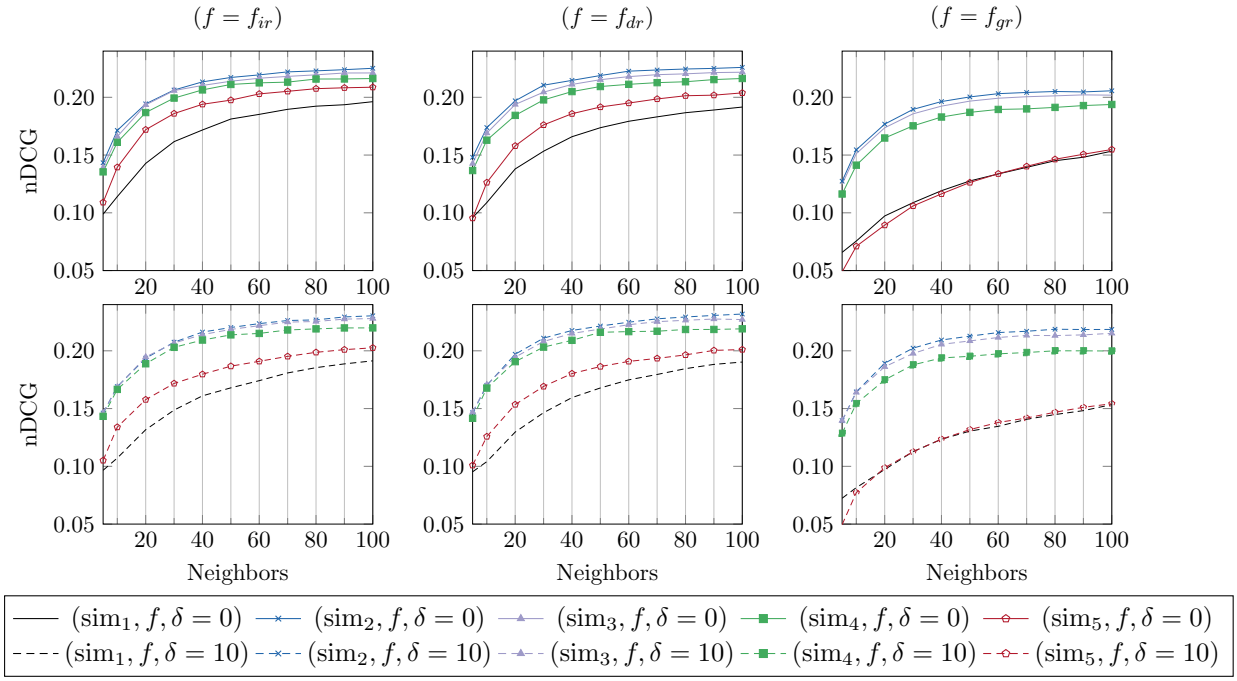


**Figure B.4**: Results including genres of LCS-based similarity for MovielensHetRec dataset. Different normalization functions using $\delta = 0$ and $\delta = 10$.

