A Biclustering Approach to Recommender Systems

Florestan de Moor

Master research Internship

Internship report

A Biclustering Approach to Recommender Systems

Domain: Machine Learning

Author:
Florestan De Moor

Supervisors:
Davide Frey (INRIA, WIDE)
Antonio Mucherino (IRISA, Univ Rennes)

école normale supérieure rennes

Abstract. Recommendation systems are a core component of many e-commerce industries and online services since they ease the discovery of relevant products. Because catalogs are huge, it is impossible for an individual to manually search for an item of interest, hence the need for some automatic filtering process. Many approaches exist, from content-based ones to collaborative filtering that include neighborhood and model-based techniques. Despite these intensive research activities, numerous challenges remain to be addressed, particularly under real-time settings or regarding privacy concerns, which motivates further work in this area.

We focus on techniques that rely on biclustering, which consists in simultaneously building clusters over the two dimensions of a data matrix. Although it was little considered by the recommendation system community, it is a well-known technique in other domains such as genomics. In this report, we present the different biclustering-based approaches that were explored. We then are the first to perform an extensive experimental evaluation to compare these approaches with one another, but also with the current state-of-the-art techniques from the recommender field. Existing evaluations are often restrained to a few algorithms and consider only a limited set of metrics. We then expose a few ideas to improve existing approaches and address the current challenges in the design of highly efficient recommendation algorithms, along with some preliminary results.

Contents

1 Introduction

Due to the democratization of the Internet, many e-commerce industries and online services such as Amazon, Netflix or IMDb appeared in the 90s. With the era of big data, their catalogs have been expanding with many new products while the systems themselves have become more popular and drawn many new users, reaching a tremendous scale in terms of involved data. As a consequence, it has become cumbersome if not impossible for an individual to search for a single item of interest. This motivated the design of algorithms able to perform automatic filtering to ease the discovery of relevant content, known as a recommendation process. The goal is to provide a customer with a list of available products that are likely to match their interests. This can be achieved through various data mining and machine learning techniques by using the available data regarding the user preferences or product characteristics.

Because of the scale of these systems, the search of highly efficient algorithms has been a major research topic and reached a peak between 2006 and 2009 with the Netflix Prize competition [1]. The online movie streaming company released an anonymized version of the data they had collected through their system over the years and challenged researchers and engineers with the task of increasing the quality of their recommendation process by a given factor. A team met the goal of the competition in 2009 and won the prize of one million dollars. A second edition was planned but was finally canceled when Netflix got sued for privacy concerns after the authors of [34] successively de-anonymized the records of two users in the publicly released dataset by exploiting available auxiliary information from the IMDb website. This competition led to the emergence of model-based approaches such as matrix factorization [28] which are considered superior to standard neighborhood-based methods that were popular at the time.

The search for efficient recommendation algorithms is still an active research topic since most current systems not only process huge amounts of data but are also very dynamic with many new users registering, new products being created, or even the preferences of some users shifting over time. For instance, Netflix indicated in a blog post published in 2012 that they have 4 million new ratings every day [4]. The processing of large chunks of data in a short time window is a fundamental challenge for news recommender systems where the interests of users for certain topics can suddenly spike and then quickly fade away. In addition to these real-time settings, privacy and security are still a hot topic to maintain a safe environment and sustain the user trust in the service. For all these reasons, new recommendation approaches are explored to develop solutions to these issues.

In this report, we focus on a biclustering approach to recommendation systems that was little covered by the literature although we believe it has great potential. The data used in the recommendation process exhibits a sample/feature structure that is similar to the structure of data for gene expressions. Among the genomics community, biclustering is a standard approach that has been heavily studied. The purpose of a bicluster is to determine a group of genes that are all highly expressed in a subset of all the samples. Similarly, in the recommendation context, we wish to find a group of users that share common preferences regarding a subset of items. Several biclusters form a biclustering, and such information can then be used to provide recommendations as we describe later in Section 3.4.

1.1 Contributions

We perform a high level survey of the recommendation systems field and then focus more thoroughly on the few biclustering-based approaches that were proposed.

Our first contribution is an extensive experimental evaluation to better understand how these approaches perform in comparison with each other, but also with the current state-of-the-art methods. To the best of our knowledge, we are the first to consider such a variety of biclustering-based and standard approaches, along with a large range of metrics that do not only focus on accuracy but also include coverage and time performance aspects. Our evaluation allows us to identify the strengths and weaknesses of biclustering-based algorithms, and to sketch what we believe to be promising avenues of research in this area.

Our second contribution consists in the exploration of several ideas. We consider the biclustering-based algorithm that we believe stands out from our evaluation and propose several improvements, including a self-tuning method for its parameters and the addition of a regularization term to the optimization problem solved. We show some preliminary experimental results that suggest interesting further work could be achieved in this direction.

1.2 Outline

First, we detail the problem we focus on in Section 2, and present the related work in Section 3. Our experimental setup is then described in Section 4 and the results we obtained along with a discussion are given in Section 5. Finally, we expose perspectives of future research we have started to explore in Section 6 and conclude in Section 7.

2 Problem Statement

A recommender system is composed of three main components:

(i) The users (or customers in case of an e-commerce application),

(ii) the items (such as products available to buy, videos, music tracks) that form the catalog of the system (which can be manually searched although its size makes it hard to find items of interest unless one knows exactly what they are looking for),

(iii) and the recommendation engine which processes all the available data and provides each user with a list of recommended items.

We denote by $\mathcal{U}$ the set of users and by $\mathcal{I}$ the set of items, in the system.

Definition 1 (User-item matrix)**.** Given a rating range $\mathcal{R} \subseteq \mathbb{R}$, the user-item matrix of the system, denoted $\mathbf{R}$, is a matrix with set of rows $\mathcal{U}$ and set of columns $\mathcal{I}$ such that:

$$\forall (u, j) \in \mathcal{U} \times \mathcal{I} \quad \mathbf{R}_{u,j} \in \mathcal{R} \cup \{\bot\}$$

An element $\mathbf{R}_{u,j}$ expresses the opinion of user u about item j, or is equal to $\bot$ if the rating is unknown, as illustrated in Figure 1.

The set of possible ratings can substantially vary depending on the considered system. It can for instance be binary (such as a thumb up/down on a YouTube video, $\mathcal{R} = \{+1, -1\}$), or more expressive (such as 5-star ratings, $\mathcal{R} = \{1 \dots 5\}$).

The user-item matrix is usually obtained by building a profile for each user of the system and inferring their preferences through the collection of available information that falls into two main categories: explicit feedback (*i.e.* opinion the user has expressed on purpose, for instance by posting a review of a film) and implicit feedback (*i.e.* what is deduced from the user behavior, such as its navigation history or video watch time). How to build this matrix is not the focus of our work and it is considered as input data in the remainder of this report.

A variety of tasks can be identified as goals of a recommender system, as expressed in [25, 21]. In this work, we focus on the two following, in terms of recommendation quality:

Task 1 (Prediction)**.** Given a user u, and an item j such that $\mathbf{R}_{u,j} = \bot$, predict the score $\widehat{r_{u,j}}$ that u would give to j if they were to rate it.

	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9	j_{10}
u_1	$\bot$	3	$\bot$	2	$\bot$	1	$\bot$	5	$\bot$	5
u_2	1	$\bot$	2	$\bot$	2	$\bot$	4	$\bot$	4	$\bot$
u_3	2	$\bot$	$\bot$	1	$\bot$	$\bot$	2	1	4	5
u_4	$\bot$	1	3	3	3	$\bot$	$\bot$	$\bot$	5	4

Figure 1: Example of a user-item matrix with $n = 4$, $m = 10$ and 5-star ratings.

Task 2 (Recommend good items)**.** Given a user u, determine a set of items unknown to u that would be relevant to u.

A particular instance is the top-N recommendation task, which has three steps:

1. Identify a set C_u of candidate items.
2. For each item j in C_u, compute a score $\widehat{r_{u,j}}$ (Prediction task).
3. Output the set L_u^N which contains the N best-scoring items of C_u.

An algorithm that is able to predict a score is also able to provide lists of recommendations through the top-N recommendation task described. However, the converse is not true: it is possible to compute a ranking of candidate items and thus to recommend the best ones while being unable to provide a meaningful score for a particular item given some rating scale. Since both tasks are not directly correlated, it may happen that an approach is particularly effective at one task but not at the other.

The main notations used in the remainder of this report are given in Table 1.

u	user	j	item
$\mathcal{U}$	set of users	$\mathcal{I}$	set of items
$\mathbf{R}$	user-item matrix	$\mathbf{R}_{u,j}$	rating of u for j
$\widehat{r_{u,j}}$	predicted rating of u for j	L_u^N	list of N recommendations for u
$\mathcal{R}$	rating range	$\perp$	unknown rating
P_u	profile of user u	P^j	profile of item j
$B = (U, J)$	bicluster	$\mathcal{B} = (B_k)_k$	biclustering

Table 1: Main notations used in this report.

3 Related Work

3.1 Biclustering

We first give a general definition of a bicluster. Although it is generic, we use the notations specific to recommendation systems introduced earlier.

Definition 2 (Bicluster)**.** Let us consider a matrix $\mathbf{R}$ of size $n \times m$ with index set of rows $\mathcal{U}$ and index set of columns $\mathcal{I}$. A bicluster $B = (U, J)$ is a sub-matrix of $\mathbf{R}$ with index set of rows $U \subset \mathcal{U}$ and index set of columns $J \subset \mathcal{I}$, and subject to a set of constraints $\mathcal{C}_b$.

When $\mathcal{C}_b = \varnothing$, we call B an unconstrained bicluster.

We now review some constraints found in the literature.

Definition 3 (Inclusion-maximal bicluster [39, 3])**.** Given a property P, an inclusion-maximal bicluster is a bicluster B such that:

(i) P holds in the bicluster,

(ii) the addition of any other row or column to B would yield a bicluster B' where P is false.

For instance, under binary settings and considering sub-matrices of constant values, an inclusion maximal bicluster may be a sub-matrix of 1's where the addition of any other row or column would introduce a 0 as illustrated in Figure 2a.

Definition 4 (Expression motif [33])**.** Given two parameters $0 < \alpha, \beta < 1$ and a set S of intervals, an expression motif is a bicluster (U, J) with the three following constraints:

(i) Size: $|U| \geq \alpha n$

(ii) Conservation: $\forall j \in J, \exists s_j \in S \ st. \ \forall u \in U \quad \mathbf{R}_{u,j} \in s_j$

(iii) Maximality: $\forall j' \notin J, \forall s \in S \quad |\{u \in U \mid \mathbf{R}_{u,j'} \in s\}| \leq \beta |U|$

Definition 5 (Consistent bicluster [29])**.** Given a parameter $0 < c \leq 1$, a consistent bicluster is a bicluster (U, J) with a consistency level $C(U, J)$ greater than c.

The goal is to expose a sub-matrix which exhibits a particular behavior, for instance containing very similar values. One could give a measure of their choice depending on the application wished. We give below the one defined in [29]:

$$C(U, J) = \min_{(u,j) \in U \times J} \frac{|\{u' \in U \mid \mathbf{R}_{u,j} = \mathbf{R}_{u',j} \neq 0\}|}{|U|}$$

i.e. it is the minimum proportion of identical non-zero values within a column of the sub-matrix.

An illustration of these definitions is provided in Figure 2 with a matrix of size 7×6 that contains integers from 0 to 5 which simulate movie ratings (2b and 2c) and its binary version using 3 as threshold (2a).

We now present a general definition of the biclustering problem.

Definition 6 (Biclustering)**.** Let's consider a matrix $\mathbf{R}$. The biclustering problem consists in finding a set $\mathcal{B} = (B_k)_{0 \leq k < K}$ subject to a set of constraints $\mathcal{C}_B$, where $B_k = (U_k, J_k)$ is a bicluster in the sense given by one of the definitions above.