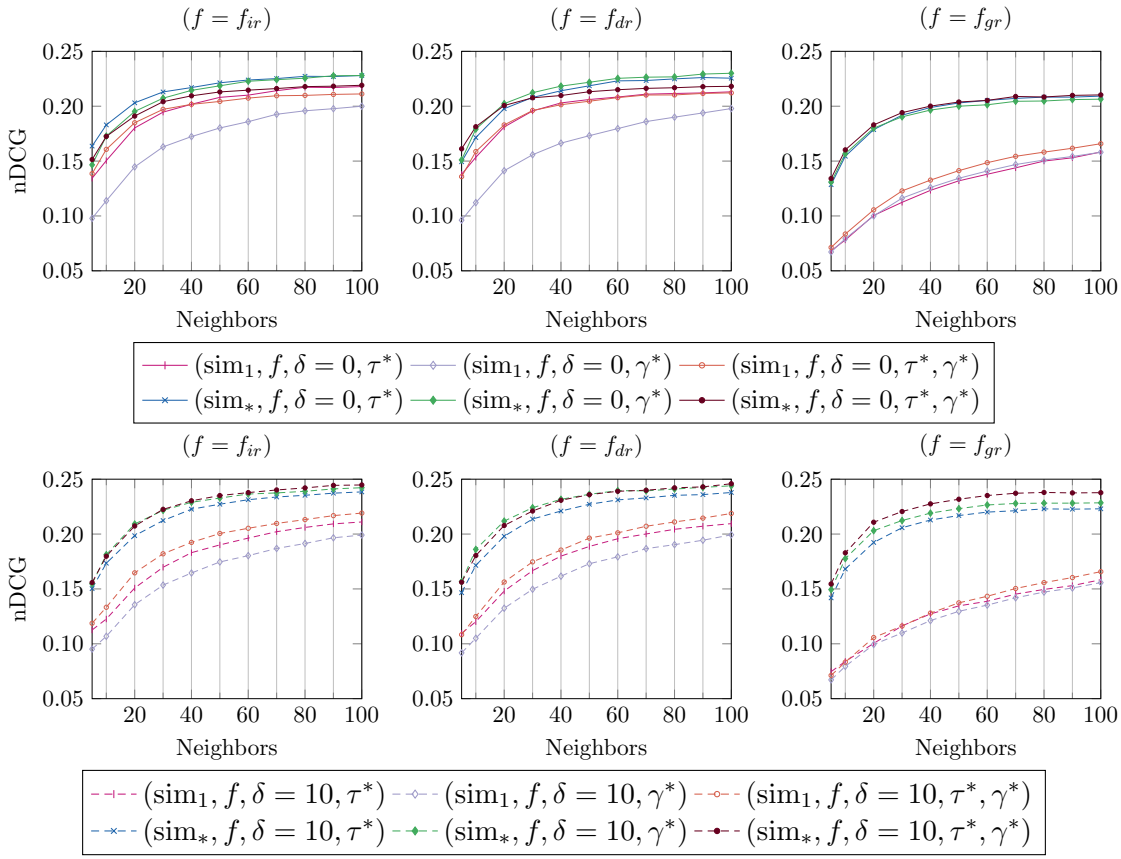


**Figure B.5**: Results including genres with the best confidence and preference filters and normalizations for MovielensHetRec dataset. Top row shows results using $\delta = 0$, bottom row using $\delta = 10$. The $*$ symbol denotes the best value among the previously reported ones is being used in the combination.

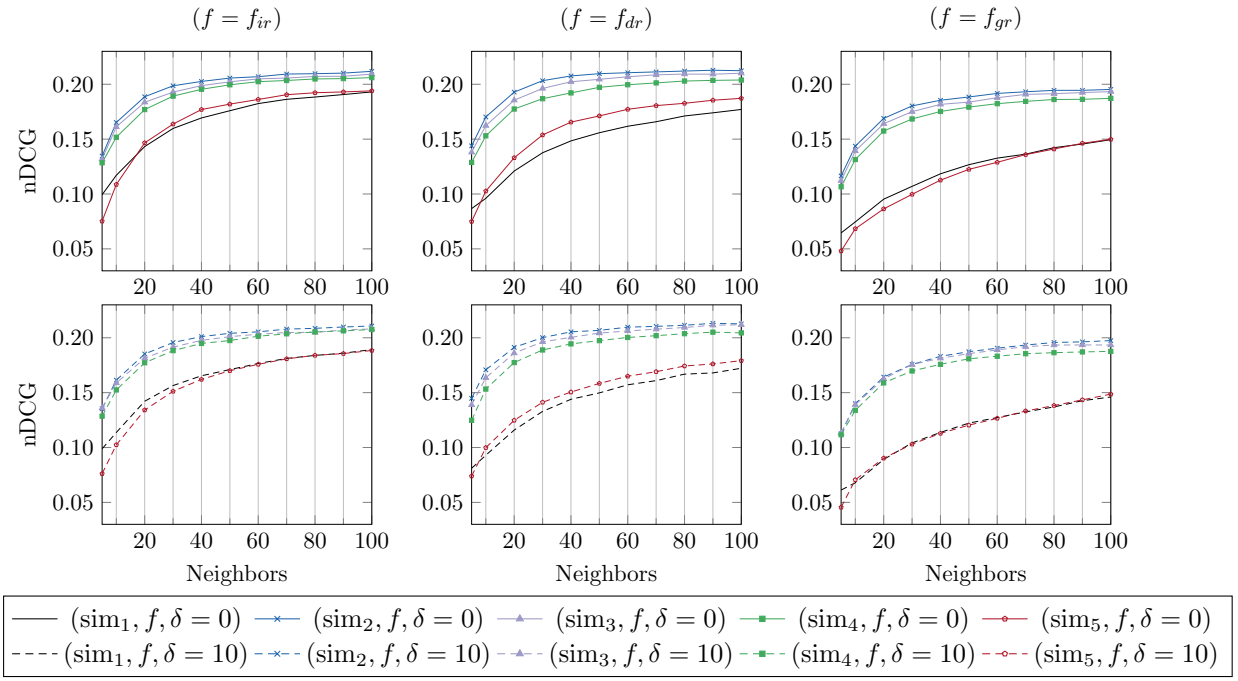


**Figure B.6**: Results including genres of LCS-based similarity for MovielensHetRec dataset. Different normalization functions using $\delta = 0$ and $\delta = 10$, sequence ordered by timestamp $(s_T)$.

**Table B.1**: Performance of some of the most representative configurations of the proposed approach in MovielensHetRec dataset including genres in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD), and diversity (AD, $\alpha$-nDCG, EILD, and Gini) at cutoff 5. The configuration for each recommender is denoted as $(\text{sim}, f, \delta, \tau, \gamma)$, that is: normalization function, transformation function, threshold for $\delta$-matching, confidence filter, preference filter. The neighborhood size in every case is $k = 100$.