If $\mathcal{B}$ contains every possible bicluster that exist in $\mathbf{R}$ and meet $\mathcal{C}_B$, then $\mathcal{B}$ is an exhaustive biclustering.

	j_1	j_2	j_3	j_4	j_5	j_6
u_1	1	1	1	1	0	0
u_2	1	0	1	0	0	1
u_3	1	1	1	1	1	0
u_4	1	1	0	0	0	0
u_5	0	0	0	0	0	1
u_6	1	0	1	1	0	0
u_7	0	1	0	1	1	0

(a) Constant inclusion-maximal:
$U = \{u_1, u_3, u_4\}$,
$J = \{j_1, j_2\}$

	j_1	j_2	j_3	j_4	j_5	j_6
u_1	5	4	3	4	0	1
u_2	4	2	4	1	2	5
u_3	4	5	4	4	5	2
u_4	4	5	1	0	2	1
u_5	2	0	1	1	0	4
u_6	4	0	3	4	1	2
u_7	0	5	2	4	3	0

(b) Expression motif:
$U = \{u_1, u_2, u_3, u_6\}$,
$J = \{j_1, j_3\}$, $\alpha = 0.5$, $\beta = 0.25$,
$S = \{[0, 2], [3, 5]\}$

	j_1	j_2	j_3	j_4	j_5	j_6
u_1	5	4	3	4	0	1
u_2	4	2	4	1	2	5
u_3	4	5	4	4	5	2
u_4	4	5	1	0	2	1
u_5	2	0	1	1	0	4
u_6	4	0	3	4	1	2
u_7	0	5	2	4	3	0

(c) Consistent:
$U = \{u_1, u_3, u_4, u_6, u_7\}$,
$J = \{j_1, j_2, j_4\}$, $C(I, J) = 3/5$

Figure 2: Illustration of several bicluster definitions

The number of biclusters to find may be specified by an input parameter. We review some constraints we found in the literature.

Definition 7 (Biclustering in COCLUST [18])**.** This approach used in the context of a recommender system tries to minimize the error on the score prediction. More precisely, it consists in finding a set of k user clusters and a set of l items clusters, which form a partitional biclustering [5] (also called checkerboard biclustering *i.e.* each cell of the matrix belong to one and only one bicluster). The biclustering is represented by two functions ρ and γ that map respectively users and items to their corresponding clusters. More precisely, the objective function is:

$$\min_{(\rho,\gamma)} \sum_{\substack{(u,j)\in\mathcal{U}\times\mathcal{I} \\ \mathbf{R}_{u,j}\neq\perp}} (\mathbf{R}_{u,j} - \widehat{r_{u,j}})^2$$

where $\widehat{r_{u,j}} = \mathbf{A}^{BICL}_{\rho(u),\gamma(j)} + (\mathbf{A}^{R}_{u} - \mathbf{A}^{RC}_{\rho(u)}) + (\mathbf{A}^{C}_{j} - \mathbf{A}^{CC}_{\gamma(j)})$ with the following quantities illustrated in Figure 3:

- $\mathbf{A}^{BICL}_{\rho(u),\gamma(j)}$: the **a**verage value of the **bicl**uster
- $\mathbf{A}^{R}_{u}$: the **a**verage rating of the **r**ow (*i.e.* of user u)
- $\mathbf{A}^{RC}_{\rho(u)}$: the **a**verage rating of the **r**ow of item **c**lusters
- $\mathbf{A}^{C}_{j}$: the **a**verage rating of the **c**olumn (*i.e.* for item j)
- $\mathbf{A}^{CC}_{\gamma(j)}$: the **a**verage rating for the **c**olumn of users **c**lusters

Definition 8 (Fitness function minimized in BIC-aiNet [12])**.** Given three parameters λ, w_r and w_c which respectively denote a residue threshold and the importance given to the number of rows and columns, this problem consists in finding a biclustering $\mathcal{B}$ that minimizes a fitness function as follows:

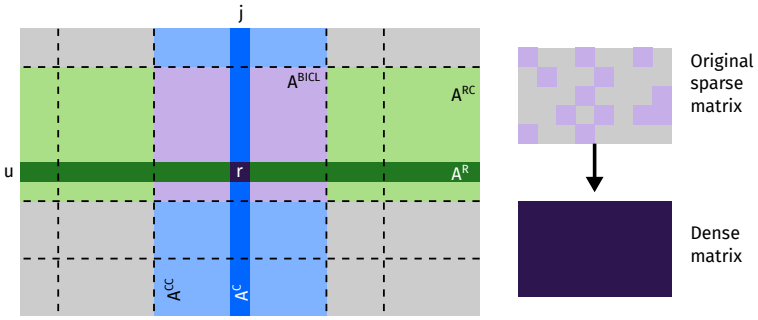


Figure 3: Score prediction in the COCLUST algorithm.

$$\min_{\mathcal{B}} \left(\max_{k} \frac{R_k}{\lambda} + \frac{w_r \cdot \lambda}{|U_k|} + \frac{w_c \cdot \lambda}{|J_k|} \right)$$

with R_k being a residue defined as $R_k = \frac{1}{|U_k| \cdot |J_k|} \sum_{(u,j) \in B_k} \left(\mathbf{R}_{u,j} - \mathbf{R}_{u,J_k} - \mathbf{R}_{U_k,j} + \mathbf{R}_{U_k,J_k} \right)^2$ where $\mathbf{R}_{u,J_k}$ is the mean value of the row, $\mathbf{R}_{U_k,j}$ the mean value of the column, and $\mathbf{R}_{U_k,J_k}$ the mean value of the sub-matrix.

Intuitively, this consists in searching the best trade-off between two conflicting elements: maximizing the size of the bicluster and minimizing the residue which quantifies the variance of the values within the bicluster.

Definition 9 (Size maximized [29, 33])**.** This problem consists in finding a biclustering $\mathcal{B}$ that maximizes the size of each individual biclusters:

$$\max_{\mathcal{B}} \left(\min_{k} \left[\min \left(|U_k|, |J_k| \right) \right] \right)$$

Alternatively, the inner minimum can be adapted if one wishes to constrain only one of the two dimensions (either the samples or the features).

We now present two properties of a biclustering which are coverage and overlapping.

Definition 10 (Coverage)**.** A biclustering $\mathcal{B} = (U_k, J_k)_{0 \leq k < K}$ of the matrix $\mathbf{R}$ is said to cover a cell $(u, j) \in (\mathcal{U} \times \mathcal{I})$ *iif*

$$\exists k \in \{0, 1, \ldots, K-1\} \ st. \quad u \in U_k \wedge j \in J_k$$

i.e. the cell appears in at least one bicluster. The coverage frequency is then given as:

$$coverage(\mathcal{B}) = \frac{|\{(u,j) \in (\mathcal{U} \times \mathcal{I}) \mid (u,j) \text{ covered by } B\}|}{n \cdot m}$$

In the case of a biclustering with a chessboard structure such as in Definition 7, we have $coverage(\mathcal{B}) = 1$.

Definition 11 (Overlapping)**.** Two biclusters $B_{k_1} = (U_{k_1}, J_{k_1})$ and $B_{k_2} = (U_{k_2}, J_{k_2})$ are said to overlap *iif*

$$(U_{k_1} \times J_{k_1}) \cap (U_{k_2} \times J_{k_2}) \neq \varnothing$$

The overlap degree [35] is defined as:

$$o-degree(B_{k_1}, B_{k_2}) = \begin{cases} \max |U_{k_1} \cap U_{k_2}|, |J_{k_1} \cap J_{k_2}| & \text{if } B_{k_1} \text{ and } B_{k_2} \text{ overlap} \\ 0 & \text{otherwise} \end{cases}$$

A biclustering $\mathcal{B} = (B_k)_{0 \leq k < K}$ of the matrix $\mathbf{R}$ is said to have overlapping *iif* there exists at least two biclusters which overlap. The overlap degree of the whole biclustering is then:

$$o-degree(\mathcal{B}) = \max_{k_1 \neq k_2} [o-degree(B_{k_1}, B_{k_2})]$$

Overlapping is not allowed in a checkerboard structure. All other biclustering definitions reviewed allow overlapping.

Remark 1. Since the biclustering problem has been extensively studied by the bioinformatics community, we could give many more definitions, but we believe the ones we have presented convinced the reader of the abundance of viewpoints. In particular, we wish to emphasize the following.

(i) One can consider constraints on the size (Definitions 3 and 4) or the content of each individual bicluster (Definitions 3, 4 and 5), at a local level.

(ii) There can be constraints on the content of the biclusters at a global level, through minimizing or maximizing an objective function for the whole biclustering set (Definitions 7, 8 and 9).

(iii) One can also wish to have constraints on the structure of the resulting biclustering set such as a checkerboard structure imposed (Definition 7) or the prohibition of overlapping.

Remark 2. With these many definitions comes a plethora of algorithms that solve specific instances of the biclustering problem. They use different approaches to tackle this problem such as exhaustive divide-and-conquer search [39], iterative heuristic search [12, 29, 33, 18] or SVD resolution through a bipartite graph point of view [16]. Because of that, one should carefully choose a relevant algorithm depending on the precise instance of the biclustering problem it wishes to solve.

3.2 Main challenges in recommender systems

We now review a set of challenges and provide some clues to help understand their prominence in the search of better recommendation algorithms. As we present the main employed techniques in Section 3.3, we mentions there why we believe the current solutions do not completely address these challenges. In Figure 4, we show an overview of a recommender system where challenges are highlighted at various points in the process.

Scalability. It is common for the item set of a recommendation system to grow at a huge rate, particularly when considering e-commerce applications where new products are constantly created. Similarly, with the democratization of the internet and the emergence of trendy applications, a recommendation system may see a quick increase in its number of users. For these reasons, the user-item matrix of the system grows in size and ends up holding huge amounts of data, although it often remains sparse. It is often assumed that scaling up is cheap through cloud computing which offers great elasticity to request the exact amount of processing power that is needed. However, the costs this approach implies are overlooked, and while they may be fine for big companies, they can be highly problematic for medium and small ones which have limited budget and resources.

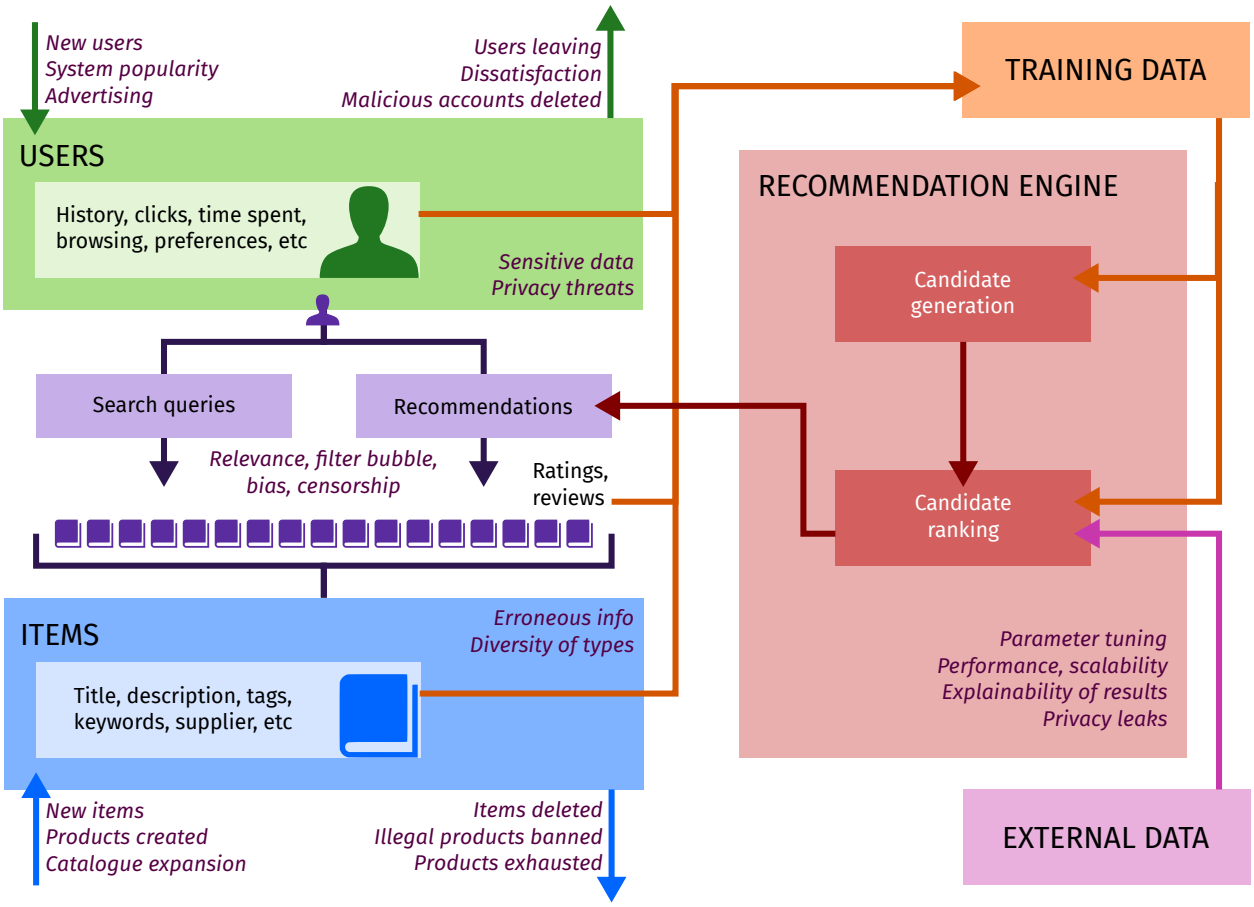


Figure 4: High level overview of a recommendation system with its users, items and recommendation engine. Challenges are highlighted in italic at several places of the process.