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| $(\text{sim}_1, f_{ir}, 0, 0, 0)$ | 0.196 | 0.130 | 0.135 | 0.086 | 0.495 | 0.736 | 10.10% | 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 | 9.83% | 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 | 9.95% | 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 | 9.79% | 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 | 10.30% | 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 | 10.00% | 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 | 35.62% | 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 | **35.98%** | 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 | 12.61% | 0.170 | 0.691 | 0.003 |
| $(\text{sim}_2, f_{dr}, 10, 0, 0)$ | 0.232 | 0.152 | **0.159** | **0.105** | 0.484 | 0.725 | 12.70% | **0.175** | 0.683 | 0.003 |
| $(\text{sim}_1, f_{ir}, 10, 70, \bar{u})$ | 0.219 | 0.172 | 0.114 | 0.075 | 0.288 | 0.370 | 29.75% | 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 | 28.77% | 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 | 29.41% | 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 | 25.88% | 0.150 | 0.538 | 0.005 |
| $(\text{sim}_1, f_{gr}, 0, 0, 0)$ | 0.153 | 0.105 | 0.104 | 0.062 | 0.503 | 0.747 | 8.36% | 0.112 | 0.699 | 0.002 |
| $(\text{sim}_1, f_{gr}, 10, 0, 0)$ | 0.153 | 0.107 | 0.103 | 0.061 | 0.505 | **0.748** | 8.20% | 0.112 | 0.704 | 0.002 |
| $(\text{sim}_1, f_{gr}, 0, 0, \bar{u})$ | 0.158 | 0.108 | 0.107 | 0.064 | 0.495 | 0.743 | 8.78% | 0.116 | 0.701 | 0.002 |
| $(\text{sim}_1, f_{gr}, 10, 0, \bar{u})$ | 0.156 | 0.107 | 0.104 | 0.062 | 0.502 | 0.746 | 8.25% | 0.113 | **0.716** | 0.002 |
| $(\text{sim}_1, f_{gr}, 0, 50, 0)$ | 0.155 | 0.109 | 0.105 | 0.062 | 0.489 | 0.720 | 14.54% | 0.111 | 0.677 | 0.002 |
| $(\text{sim}_2, f_{gr}, 0, 0, 0)$ | 0.206 | 0.134 | 0.142 | 0.092 | 0.473 | 0.714 | 11.59% | 0.155 | 0.682 | 0.003 |
| $(\text{sim}_2, f_{gr}, 10, 0, 0)$ | 0.219 | 0.142 | 0.150 | 0.099 | 0.478 | 0.711 | 12.44% | 0.166 | 0.673 | 0.003 |
| $(\text{sim}_1, f_{gr}, 10, 30, \bar{u})$ | 0.158 | 0.111 | 0.105 | 0.063 | 0.485 | 0.718 | 8.23% | 0.113 | 0.689 | 0.002 |
| $(\text{sim}_2, f_{gr}, 10, 30, \bar{u})$ | 0.231 | 0.154 | 0.156 | 0.104 | 0.468 | 0.679 | 12.33% | 0.171 | 0.637 | 0.003 |

# Appendix C

# Performance results in Lastfm dataset

In this appendix we show the results obtained by the LCS-based recommenders in the Lastfm dataset. This dataset originally does not have explicit ratings but a number labeled as "weight" that indicates the number of times a user has listened to a specific music band or artist. Besides, although there is content-based information associated with the tags, we will only show the pure collaborative-filtering approach. It is also important to remember that for this dataset the explicit ratings have been obtained by transforming the original listenings into artificial ratings, so there are no half scales. However, to avoid confusion with the previous results, we still maintain the threshold value of 10 ($\delta = 10$) to indicate a discrepancy of $\pm 1$ in the ratings ($\delta = 1$).

## C.1 Results for Lastfm dataset

In Figure C.1 we show the results of the Lastfm dataset with two values of $\delta$. On the one hand, it is relevant the constant decline when using a large number of neighbors in both cases; on the other hand, we see that having a threshold of $\delta = 10$ produces better results in terms of nDCG than the same configuration with exact matchings, unlike in the MovielensHetRec dataset. We must recall that this parameter increments the number of potential neighbors that a user may have by allowing soft matchings. This behavior may be due to sparsity of the dataset, where less neighbors may produce worse recommendations. In fact, in the MovielensHetRec dataset, the average ratings per user was $\approx 405$ ($855,598$ ratings for $2,113$ users), and for Lastfm the average is $\approx 49$ ($92,834$ ratings for $1,892$ users); hence, in the first case it is easier to find neighbors that have rated in the same way some of the items. In Lastfm dataset, however, allowing only perfect matchings may find neighbors having only one or two items in common, which will eventually reduce the performance of the recommender. Thus, we conclude that allowing small discrepancies in ratings between users (soft matchings or $\delta > 0$) help to find neighbors with higher quality in sparse datasets.