The design of highly efficient protocols that can scale under heavy usage is therefore a fundamental challenge, especially for small infrastructures.

Real-time. Today's data is not only big but also fast. Systems thus need to be able to process quickly new information and to cope with temporal constraints. Ignoring the latest information could result in less relevant recommendations and thus damage the user experience. The search for algorithms that are both scalable and responsive under today's real-world dynamic settings is a key challenge, especially when it comes to news recommenders or micro-blogging platforms where spikes of interest and trends can appear all of a sudden and last only a couple of hours.

Parameter tuning. Most recommendation techniques use a few parameters that need to be set at initialization time, and which may greatly influence performance. Determining the ideal value for these is often performed through a long iterative process, which can reveal to be cumbersome if their space domain is huge or if no clear behavior emerges. Furthermore, some parameters cannot be associated with an intuitive meaning, which makes it even harder to determine reasonable values.

Explainability. Like other machine learning techniques, the recommendation process suffers from explainability issues: while some methods such as content-based allow for an easy understanding of the reasons why specific items were output to a particular user, others behave as black boxes and

give results with no intuitive explanations. It has therefore become a growing concern to design methods with explainable results, especially since recommendation systems have started to be used for critical tasks such as medical diagnosis [46]. In this respect, some recent work focused on matrix factorization techniques [2].

Security and privacy concerns. Because recommendation systems are becoming the primary way to find items of interests in a catalog, at the expense of search bars, many worries can be expressed. How to detect biases introduced from the owner of the system to highlight some products for hidden interests? Can the system be attacked to perform censorship or to promote particular products? Do users get trapped in a filter bubble? These are legitimate questions which have led to extensive research to study possible attacks and elaborate efficient countermeasures. To address the third question, we can notice a growing trend for the design of systems that favor unexpected recommendations (a concept known as serendipity [17]).

Privacy is also a major challenge in these systems that manipulate huge quantities of data that can be considered as sensitive information. It is therefore important to devise techniques to protect them and avoid privacy leaks. This is particularly fundamental for systems that rely on collaborative filtering where the preferences of every user are aggregated to produce recommendations for a specific individual.

3.3 Recommendation techniques

The design of efficient algorithms has been an active research topic for decades to address the many issues mentioned in the previous section. Three main categories are usually distinguished: content-based, collaborative filtering (CF) and hybrid techniques. We now give a brief overview of each technique but focus more thoroughly on CF approaches since they have been largely used in numerous academic and industrial systems.

3.3.1 Content-based

These techniques rely on the characteristics of the items to provide recommendations to a specific user given its history profile [37]. This is achieved by taking advantage of the tags, keywords or technical specifications of an item which can be given by its creator or automatically generated via an analysis.

Depending on the item nature, this can reveal challenging, since it is much easier to extract information from plain text than from an audio or video recording. The items that have similar characteristics to the ones already positively rated by a user u then get recommended to u. Given that the recommendations are provided independently of the opinion of all the other users of the system, these techniques have a natural resilience to privacy threats. However, they often turn out to have poorer recommendation quality than other approaches, while being hard to implement in some cases where it is difficult to infer the item characteristics with an analysis.

3.3.2 Collaborative filtering

Since most recommendation systems are used by e-commerce industries or online social platforms, they inherently have to deal with many users. Collaborative filtering aims at taking full advantage of the preferences of a group or all the users in the system, to provide more accurate recommendations. It makes use of a user-item matrix where a cell indicates the rating of a user u regarding an item j,

or is empty if u has no associated opinion on j. The purpose is then to predict the missing values by exploiting all the information available in the matrix. Intuitively, these approaches often rely on the assumption that users who shared similar interests in the past will also share similar interests in the future.

A simple method (IAVG) is to consider the popularity of the items: the score predicted is the average rating for this item, no matter the user.

$$\widehat{r_{u,j}} = \frac{1}{|P^j|} \sum_{v \in P^j} \mathbf{R}_{v,j}$$

A better version (IUAVG) can be derived by including information about the user. It adds to the prediction a bias which measures how different the average rating of this user is compared to the average rating in the whole dataset.

$$\widehat{r_{u,j}} = \frac{1}{|P^j|} \sum_{v \in P^j} \mathbf{R}_{v,j} + \left(\frac{1}{|P_u|} \sum_{s \in P_u} \mathbf{R}_{u,s} - \frac{\sum_{v \in \mathcal{U}} \sum_{s \in P_v} \mathbf{R}_{v,s}}{\sum_{v \in \mathcal{U}} |P_v|} \right)$$

Collaborative filtering is known to provide high recommendation quality and is currently used by numerous companies such as Netflix [20] or Amazon [30]. However, these techniques suffer from several weaknesses. First, it is difficult to provide recommendations to new users and new items cannot be recommended, this is known as the cold start problem. Then, scalability is an issue, these techniques are often computationally expensive and do not fit well to huge real-time systems. Last but not least, the collaborative aspect raises many concerns about user privacy and there has been active research to both identity potential threats such as shilling [43] or Sybil attacks [9] and propose privacy-preserving mechanisms that include decentralized settings with homomorphic encryption [11], profile obfuscation [7] or designs that provide differential privacy [32].

We now give further details about the most popular collaborative filtering algorithms, that are often classified into two categories: neighborhood-based and model-based.

Neighborhood-based. These rely on the search of similar users or items, according to some metric *sim* such as the cosine angle or the Pearson correlation which are popular choices:

$$cosine(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} \qquad pearson(\mathbf{a}, \mathbf{b}) = cosine(\mathbf{a} - \bar{\mathbf{a}}, \mathbf{b} - \bar{\mathbf{b}})$$

where $\bar{\mathbf{x}}$ denotes the average value of the vector $\mathbf{x}$. These metrics allow to compute a similarity score between two users or two items by considering either user or item profiles.

In the user-based version (denoted UBKNN) [8], a kNN graph is built to find the k nearest users of each user u. Candidate items for recommendation are those liked by the neighbors of u but which are still unknown to u. They are given a score $\widehat{r_{u,j}}$ as showed below, and the most relevant ones get recommended with a top-N approach.

$$\widehat{r_{u,j}} = \frac{\sum_{v \in \text{KNN}(u)} sim(u, v) \cdot \mathbf{R}_{v,j}}{\sum_{v \in \text{KNN}(u)} sim(u, v)}$$

The item-based version (denoted IB) [41] computes similarities between items instead of users. Each candidate item is ranked using the following score:

$$\widehat{r_{u,j}} = \frac{\sum_{i \in P_u} sim(j,i) \cdot \mathbf{R}_{u,i}}{\sum_{i \in P_u} sim(j,i)}$$

where P_u is the profile of u, *i.e.* the set of items rated by u. The best scoring items are then recommended to u.

The item-based algorithm is often considered more efficient and has been the core feature of the Amazon recommendation system [30].

Model-based. This set of techniques rely on the build of a model to predict the user preferences, and include diverse approaches such as matrix factorization [28] or Bayesian and neural networks [36, 48].

We focus our review on matrix factorization (MF), a powerful technique popularized during the Netflix Prize competition and that is still one of the tools used in the Netflix recommendation system [20]. It was also used for candidate generation in the YouTube video recommender but has since been replaced by a similar approach based on neural networks [14].

The goal is to factorize the user-item matrix, *i.e.* find two matrices vectors $\mathbf{U}$ and $\mathbf{J}$ of sizes respectively $n \times K$ and $m \times K$, where K is a parameter denoting the number of hidden features, so that we have $\widehat{\mathbf{R}} = \mathbf{U} \cdot \mathbf{J}^\mathsf{T} \sim \mathbf{R}$ as illustrated in Figure 5. The score prediction is then performed as follows: $\widehat{r_{u,j}} = \widehat{\mathbf{R}}_{u,j}$

Several approaches were proposed to get a factorization such as singular value decomposition (SVD, SVD++), stochastic gradient descent (SGD) or alternating-least-squares (ALS). In our experimental evaluation, we use the last two as they give better results. SGD is a variant of the classical gradient descent algorithm where only a random subset of gradients is considered at each iteration to avoid expensive computations. Regarding ALS, we consider the variant proposed by [49] (ALSWR *i.e.* alternating-least-squares with weighted λ-regularization) which adds a Tikhonov regularization to the loss function minimized. The optimization problem solved is then the following:

$$(\mathbf{U}, \mathbf{J}) = \underset{\mathbf{U},\mathbf{J}}{argmin} \sum_{\substack{(u,j) \in \mathcal{U} \times \mathcal{I} \\ \mathbf{R}_{u,j} \neq \perp}} \left(\mathbf{R}_{u,j} - \left[\mathbf{U}\mathbf{J}^\mathsf{T} \right]_{u,j} \right)^2 + \lambda \cdot (\|\mathbf{U}\mathbf{\Gamma}_U\|_2 + \|\mathbf{J}\mathbf{\Gamma}_J\|_2)$$

where $\mathbf{\Gamma}_U$ (resp. $\mathbf{\Gamma}_J$) is a diagonal matrix containing the number of ratings of every user (resp. of every item). In practice, an approximate solution is found through an iterative process. Starting from a matrix $\mathbf{J}$ filled with average statistics for the first row and random values for the remaining

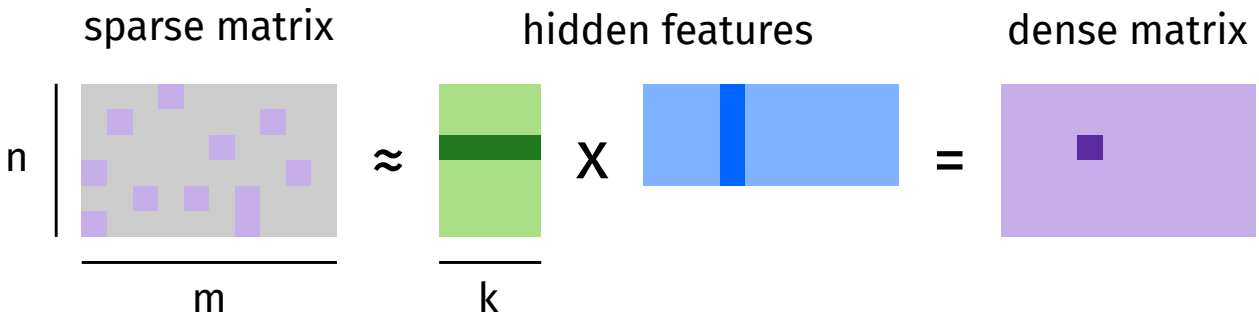


Figure 5: Factorization of a user-item matrix with k hidden features.

cells, the algorithm then performs the following at each step. Given $\mathbf{J}$, it finds $\mathbf{U}$ to minimize the objective function. Then, given $\mathbf{U}$, it updates $\mathbf{J}$ to minimize the objective function. The process is repeated until some convergence criteria is met, such as a bound on the improvement gained from one iteration.

Matrix factorization is often considered superior to classical neighborhood-based techniques since it is more memory-efficient and offers a greater recommendation quality. However, it requires an appropriate tuning to determine a relevant number of latent factors and to avoid overfitting (a common problem in machine learning where the model is too close to the training data and fails to provide reliable predictions).

3.3.3 Hybrid

It is possible to use content-based, collaborative filtering and various other approaches together. Such systems are called hybrid [10] and aim at combining the strength of different techniques while eliminating their individual weaknesses.

For example, algorithms augmented with demographic information or item characteristics in a content-based fashion are possible solutions to the cold start problem of collaborative filtering methods as presented in [42].

3.4 Biclustering-based recommenders

Biclustering was first introduced in genomics to group the rows and the columns of a data matrix [13]. This paradigm gives a similar role to rows and columns which become in some way inseparable and exchangeable. Biclusters may form a partition or be overlapping. Biclustering have become beneficial in many applications, not only in genomics but also in other contexts where data have the same sample/feature structure.

For instance, in genomics, a value $\mathbf{R}_{i,j}$ can represent the expression level of a gene j under condition i. In document analysis, it can denote the number of occurrences of word i in document j, while in e-commerce it can express the rating of a user i about an item j. Therefore, the design of biclustering based recommenders is an interesting research topic to evaluate the relevance of such approaches and to determine if this domain can benefit from the extensive literature about biclustering applied to genomics, in particular regarding scalability and parameter tuning.

We now give an overview of the existing literature regarding the design of recommendation systems based on biclustering. For each approach, we give brief explanations as well as its strengths and limitations regarding the challenges we have identified at the beginning of Section 3.3. To the best of our knowledge, none of the biclustering-based recommendation algorithms available in the literature address privacy concerns.

We illustrate in Figure 6 the different choices made in terms of bicluster and biclustering definitions since they turn out to be very diverse. As detailed in Remark 1, some approaches consider local constraints (blue cells) or unconstrained biclusters (light blue cell), global optimization objectives (purple cells) or simply an exhaustive biclustering (light purple cell). A combination of both local and global constraints is sometimes adopted. Some papers build their own biclustering algorithm to solve the precise problem they define while others are more modular and use well-known algorithms from the bioinformatics community.

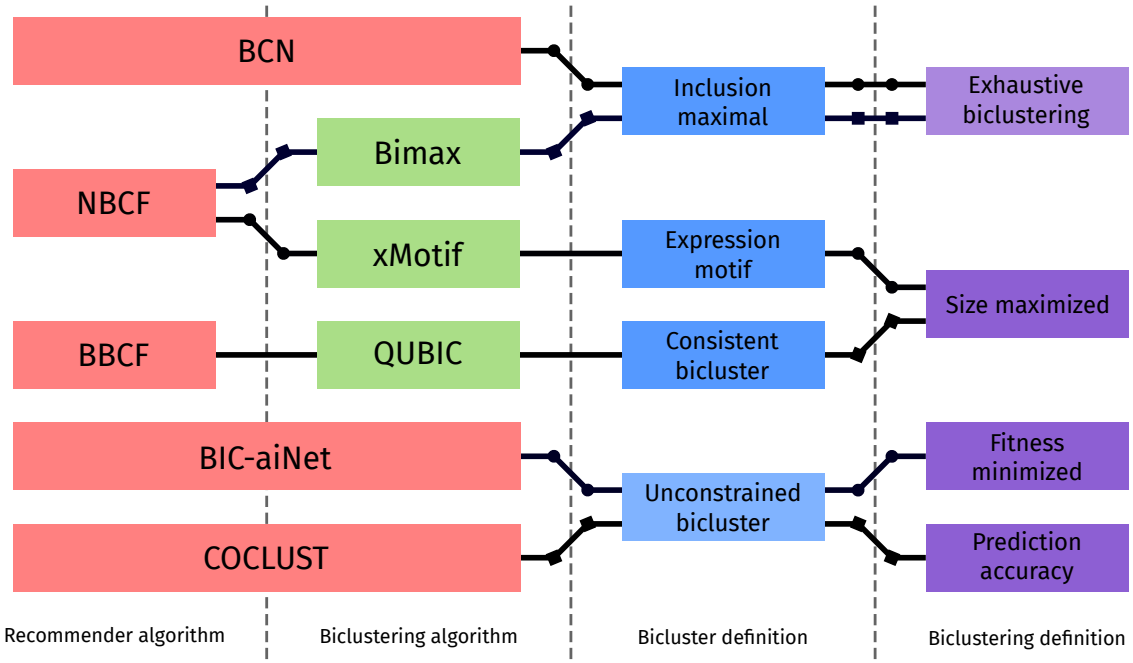


Figure 6: Overview of several biclustering-based recommendation algorithms in terms of constraints considered.

3.4.1 COCLUST

COCLUST [18] consists of a biclustering approach for collaborative filtering as an alternative to current model-based approaches that are highly effective but also computationally expensive. It considers a partitional biclustering as defined in Definition 7 which is build with the weighted Bregman biclustering algorithm from [5] that returns a locally optimal solution. The algorithm is adapted to not only compute the bicluster means, but to also take into account the biases induced by the specific user and item being considered , which results in more accurate predictions. The process starts with random partitions, and then optimize the assignments by considering alternatively the rows and the columns until some convergence criteria is met. The numbers of item clusters and user clusters in the checkerboard structure are given as input parameters.

The paper presents an incremental training method to cope with the appearance of new ratings in the context of real-time systems, as these may not be initially assigned to a bicluster in case of a new user or a new item. The total number of biclusters remains constant and depends on the initial parameters, which may turn out to be problematic in the long term. If the amount of data has significantly grown, one could wish to increase the number of biclusters to improve the prediction accuracy, but this requires to rerun the whole biclustering process, which is costly.

A parallel version of the algorithm is also provided to distribute the computation over multiple processes and improve its scalability. However, as mentioned in the paper, a good partitioning schema of the data is required to avoid unbalancing the load between the processes, which would harm the obtained speedup, especially when working with sparse matrices.

An experimental evaluation is conducted using several real-world datasets. It shows that COCLUST provides a high recommendation quality in terms of MAE score while outperforming the

state-of-the-art model-based approaches regarding time performances. Some clues are presented regarding the choice of the input parameters *i.e.* the number of biclusters, but we believe a more extensive analysis would be required to have a clear understanding of how this affects recommendation quality and of how to determine the best settings in a practical system.

3.4.2 NBCF

A new algorithm denoted NBCF is proposed by the authors of [45]. They perform a biclustering of the entire user-item matrix, using existing algorithms: either Bimax [39] in case of binary ratings which makes an exhaustive search of inclusion-maximal biclusters (see Definition 3), or xMotif [33] which searches for size-maximized expression motifs (see Definition 4). Given a user, NBCF determines the k nearest biclusters of this user according to a similarity metric, and then uses them to predict a score for the items for which the opinion of u is unknown.

The authors perform an extensive experimental evaluation and study the impact of bicluster sizes and of the value of k on the precision/recall results as well as on the average time to provide a recommendation. A comparison on real-world datasets shows that NBCF is significantly better than state-of-the-art item-based and user-based approaches regarding both recommendation quality and time performances. However, this approach is not compared to the popular model-based techniques such as matrix factorization. In particular, it is not clear whether NBCF suffers from the same scalability issues that are usually inherent to neighborhood-based approaches, since they induce many similarity computations.

A great strength of this approach is its modularity: one could easily plug another state-of-the-art algorithm to perform the biclustering which is significant given the amount of work performed in this field by the bioinformatics community. However, this also raises concerns about the impact of the biclustering algorithm choice on the actual performance of the back-end recommendation part. We can see in the paper that better results are obtained with xMotif than with Bimax but conclusions are not easy to draw since the latter operated on a binary version of the data unlike the former. Thus, we believe these concerns would require a thorough analysis to be addressed.

3.4.3 BIC-aiNet

The authors of [12] present BIC-aiNet, a biclustering algorithm to provide recommendations in the context of binary ratings. They use an iterative heuristic search to perform a biclustering that minimizes an objective fitness function as defined in Definition 8. To predict a score, a mean rating is computed within the most relevant bicluster corresponding to the user and that contains the considered item.

An experimental evaluation using a real-world dataset is conducted, where results claim that BIC-aiNet performs better in terms of precision/recall and MAE than several other algorithms including NBCF from [45]. However, the paper does not give any insights about how to choose the value of the many parameters such as the number of biclusters and others that appear in the fitness computation, and the impact they may have on the recommendation quality. The scalability of this approach is also not analyzed, which makes it unclear whether it could handle large amounts of data with reasonable time performances.

3.4.4 BCN

More recently, [3] proposes BCN, a new approach that makes use of an exhaustive biclustering where biclusters are then ordered to form a complete lattice. Given a user u, it first looks for the smallest bicluster C that contains u. It explores all the neighborhood biclusters of C in the lattice to find candidate items, ranks them by computing a similarity-based score and recommends the most relevant ones.

Experiments performed on real-world datasets suggest that BCN gives better results than existing state-of-the-art algorithms in terms of recommendation quality. The authors however mention that BCN performs particularly well for sparse data while being slightly outperformed when working with dense datasets. They also show that BCN is efficient in terms of scalability. One strength is that, contrary to model-based approaches, BCN does not need to re-train a model when new ratings appear, which makes it more effective under dynamic settings. Another is that no parameter tuning is required in BCN.

However, because of the partial order chosen to obtain a complete lattice, this approach works with binary data. A first discretization step is required when the ratings are not binary values, for instance using a threshold, which means a lot of valuable information must be discarded. It is not clear how BCN could be adapted to accommodate non-binary data.

3.4.5 BBCF

The authors of [44] presents an algorithm denoted BBCF that is built upon NBCF described earlier. They first perform a biclustering of the user-item matrix using QUBIC [29], which finds size-maximized coherent biclusters as detailed in Definitions 5 and 9. They use the same metric as NBCF to determine a set of k biclusters that are jointly the most similar to the considered user. These biclusters are then merged to obtain a sub-matrix, which is used to provide recommendations in a classical item-based KNN fashion [41].

BBCF is evaluated with real-world datasets, and shows very good performances compared to item-based and model-based approaches regarding recommendation quality and throughput. The impact of different parameters such as the number k of nearest neighbors is analyzed, although determining a general trend seems to remain difficult. Surprisingly, BBCF is not compared to NBCF despite being a direct extension of its core idea.

Like NBCF, BBCF is modular in terms of biclustering algorithm used, which raises the same concerns regarding the impact this choice has on performances. The authors also state that the biclustering process is usually computed offline. In the case of a dynamic system, they mention that the search of an effective method to incorporate new ratings without having to re-run the whole biclustering, which is a costly process, is still to be addressed.

3.5 Evaluation of recommenders

The evaluation of recommendation systems is a delicate process that still lacks standardized guidelines as stated by the authors of [40] with a blatant example: they found out that very few results published in the two previous ACM Conference on Recommender Systems (RecSys) were reproducible. This is often due to one or a combination of the following reasons:

(i) The source code of the evaluation is not publicly released and not enough details are provided to re-implement it from scratch.

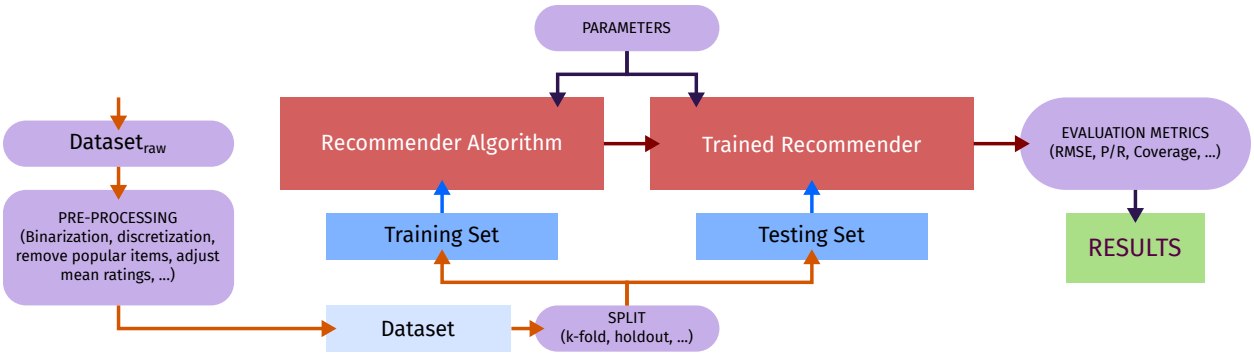


Figure 7: Offline evaluation of a recommender algorithm. Rounded purples boxes highlight the different elements where user choices may impact the final results.