We now show the results when applying the confidence parameter ($\tau$) in the Lastfm dataset in Figure C.2. In this case, the values for the confidences values are much lower than the ones used in the MovielensHetRec dataset (due to the sparsity, as discussed before). As we can see here, the best performance is obtained when using a value of $\tau = 10$ using both thresholds. When the matchings are exact ($\delta = 0$), the results do not increment or decrement when using different neighbors. The reason behind this is that we are reducing the neighbors so much that there are only a few users that can match the condition imposed

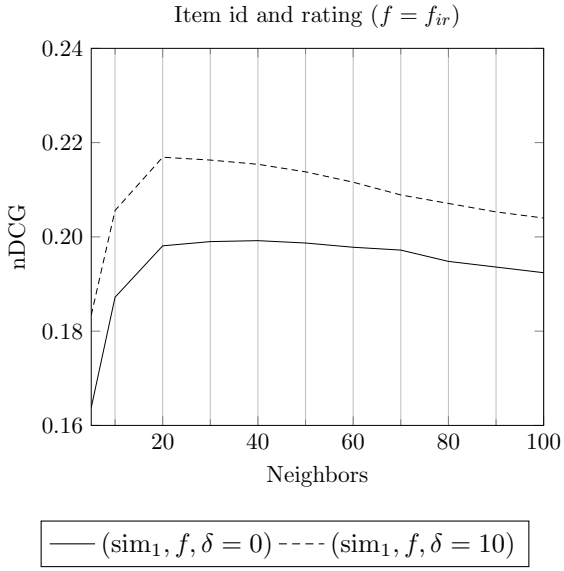


**Figure C.1**: Results of LCS-based similarity for Lastfm dataset. Pure collaborative-filtering approach with $\delta = 0$ and $\delta = 10$.

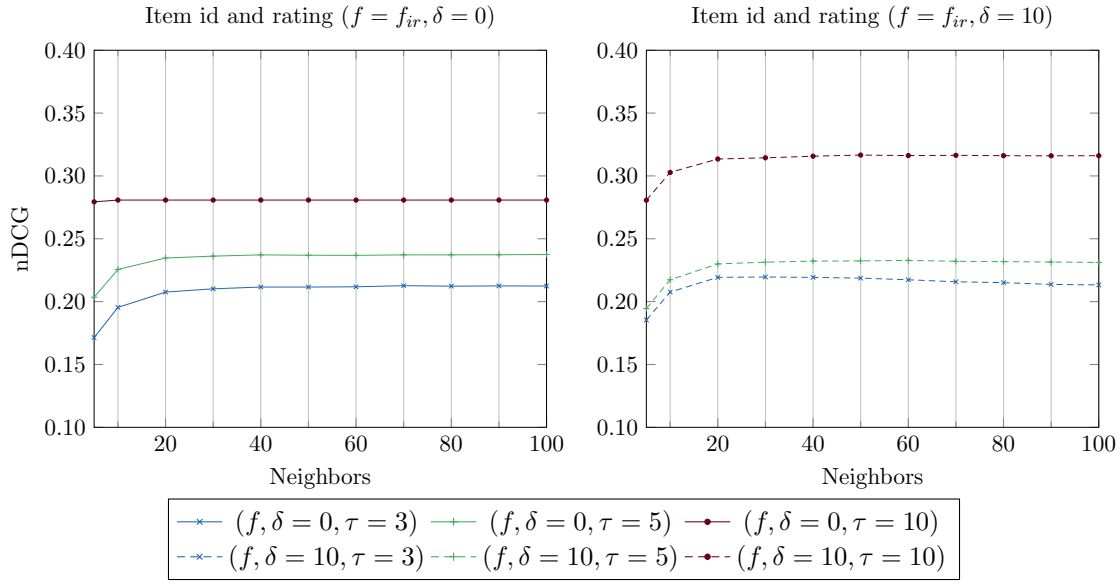


**Figure C.2**: Results of LCS-based similarity for Lastfm dataset. Different values of confidence filter parameter $\tau$ using $\delta = 0$ and $\delta = 10$.

**Table C.1**: Coverage of Lastfm dataset. Different values of confidence ($\tau$). The configuration for each recommender is denoted as $(f, \delta, \tau)$.

| **Recommender** | **Coverage** |
|---|---|
| $(f_{ir}, 0, 3)$ | 1,758.8 |
| $(f_{ir}, 0, 5)$ | 1,130.8 |
| $(f_{ir}, 0, 10)$ | 98.6 |
| $(f_{ir}, 10, 3)$ | 1,848.6 |
| $(f_{ir}, 10, 5)$ | 1,699.8 |
| $(f_{ir}, 10, 10)$ | 663.6 |

by the confidence value. In fact, there will be an important number of users that have less ratings than the value of confidence, making impossible to provide recommendations for them. The increase in performance in terms of nDCG is hence misleading as the users that we can make recommendations for are reduced drastically. Another relevant aspect about these results is that even when using a threshold of 10 ($\delta = 10$), the results for confidence values $\tau = 3$ and $\tau = 5$ are practically the same, although we can see that when using a confidence of 3, when using a high number of neighbors, the results are slightly worse. In this case, even if we augment the coverage with the threshold value, the quality of those neighbors are worse than the neighbors that have rated the items in the same way. However, we can see differences when using a confidence of 10 ($\tau = 10$). In this case, the performance in terms in nDCG is higher when allowing a threshold of 10 ($\delta = 10$).