(ii) The datasets used are proprietary or not available anymore.

(iii) Not enough details are provided regarding the experimental setup (data pre-processing, value of parameters, protocols to compute the evaluation metrics, etc).

Since the relevance of a recommendation seems to be a very subjective notion that differs from one individual to another, it may appear appropriate to perform an evaluation with the implication of real-humans in the loop, for example through A/B testing as YouTube does in [14]. However, this requires to have access to a significant poll of users or to own a real-world recommendation system like GroupLens Research does with the MovieLens recommender. Such setup is not easily available, and researchers often fall back on offline evaluation as illustrated in Figure 7, with the use of a dataset obtained through real-world data crawling or by synthetic generation. The dataset is split into a training set, used to train the algorithm, and a testing set where the algorithm predict scores that are then compared to the ground truth through various metrics. Because the results may be dependent on the actual split of the dataset, cross-validation consists in several rounds with different training and testing sets. A common variant is K-fold cross-validation, where the dataset is split into K parts of roughly the same size. There are K rounds where each part gets to be the testing set while the $K-1$ others are merged to form the training set. Results are then aggregated over the rounds to get final results.

Many metrics were considered over the years [25] but were initially focused on measuring accuracy. More recent work emphasize the importance of new metrics such as diversity, serendipity and coverage [26, 31].

We believe it is still unclear which metrics accurately model the user experience and should be considered to perform a relevant evaluation of a recommendation system. The authors of [6] show that a single metric such as precision (a well-known metric from the information retrieval field) can be computed in various ways and lead to different results. They present different methodologies to compute the set of candidate items to be recommended to a given user. We give further details about these in Section 4 where we describe our experimental setup.

RMSE and precision are two metrics related to accuracy, but may exhibit no clear correlation as presented in [15] since they are designed to evaluate different tasks (prediction for RMSE, recommend good items for precision). The authors of [15] also study the bias introduced by the existence of highly popular items in the dataset which constitute easy recommendations. They build a new version of the dataset with these items removed to better measure the personalization

power of a recommender algorithm.

More recently, [47] has implemented and run 21 algorithms to obtain rankings with various metrics. The correlation between those rankings are then analyzed to identify clear guidelines to deal with the sparsity of datasets and the choice of a cut-off for metrics from the information retrieval field such as precision and recall.

4 Experimental Setup

Our first contribution is an experiment evaluation of biclustering-based techniques. We thus implement the following algorithms: COCLUST, BBCF, NBCF and BCN. We could not come up with a working implementation of the BIC-aiNet algorithm due to the lack of details in the paper. Despite some messages exchanged with their authors, we were unable to obtain more information or the original source code, resulting in the exclusion of BIC-aiNet from our evaluation.

We now describe our experimental setup. We first present the framework in which we have implemented the algorithms listed above. We then give details about the metrics we compute and the datasets we use. Finally, we list the reference methods we consider to compare results.

4.1 Framework

We use Apache Mahout[1], a Java framework for the development of scalable machine learning applications. In particular, several state-of-the-art recommendation algorithms such as matrix factorization or user-based k-nearest neighbors methods are already implemented, which motivated our choice since these constitute interesting references for our experimental comparison. The addition of new algorithms can be performed by implementing the generic recommender interface.

We build a modified version of Apache Mahout 0.13.1 to add the support of K-fold cross-validation and to have better control over the evaluation protocol as well as the evaluation metrics. The modified version of Apache Mahout with the implementation of all the biclustering-based recommendation algorithms we consider, as well as the code used for evaluation are publicly available[2,3].

All our experiments are run within a Docker container (to ease the reproducibility of our results) based on Ubuntu 18.04.1 LTS with Java 8, on two servers with 48 logical cores (resp. 32) and 188 GB (resp. 126 GB) of RAM.

4.2 Evaluation metrics

In this work, we use 5-fold cross-validation, repeat each experiment several times to consider the average results. The K folds, and thus the training and testing sets, are built as follows: for each user u, we consider its profile as a vector $(j_k)_k$ containing the items rated by u, which order is obtained with a random shuffle. The cell (u, j_k) of the user-item matrix (with $\mathbf{R}_{u,j_k}$ as rating) is mapped to the fold No. $k \mod K$.

We now give a description of the different metrics we consider, which can be organized in three categories: accuracy, coverage and performance.

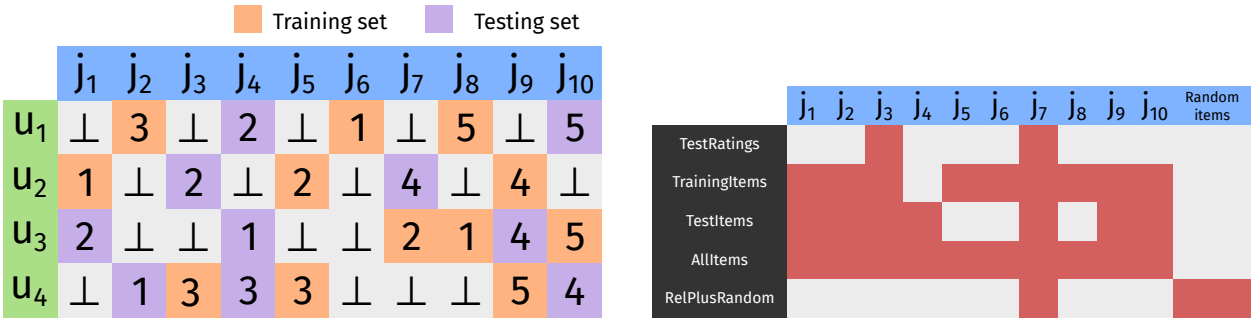
4.2.1 Accuracy

The accuracy of a recommendation system is a major challenge since the goal is to provide relevant recommendations to the users. The first set of metrics we consider is composed of Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). They are based on the idea that the recommender should be able to accurately predict the missing ratings in the user-item matrix. More

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

[2] `https://github.com/fdemoor/mahout/tree/branch-0.13.0`

[3] `https://github.com/fdemoor/binder`



(a) Training and testing sets

(b) Candidate items for user u_2 ($th = 3$)

Figure 8: Candidate items selection methodologies for metrics from the information retrieval field.

precisely, for each element in the training set $\mathcal{T}_t$, we compute the error between the predicted score and the ground truth. Since this is an error score, the lower the better. RMSE relies on a quadratic model while MAE simply consider an absolute value. The errors are then summed to obtain a final score, as follows:

$$\text{MAE} = \frac{1}{|\mathcal{T}_t|} \sum_{(u,j)\in\mathcal{T}_t} |\widehat{r_{u,j}} - \mathbf{R}_{u,j}| \qquad \text{RMSE} = \left(\frac{1}{|\mathcal{T}_t|} \sum_{(u,j)\in\mathcal{T}_t} (\widehat{r_{u,j}} - \mathbf{R}_{u,j})^2 \right)^{1/2}$$

It also has become common to use classification metrics from the information retrieval field such as precision and recall. It is argued that they are more related to the user experience since they are shown a list of recommendations in a practical system, and the actual predicted ratings remain hidden. The evaluation is performed as follows: the trained recommender tries to provide each user u with a list $L_u^N \subseteq C_u$ of N recommendations where N is called the cut-off and C_u if the set of candidate items for u. Considering the testing set, each user is associated a set Rel_u of known relevant items (which requires a like-threshold th, for instance 3 for 1-5 stars ratings):

$$Rel_u = \{j \mid (u,j) \in \mathcal{T}_t \wedge \mathbf{R}_{u,j} \geq th\}$$

Intuitively, precision measures the proportion of relevant recommendation (*i.e.* how many recommended items were actually liked) while recall measures the coverage of the user interests (*i.e.* how many liked items were recommended). Results are then aggregated over all users, and the higher the better.

$$\text{Precision@}N = \frac{1}{|\mathcal{U}|} \sum_{u\in\mathcal{U}} \frac{|L_u^N \cap Rel_u|}{|L_u^N|} \qquad \text{Recall@}N = \frac{1}{|\mathcal{U}|} \sum_{u\in\mathcal{U}} \frac{|L_u^N \cap Rel_u|}{|Rel_u|}$$

The authors of [6] describe several methodologies to compute C_u that we detail below and are illustrated in Figure 8.

(i) *TestRatings* (TR)
$C_u = \{j \mid \exists v \in \mathcal{U} \text{ st. } (v,j) \in \mathcal{T}_t\} \setminus \{j \mid (u,j) \in \mathcal{T}_T\}$ *i.e.* the set of item rated by u in the testing data.

(ii) *TestItems* (TeI)
$C_u = \{j \mid (u, j) \in \mathcal{T}_t\}$ *i.e.* the set of items existing in the testing set, except those for which u has a known rating.

(iii) *TrainingItems* (TrI)
$C_u = \{j \mid \exists v \in \mathcal{U} \text{ st. } (v, j) \in \mathcal{T}_T\} \setminus \{j \mid (u, j) \in \mathcal{T}_T\}$ *i.e.* the set of items existing in the training set, except those for which u has a known rating.

(iv) *AllItems* (AI)
$C_u = \mathcal{I} \setminus \{j \mid (u, j) \in \mathcal{T}_T\}$ *i.e.* the set of all items in the dataset, except those for which u has a known rating.

(v) *RelPlusRandom* (RPR)
$C_u = Rel_u \cup RandomItems_u$ *i.e.* the set of relevant items plus some random items irrelevant to u.

Because of the way we build the training and testing, the three methodologies TrainingItems, TestItems and AllItems are quite similar. They suffer from some biases which make it hard to interpret the absolute values of the metrics: it is likely that u will be recommended some items for which we do not possess the ground truth value, those are considered as a "miss" while they may actually be a "hit" if we had perfect knowledge about the interests of u. This however do not apply to TestRatings where we consider only items where the ground truth is known. The set of candidate items is however much smaller and do not reflect a real-world scenario. The RelPlusRandom methodology is based on the idea that the relevant items should be recommended and not the random ones. It relies on the assumption that the random items added are irrelevant recommendations, which is hard to ensure since the preferences of u are not fully known. This approach can be turned into many variants such as the strategy used in [15] where a single relevant item is blended with 1000 random items.

In our evaluation, we consider the TestRatings (since there is no bias introduced by the lack of perfect knowledge) and TrainingItems (because it better reflects realistic settings) methodologies.

4.2.2 Coverage

Recommender systems were traditionally evaluated with accuracy metrics, but there has been some work towards the use of new metrics beyond accuracy, such that serendipity, diversity or coverage [26]. We consider here the latter that gives indications of the amount of users and items covered by the recommender algorithm in the system.

We consider again the list L_u^N of recommendation provided to some user u with a cut-off N, and we denote by $\text{UserCoverage} : \{0 \dots N\} \to \mathbb{N}$ the function where:

$$\text{UserCoverage@}N(k) = \frac{|\{u \in \mathcal{U} \mid |L_u^N| \geq k\}|}{n}$$

Two quantities appear to be particularly of interest: the proportion of users for which the system was able to provide at least one recommendation, and those where all recommendation asked were given (denoted respectively ReachAtLeastOne and ReachAll).

$$\text{ReachAtLeastOne@}N = \text{UserCoverage@}N(1) \qquad \text{ReachAll@}N = \text{UserCoverage@}N(N)$$

Dataset	*Domain*	*Rating range*	*#Users*	*#Items*	*#Ratings*	*Density*	*Release*
ML-100K	Movies	1-5 stars	943	1,682	100,000	6.30%	1998
ML-1M	Movies	1-5 stars	6,040	3,706	1,000,209	4.47%	2003
BookCrossing	Books	0-10 stars	103,022	327,488	1,133,161	0.0034%	2004
Jester-1	Jokes	$[-10; +10]$	73,421	100	4,136,360	56.34%	2003

Table 2: Characteristics of the different datasets considered. Density refers to the percentage of non-empty cells in the matrix.

The size of the list of recommendations given to some user u can be lower than N for a couple of reasons. First, u may have a profile so large that there does not exist N items unknown to u in the system. However, in practical systems such as e-commerce industries where catalogs span millions of products, this is highly unlikely to occur.

The second reason is that the recommender algorithm was unable to determine N items considered as relevant for u, which happens if the set of candidate items computed has less than N elements. Such situation could indicate a limitation of the algorithm if we consider that there ought to exist N items u doesn't know and could be interested in. Since it depends on the system catalog and the subjective tastes of u, it is difficult to determine of this is a reasonable assumption.

We can also notice that an algorithm could easily counteract user coverage issues by filling incomplete lists with popular or sponsored items.

ReachAtLeastOne and ReachAll refer to coverage from a user perspective, but we can also compute the proportion of the item catalog that was covered by the recommendations:

$$\text{ItemCoverage@}N = \frac{|\bigcup_{u \in \mathcal{U}} L_u^N|}{m}$$

A low value expresses that many items get recommended to the same users. This can happen if these are popular products, but it can also indicate that the recommender algorithm fails at capturing the specific interests of each individual, thus falling back on generic recommendations

4.2.3 Time performance

We also measure the time that it takes to generate N recommendations for every user in the system, along with the training part of the algorithm (such as the build of a model or a kNN graph). We use the `nanoTime` method from the Java `System` class to compute the elapsed time. We obtain the following metric, where t denotes a function that measures the duration of an operation:

$$\text{Time@}N = t(\text{training}) + \sum_{u \in \mathcal{U}} t\big(\text{generation of } L_u^N\big)$$

We report the average results over the different folds.

4.3 Datasets

The characteristics of the different datasets used for evaluation are summarized in Table 2.

We first consider the MovieLens datasets released by GroupLens, a research lab at University of Minnesota which owns a non-commercial only movie recommendation system [24]. These are

Algorithm	*Parameters*
Random	-
IAVG, IUAVG	-
UBKNN	Similarity metric sim, number of nearest neighbors k
IB	Similarity metric sim
MF SGD	Numbers of iterations n_i, of hidden features K
MF ALSWR	Numbers of iterations n_i, of hidden features K, regularization parameter λ
COCLUST	Numbers of user clusters k, of item clusters l, of iterations n_i
NBCF, BBCF	Number of nearest biclusters k, biclustering algorithm
BCN	Like-threshold th

Table 3: List of recommendation algorithms considered and their parameters.

popular datasets used by the academic community for the evaluation of recommender algorithms but also by the industry or for education purposes. There are four MovieLens datasets consisting of 100K, 1M, 10M and 20M ratings that were released respectively in 1998, 2003, 2009 and 2015. These constitute in a list of tuples containing movie ratings collected through the MovieLens system over the years. While the first datasets only contain 5-star integer ratings, as of the 3rd version of MovieLens, the latest two datasets (ML-10M and ML-20M) also include half-star ratings, which doubles the set of possible values.

We also consider the BookCrossing dataset which consists in a set of book ratings [50]. This dataset was obtained through a 4-week crawl from August to September 2004 of the Book-Crossing community. The latter is a book club launched in 2001 where people leave books in public places so that they can be picked and read by others. They then have also the possibility to leave 10-star ratings of the books through the community website. In the dataset, each book is identified by its ISBN-10 value. We perform some pre-processing to remove from the dataset the invalid ISBN values, and we convert all identifiers to ISBN-13 values to work only with integers since ISBN-10 can contain a "X" as check digit. Our pre-processing step removed 16,619 ratings from the original file, which results in a dataset containing 1,133,161 ratings.

Jester is an online joke recommendation system, were users can rate jokes. Two datasets Jester-1 and Jester-2 are available with data collected respectively from April 1999 to May 2003 and from November 2006 to May 2009 [19]. These are particularly interesting for two main reasons. First, ratings are continuous in $[-10; +10]$, contrary to the other datasets we considered where ratings are discrete using stars pattern. Secondly, the number of items is respectively 100 and 150 which results in a very dense matrix, while the MovieLens or BookCrossing datasets have a low density.

4.4 Algorithms and settings

We list the recommenders we consider along with their parameters in Table 3 and now give some further details.

Baselines. Apache Mahout provides a class to run a random recommender. Predictions are performed by returning a random score between the minimum and the maximum existing rating in the dataset. We consider this approach in our evaluation to provide a baseline score for the various metrics considered, along with the simple methods IAVG and IUAVG.

Neighborhood-based. We use the generic implementations of user (UBKNN) and item-based (IB) recommenders in Apache Mahout, with two similarity metrics: the cosine angle and the Pearson correlation. For IB, we also consider the log-likelihood to compute item similarities.

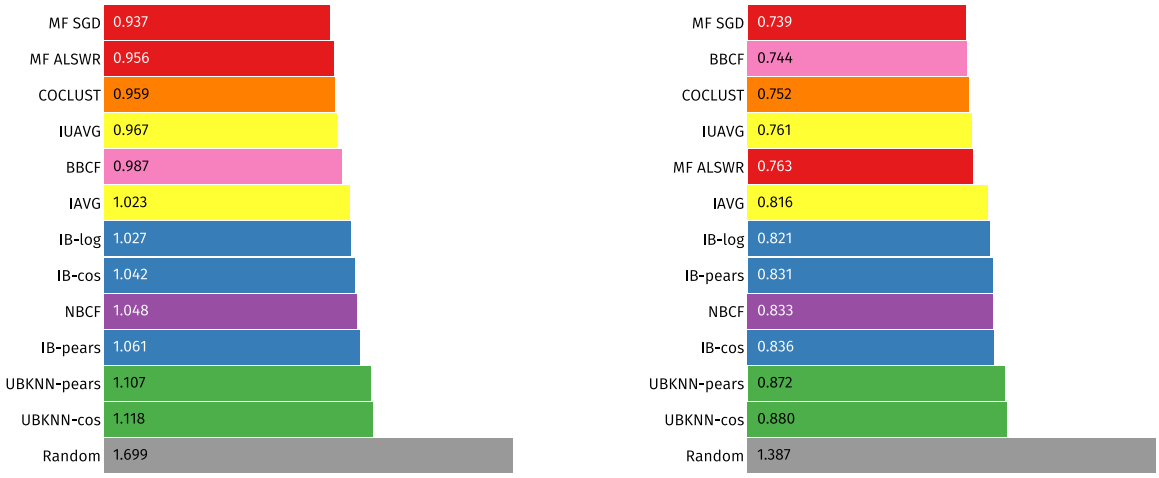
Model-based. We use the implementations of MF SGD and MF ALSWR in Apache Mahout. Regarding MF ALSWR, experiments conducted on the Netflix Prize dataset in [49] show that while the result improves as the number of iterations grows, the gain in terms of RMSE score becomes tiny at some point. As a consequence, we fix the number of iterations to 25. Regarding the regularization parameter λ and the number of hidden features K, we process with an empirical approach to determine relevant settings. Intuitively, higher K often yields better recommendation quality since it enables to grasp more accurately the preferences of the users, at the cost of more expensive computations.

Biclustering-based Recommenders We implement the different biclustering-based algorithms presented in our modified version of Mahout. We set $n_i = 25$ and try various number of user and item clusters for COCLUST to select the best results yielded. We run NBCF and BBCF using QUBIC (consistency level of 0.95, 100 biclusters output, and 100% overlap allowed) to perform the biclustering of the user-item matrix before the search of the nearest biclusters, with $k = 20$. The BCN algorithm was implemented using the Java lattice library from [22] with a single level to find neighboring biclusters. Since BCN considers binary ratings, we do not compare the prediction errors with the other approaches, and we only present precision and recall for this algorithm.

5 Evaluation Results

5.1 Accuracy

We give the RMSE and MAE results in Figure 9 for the ML-100K dataset. As expected, we see that matrix factorization performs better than classical memory-based approaches based on similarity metrics. Surprisingly, IAVG and IUAVG yield good results despite being very simple methods. Overall, the best performing approach within the biclustering spectrum is COCLUST which is close to standard matrix factorization algorithms.

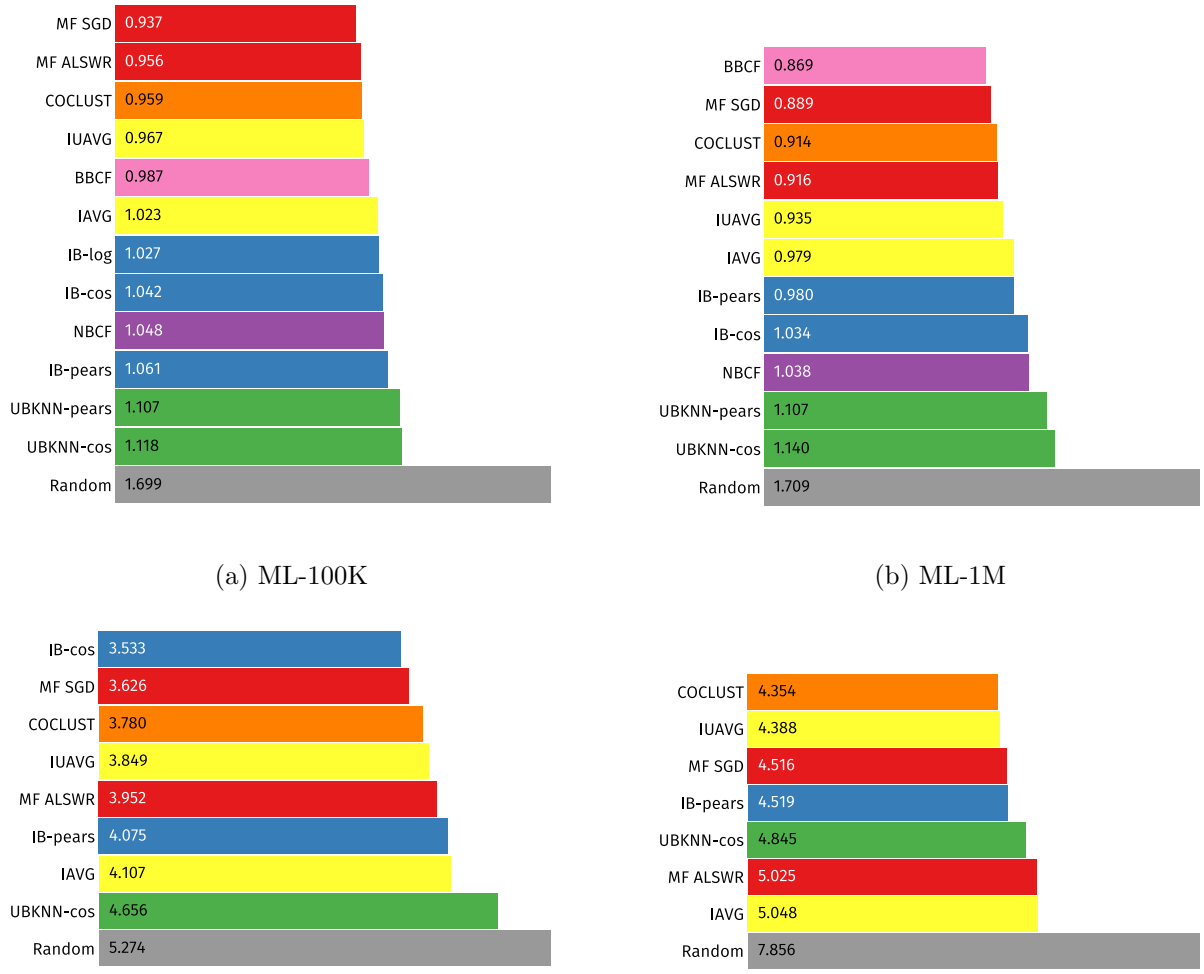


(a) RMSE (b) MAE

Figure 9: Score prediction with the ML-100K dataset.

The results for the other datasets are shown in Figure 10 where we only present the RMSE score since MAE exhibits similar trends. We can see here the impact of sparsity. In the BookCrossing dataset which is very sparse (0.0034%), the item-based approach is performing really well while others like matrix factorization and COCLUST are less efficient. This is expected since the user-item is mainly composed of empty cells *i.e.* there is less training data available. For the Jester-1 dataset which has a much higher density (56.34%), COCLUST is performing the best, closely followed by IUAVG.