In order to see the effect of this parameter over the coverage, in Table C.1 we show the coverage (the number of test users for whom we can make recommendations) of the recommenders of Figure C.2. As we can see, as the value of confidence increases, the coverage decreases. However, it is relevant to note that when using $\delta = 10$ we obtain higher results in terms of nDCG and higher values of coverage than those configurations with exact matchings for a $\gamma = 10$.

In Figure C.3 the reader can see the results for different values of the preference filter. In this case, both the preference values of 4 and $\overline{u}$ retrieve practically the same results for both thresholds (the differences are so small that they cannot be distinguished in the plots). On the other hand, contrary to the previous experiment, here we see that there is a significant growth in performance between 5 and 30 neighbors, but also a slower decrease when using more than 40 neighbors. We must take into account that the preference filter only allow us to generate shorter sequences, but we are not filtering neighbors, so in the long term it is normal that the neighbors lose some quality. In Table C.2 we show the results of coverage for these recommenders, in this case we see that the loss of coverage is in fact low with respect to the results obtained by modifying the confidence parameter, but also the effect in terms of performance is rather small. In this case, this behavior reminds us to the plots reported in Figure C.1, as the performance decay when using 30-40 neighbors, indicating that this parameter has little impact on the recommenders.

Now, we present in Figure C.4 the results when applying different values of preference and confidence at the same time. The value of preference is fixed to be the average of every user. In this case, we see an improvement when applying a confidence of 3 with a threshold of 10 with respect to the same configuration but with exact matchings ($\delta = 0$). However, when using a larger value of confidence ($\tau = 5$) the behavior is the opposite. This is an interesting observation because it indicates that when using a low number of the confidence filter, the new neighbors obtained by augmenting the matching are better than the ones obtained with a higher value of confidence. A possible explanation for this

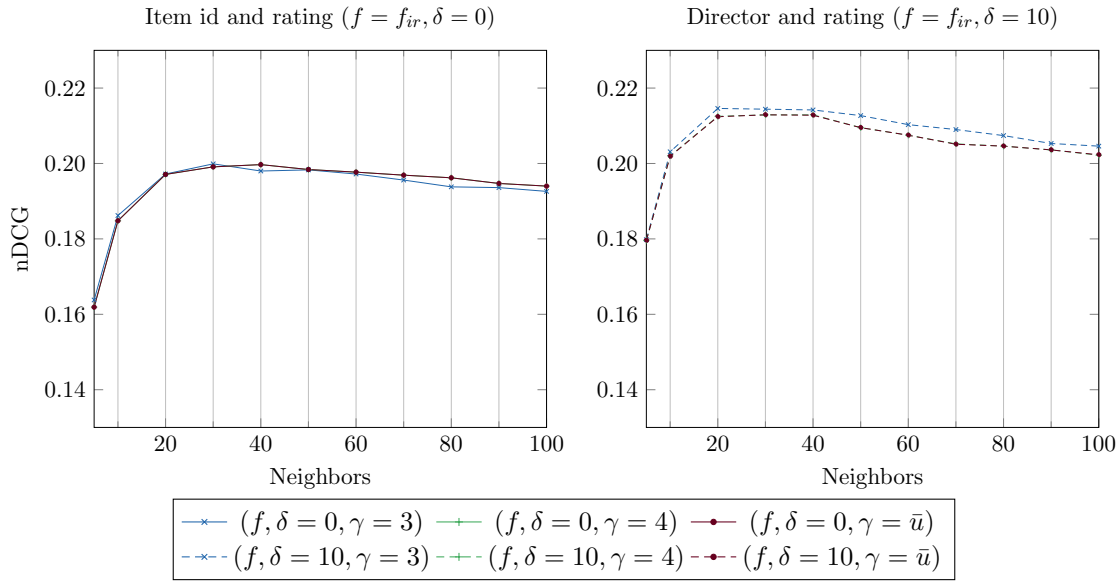


**Figure C.3**: Results of LCS-based similarity for Lastfm dataset. Different values of preference filter parameter using $\delta = 0$ and $\delta = 10$.