We also run some experiments with a pre-processing on the ML-100K dataset where only the long-tail of items is considered. As illustrated in Figure 11a, one third of the ratings only affect 6.8% of the existing items. We thus have a group of highly popular items that concentrate many ratings. [15] states that recommending these items are then easy recommendations and are not of great relevance. The goal is rather to grasp the specific interests of each individual. Since these items introduce a bias, we remove them from the dataset and run again a prediction evaluation. Results are given in Figure 11b. Overall, we obtain similar trends except for the BBCF algorithm which falls down at the bottom of the ranking, and for IB which has a slightly improved accuracy. We deduce the BBCF was mainly able to provide accurate predictions for the highly popular items: they were likely to appear is more biclusters returned by the QUBIC algorithm, resulting in more



(a) ML-100K

(b) ML-1M

(c) BookCrossing

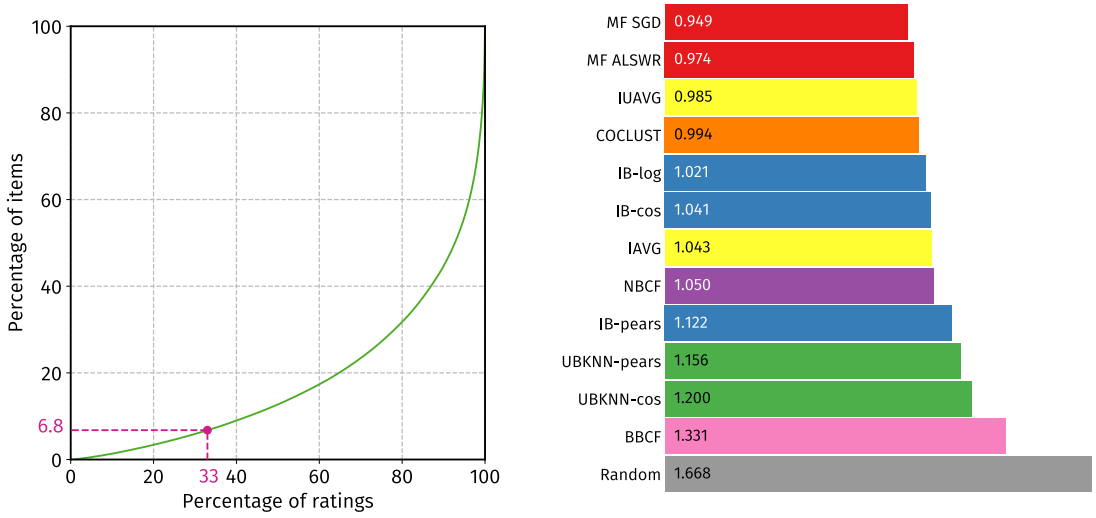
(d) Jester-1

Figure 10: RMSE score for various datasets.

information to compute a score.

Precision and recall are given in Figure 12. For the TrainingItems methodology, we see that matrix factorization and the user-based KNN approach obtain good results. COCLUST is outperformed by the simple IUAVG algorithm. NBCF and particularly BBCF yield high scores for precision, but these are mitigated by their poor results for coverage as shown later: BBCF was unable to provide recommendations to more than 80% of the users. Due to the high number of candidate items, our implementation of BCN was not efficient enough to end in a reasonable amount of time, we thus do not present results for this methodology.

For the TestRatings methodology, NBCF and BBCF perform here worse than all the other approaches, including BCN which has good precision and recall.



(a) Long-tail

(b) RMSE

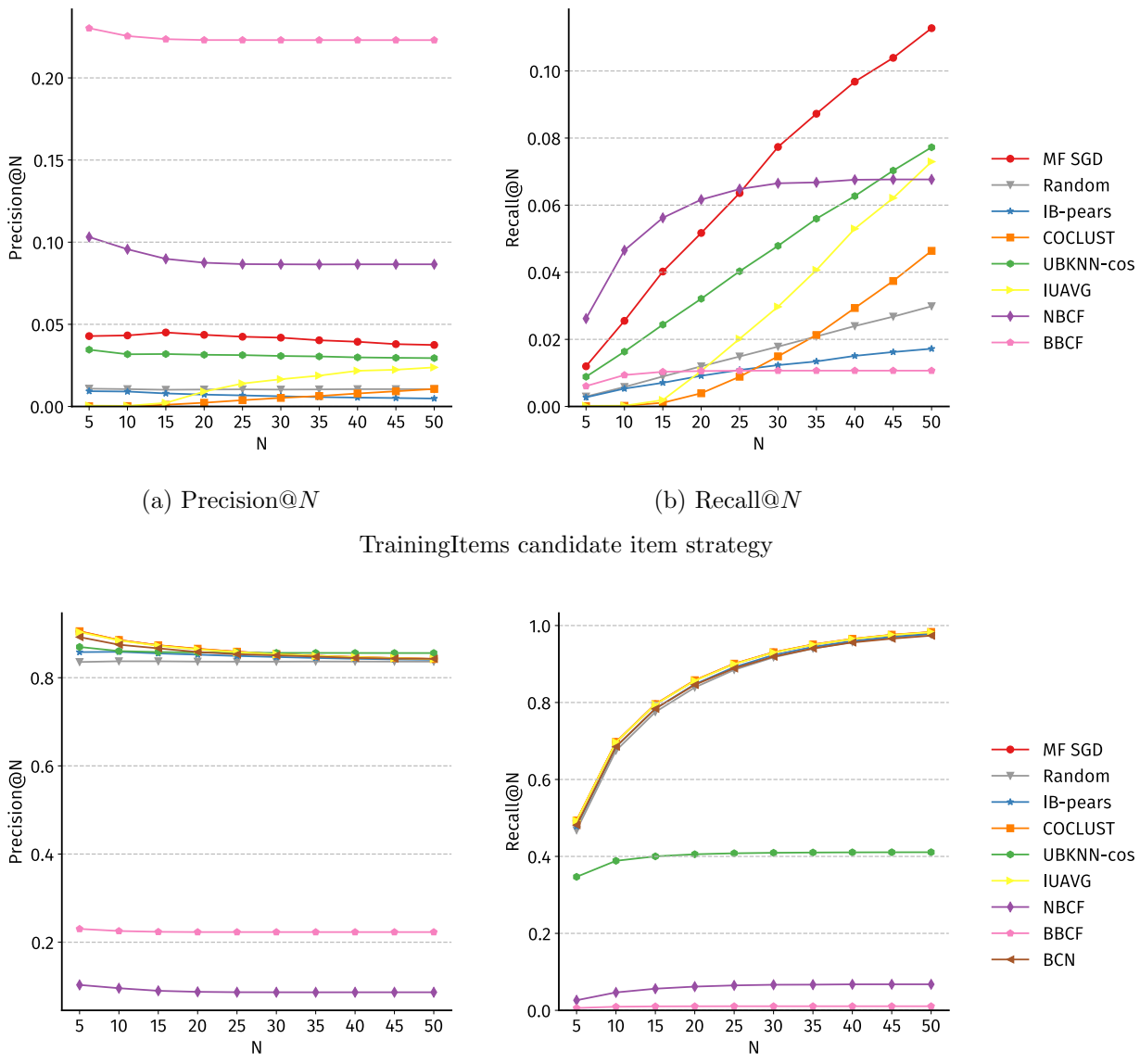
Figure 11: RMSE with the most popular items removed from the ML-100K dataset.

5.2 Coverage

Coverage results are given in Figure 13 for the TrainingItems and Testratings methodologies.

We see that IB achieves a high item coverage which is consistent since this particular approach searches for relations between items. COCLUST also gives good results with TrainingItems, contrary to MF and IUAVG which seem to recommend only a very small subset of items. BBCF and NBCF obtain the worse results since they are highly limited by the set of items covered by the biclustering computed. If an item does not appear in a bicluster, then no score can be predicted and the item cannot be recommended. This problem also arises with BCN, but can be mitigated by exploring a greater part of the lattice to cover more items.

We notice a similar trend for user coverage: all approaches perform well except the biclustering-based approaches such as NBCF, BBCF and BCN. Because COCLUST considers a checkerboard biclustering, it does not suffer from this issue, each item and user can be mapped to a corresponding bicluster. BBCF exhibits very poor results because of the way it predicts ratings: if the user does not appear in the submatrix formed by the merge of the nearest biclusters, then no recommendations can be provided. This issue could be mitigated with several options. First, we tried to grow the submatrix to include ratings from the user, but it worsened accuracy results and did not increase significantly the coverage. Another option is to generate more biclusters to increase the percentage of the user-item matrix covered, but this can prove costly depending on the biclustering algorithm used. Finally, we can consider a hybrid system that falls back to another algorithm when BBCF fails to provide enough recommendations, and we can indeed obtain promising results as showed in Figure 14 where MF SGD is used as fallback solution. The approach BBCF + MF obtains a ReachAtLeastOne@N of 1 and slightly improved Precision / Recall results than matrix factorization alone.



(a) Precision@N (b) Recall@N

TrainingItems candidate item strategy

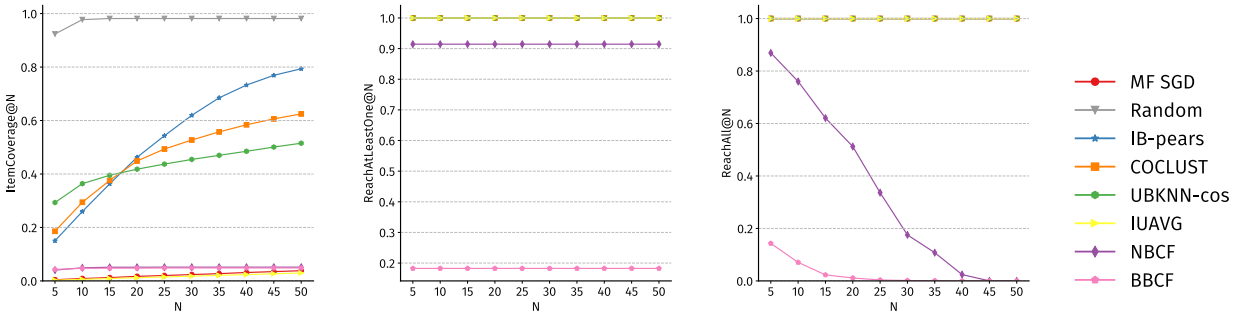
(c) Precision@N (d) Recall@N

TestRatings candidate item strategy

Figure 12: Precision / recall with the ML-100K dataset and N recommendations provided.