**Table C.2**: Coverage of Lastfm dataset. Different values of preference ($\gamma$). The configuration for each recommender is denoted as $(f, \delta, \gamma)$.

| **Recommender** | **Coverage** |
|---|---|
| $(f_{ir}, 0, 3)$ | $1,877.8$ |
| $(f_{ir}, 0, 4)$ | $1,871.0$ |
| $(f_{ir}, 0, \overline{u})$ | $1,871.0$ |
| $(f_{ir}, 10, 3)$ | $1,879.0$ |
| $(f_{ir}, 10, 4)$ | $1,873.2$ |
| $(f_{ir}, 10, \overline{u})$ | $1,876.6$ |

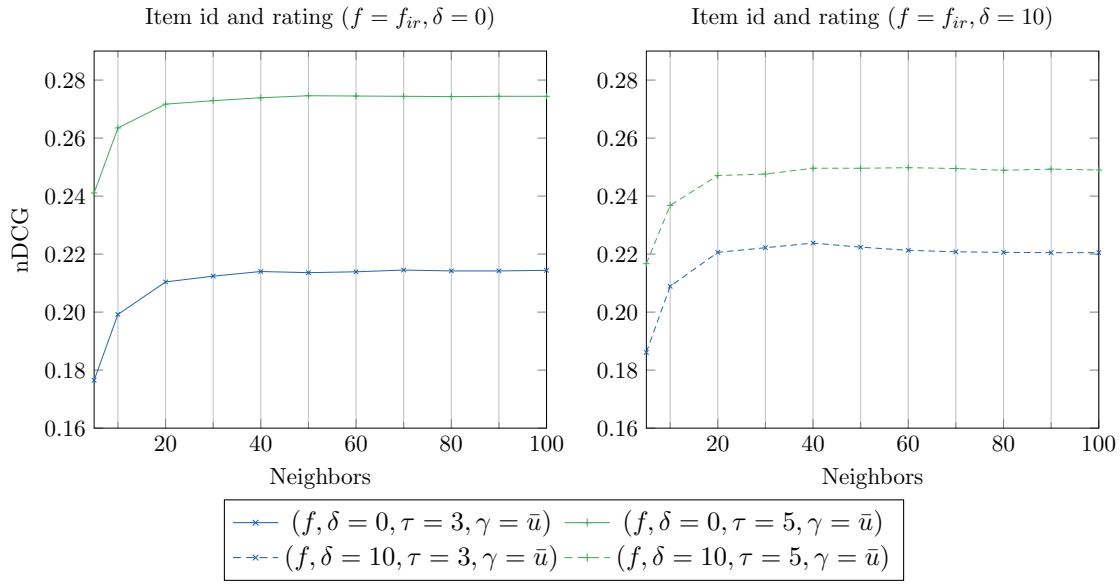


**Figure C.4**: Results of LCS-based similarity for Lastfm dataset. Combination of different values of preference and confidence filtering using $\delta = 0$ and $\delta = 10$.

could be related to the coverage. Table C.3 shows these values. We observe that the best recommender in terms of nDCG is the one that obtains the lowest values of coverage. It is relevant the difference of coverage when using a threshold of 0 and a threshold of 10 for a confidence of 5 ($\tau = 5$). We obtain almost twice the number of users that can be recommended, at the expense of only losing 0.03 of nDCG. This, together with the rest of results regarding coverage, evidences the difficulty of obtaining a recommender having good results in both coverage and ranking quality metrics. In fact, we can end up having very good recommenders but with a very low coverage, recommending only to users whose rating history is large, or, in other terms, *easier* users.

Considering the improvement of performance in the MovielensHetRec dataset when using normalization functions, in Figure C.5 we show the results of the same four normalization functions applied to the Lastfm dataset. In this case we observe a slightly different behavior. Unlike the case of MovielensHetRec, there is only one normalization function that outperforms the basic configuration, $\text{sim}_2$. Actually, in the previous experiment this normalization function was also the best performing one, but in this case the main difference is that the rest of normalization functions degrade to practically the same recommendations. Furthermore, we also observe the same situation found in Figure C.1: when more than 50 neighbors are used, the performance begins to drop. Although this is an

**Table C.3**: Coverage of Lastfm dataset. Different values of confidence ($\tau$) and preference ($\gamma$). The configuration for each recommender is denoted by the following order $(f, \delta, \tau, \gamma)$.

| **Recommender** | **Coverage** |
|---|---|
| $(f_{ir}, 0, 3, \overline{u})$ | $1,587.6$ |
| $(f_{ir}, 0, 5, \overline{u})$ | $700.6$ |
| $(f_{ir}, 10, 3, \overline{u})$ | $1,771.0$ |
| $(f_{ir}, 10, 5, \overline{u})$ | $1,243.2$ |

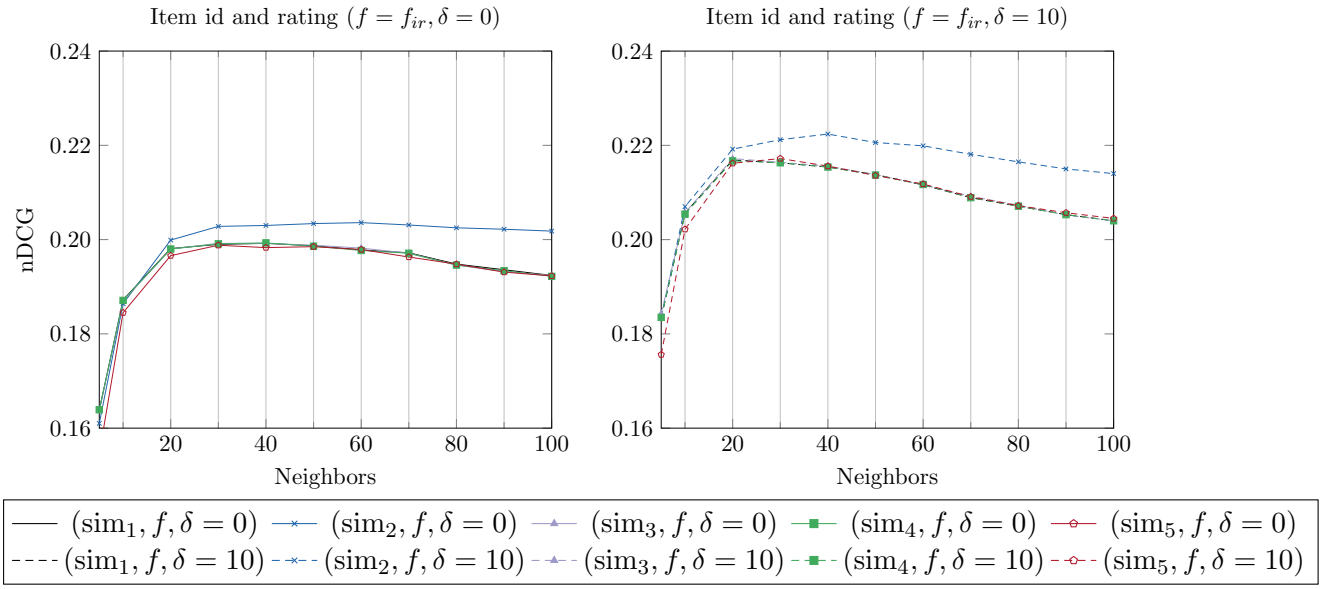


**Figure C.5**: Results of LCS-based similarity for Lastfm dataset. Different values of normalizations using $\delta = 0$ and $\delta = 10$.

expected behavior because of previous experiments presented in this section, in the other dataset the performance continued increasing when more neighbors were used. Again, this may be explained by the sparsity of the dataset. In average, for this dataset each item has been rated by 5 users, while in the MovielensHetRec the average is close to 85, thus, when using a higher number of neighbors, we would start considering other users having a small number of items in common with the target user, making these neighbors useless to make recommendations.

As done for the MovielensHetRec dataset, in Figure C.6 we show the results of different values of preference and confidence (and combinations) with and without the normalization function $\text{sim}_2$. Although we performed experiments for the rest of normalization functions and combinations of preference and confidence, we think that this example is enough to illustrate the effect of these three parameters in this dataset. In this case we see that, in general, when using a threshold of 10 ($\delta = 10$), the performance is lower than when using exact matchings. However, we must take into account the coverage component analyzed in the previous experiments, which tends to be higher for $\delta > 0$. In this figure we also see that when using a large value of confidence (in this case, 5) the normalization function has practically no effect. The reason for this could be that, since we are using a high confidence value, the neighbors are the best ones that the user can have, so applying a normalization between them has practically no effect. However, when using only the preference value we see that including the normalization into the process produces results that are slightly higher.

We now present the results for the Lastfm dataset analyzing the rest of ranking metrics, together with novelty and diversity metrics for both the LCS-based recommenders and the baselines. Table C.4 includes these results. As pointed out before, for this dataset we do not report results using any content-based approach from LCS or from any other baseline. In summary, when analyzing the LCS approaches we see that the configurations where a normalization function is applied perform better in most of the metrics than the non normalized approaches, even for novelty and diversity metrics. Furthermore, in this case,

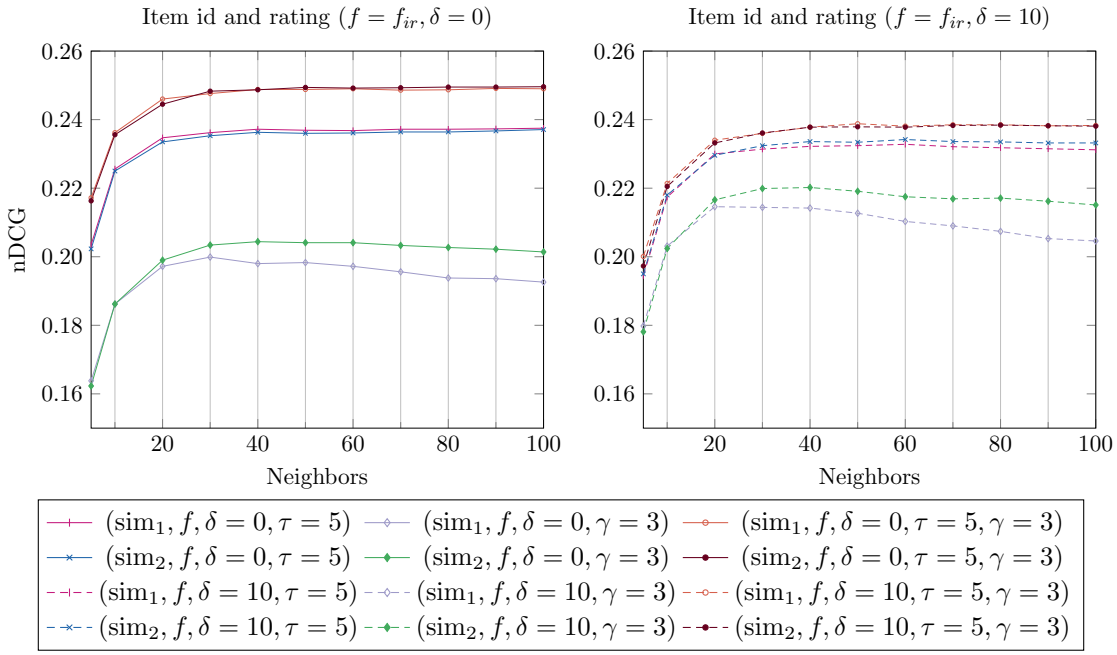


**Figure C.6**: Results with the best confidence and preference filters and normalizations for Lastfm dataset. Results using $\delta = 0$ and $\delta = 10$.

using both normalization and a threshold of 10 entail better results in all the used metrics (only for EPD and EILD our recommenders with exact matchings obtain better results). However, the improvement in general is lower in percentage with respect to the experiments of MovielensHetRec, probably because there are less parameters to configure.

Let us now analyze the behavior of the baselines. In this case, we observe that although the UB1 and MF algorithms beat the best configuration of our LCS-based recommender, its results are highly competitive; in fact, it outperforms most of the other baselines, such as popularity and both IB1 and IB2 in all ranking metrics. This is the same behavior that we observed for the MovielensHetRec dataset. For the UB1 and UB2 case, the results of the LCS recommender are close to these baselines in terms of ranking evaluation metrics, but as we can see, it obtains higher values in Aggregate diversity, which makes our approach the user-based recommender with the highest results in this metric. However, it is also relevant to show that the IB recommenders obtain the highest values in diversity metrics, especially in Gini and Aggregate diversity, indicating that they are the recommenders that are able to retrieve many different items.

**Table C.4**: Performance of baselines in Lastfm dataset in terms of ranking quality (nDCG, P, R, MAP), novelty (EPC, EPD) and diversity (AD, $\alpha$-nDCG, EILD, Gini).

| Recommender | nDCG | P | R | MAP | EPC | EPD | AD | $\alpha$-nDCG | EILD | Gini |
|---|---|---|---|---|---|---|---|---|---|---|
| MF | **0.261** | **0.123** | **0.288** | **0.203** | **0.925** | 0.870 | 26.77% | **0.223** | 0.874 | 0.014 |
| Pop | 0.082 | 0.040 | 0.093 | 0.060 | 0.792 | **0.922** | 1.35% | 0.064 | **0.933** | 0.000 |
| UB1 | 0.223 | 0.106 | 0.246 | 0.172 | 0.883 | 0.895 | 53.60% | 0.191 | 0.896 | 0.006 |
| UB2 | 0.222 | 0.106 | 0.245 | 0.171 | 0.883 | 0.895 | 52.70% | 0.191 | 0.896 | 0.005 |
| IB1 | 0.211 | 0.101 | 0.235 | 0.162 | 0.913 | 0.732 | 81.93% | 0.171 | 0.721 | 0.027 |
| IB2 | 0.214 | 0.100 | 0.231 | 0.167 | 0.912 | 0.694 | **86.81%** | 0.175 | 0.681 | **0.034** |
| $(\text{sim}_1, f_{ir}, 0)$ | 0.199 | 0.094 | 0.219 | 0.154 | 0.866 | 0.906 | 49.72% | 0.171 | 0.906 | 0.004 |
| $(\text{sim}_2, f_{ir}, 0)$ | 0.204 | 0.096 | 0.223 | 0.157 | 0.868 | 0.900 | 56.99% | 0.174 | 0.899 | 0.004 |
| $(\text{sim}_1, f_{ir}, 10)$ | 0.215 | 0.102 | 0.237 | 0.166 | 0.873 | 0.901 | 47.90% | 0.186 | 0.902 | 0.004 |
| $(\text{sim}_2, f_{ir}, 10)$ | 0.222 | 0.106 | 0.245 | 0.171 | 0.879 | 0.887 | 59.22% | 0.190 | 0.887 | 0.006 |