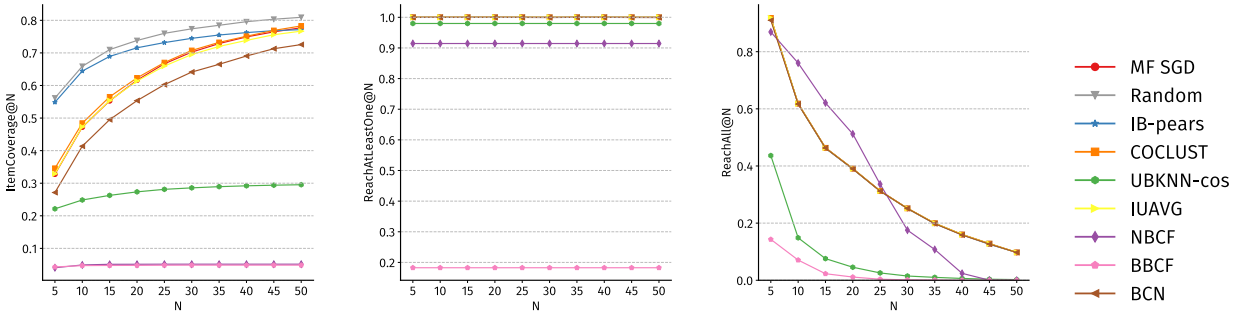
5.3 Time performance

We now report time performance results in Figure 15. The best performing approach is IUAVG since it is a simple method that requires only to compute averages over the user-item matrix. Neighborhood-based algorithms (UBKNN, IK) are more expensive because of the similarity computations and the search of close users or items. COCLUST exhibits the same order of magnitude than MF SGD: they both use an iterative algorithm to optimize their model. Other biclustering-



(a) ItemCoverage@N (b) ReachAtLeastOne@N (c) ReachAll@N

TrainingItems candidate item strategy



(d) ItemCoverage@N (e) ReachAtLeastOne@N (f) ReachAll@N

TestRatings candidate item strategy

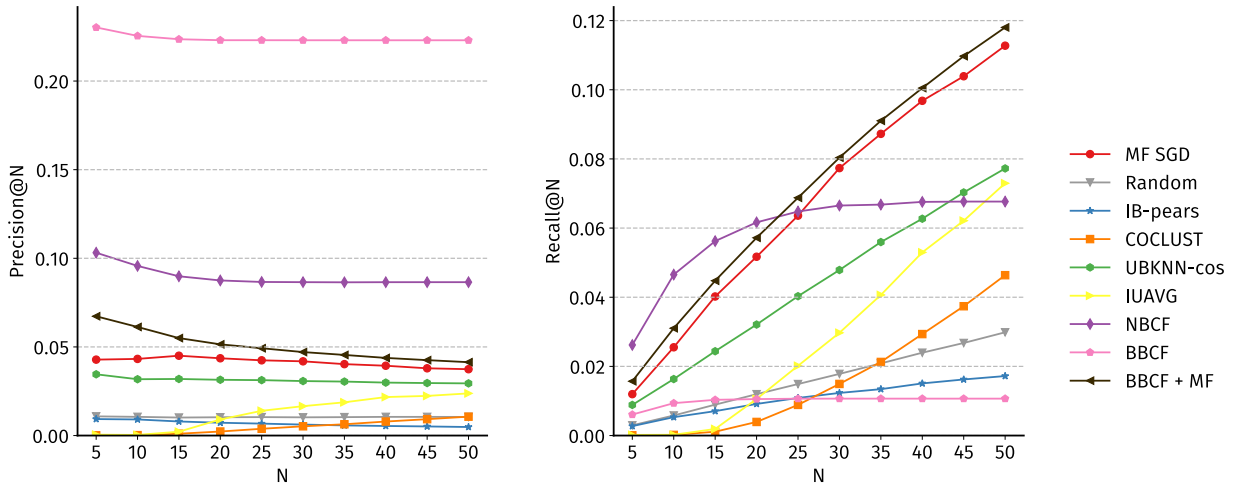
Figure 13: Coverage with the ML-100K dataset and N recommendations provided.

based approaches such as NBCF, BBCF or BCN expose much higher computation time, which is due to the biclustering part. Where COCLUST considers a partional biclustering which is cheaper to build, these consider a broader definition of the biclustering problem which is NP-hard. Despite the use of heuristics, the iterative search of biclusters (xMotif, QUBIC) or the exploration of a complete lattice (BCN) remains a costly operation that fails to achieve the same efficiency as standard other approaches.

5.4 Evaluating metrics

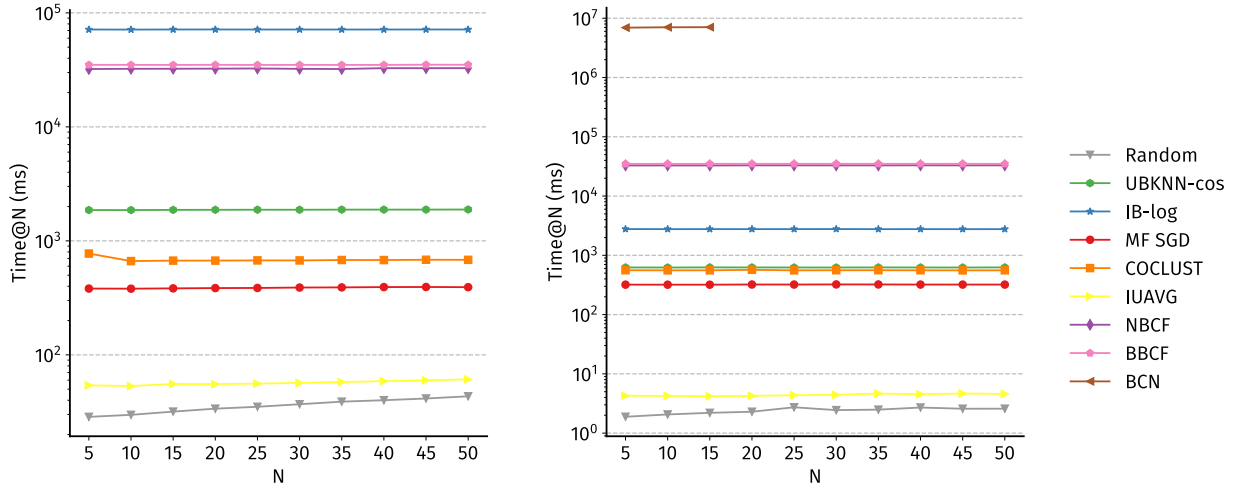
Since we observe varying results with different candidate items selection strategies, we perform some additional experiments to better how evaluation metrics are correlated.

We get inspiration from the work of [47] and run experiments with a list of 12 algorithms presented earlier in this report (namely: Random, IAVG, IUAVG, UBKNN-cos, UBKNN-pears, IB-cos, IB-pears, IB-log, MF SGD, MF SVD++, MF ALSWR and COCLUST). For a given metric, we look at the results and rank the algorithms. We then use Kendall's tau [27] correlation coefficient, which gives a score between -1 (opposite ranking) and +1 (same ranking). Let $(x_i)_{0 \leq i < 0}$ and $(y_i)_{0 \leq i < 0}$ be the ranks of the n algorithms. A pair (i, j) is said to be concordant if $(x_i > x_j \wedge y_i > y_j) \vee (x_i < x_j \wedge y_i < y_j)$ *i.e.* if algorithm i is better than algorithm



(a) Precision@N (b) Recall@N

Figure 14: Precision / recall with the TrainingItems methodology and the ML-100K dataset, with a hybrid system that falls back on MF SGD to fill in the missing recommendations of BBCF.



(a) TrainingItems (b) TestRatings

Figure 15: Time@N with the ML-100K dataset and N recommendations provided.

j in the first ranking, it is also in the second. However, a pair is said to be discordant if $(x_i > x_j \wedge y_i < y_j) \vee (x_i < x_j \wedge y_i > y_j)$ *i.e.* if algorithm i is better than j in the fist ranking, it is worse in the second. The standard formula is then:

$$\tau = \frac{\#\text{concordant pairs} - \#\text{discordant pairs}}{\#\text{pairs}}$$

We consider the τ_B version that also takes into account pairs that neither concordant nor

discordant. We give a few results we have obtained in Figure 16.

We can see in Figure 16a that RMSE results are heavily correlated with precision computed with TestRatings but far less when it comes to other strategies. As stated before, other methodologies suffer from the bias of imperfect knowledge. It then appears consistent that RMSE exhibits results closer to precision with TestRatings than with TrainingItems. We also notice that MAE is surprisingly correlated with precision and recall no matter the strategy, which questions the relevance of a quadratic approach to measure the prediction error.

In Figures 16b and 16c, the value of N (the number of recommendations provided) also exhibits some importance. For both TestRatings and TrainingItems, Precision@10 and Precision@50 are poorly correlated, while results seem to be more stable past a certain value. We thus agree with the conclusions of [47]: deeper cut-offs should be preferred.

We include more metrics in Figure 16d and notice that accuracy and coverage metrics seem quite independent or even negatively correlated, especially with TrainingItems. Results are a bit different with TestRatings where precision, user coverage and item coverage give similar rankings of the algorithms. We however believe they are less significant since the set of candidate items is much smaller which makes it harder to properly evaluate coverage.

We have drawn interesting insights from these additional experiments, and we believe some further work could be achieve in this respect to answer the following questions. To which extent are the many existing metrics correlated, depending on the way they are computed and the characteristics of the dataset considered? Which metrics are more appropriate to properly evaluate the user experience and thus validate a recommendation algorithm as better than another?

5.5 Incremental training

As stated earlier, today's systems are highly dynamic and have to face many constraints: new users are likely to join the system, new items are regularly added to the catalog, and ratings may shift over time. Because of that, it is important for a recommendation algorithm to be able to quickly adapt to these changes, also known as incremental training.

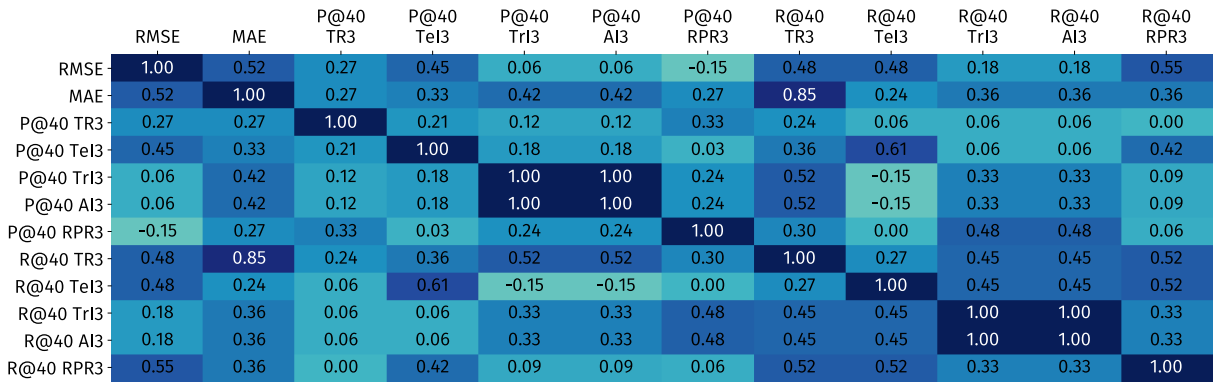
In the case of IUAVG, the cost is low if the averages are computed with an adequate data structure that keeps track of the number of elements summed. For other approaches such as IB, UKBNN and MF, it is much more expensive since the changes can have a huge impact on similarity values or the latent features coefficients. Biclustering-based systems such as BBCF, NBCF and BCN suffer from the same drawback: the biclustering algorithm must be run again.

However, COCLUST and its partional biclustering approach have a great advantage in this category. Changes are likely to affect only a few biclusters, which means we simply need to update some cluster assignments and run a few iterations to converge again, contrary to MF where the changes may affect the entire factorization and lead to a rebuild of the whole model. COCLUST thus naturally offers a cheap incremental training process that we believe is of great interest for real-time settings.

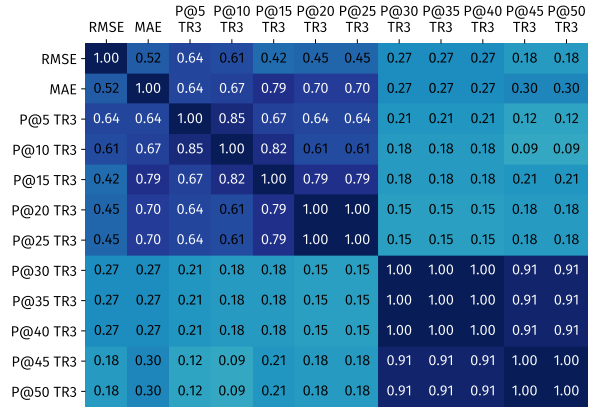
5.6 Discussion

We give a qualitative summary of the results we obtained with the ML-100K dataset in Table 4.

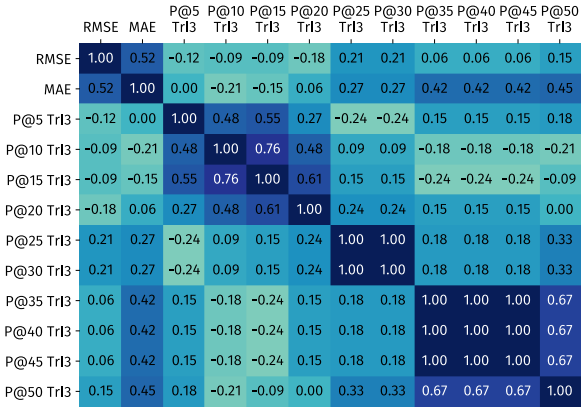
We observe that the partitional biclustering approach of COCLUST can compete with matrix factorization techniques while having a great advantage when it comes to incremental updates. The strength of COCLUST lies in a fast iterative training where all biclusters are built at the



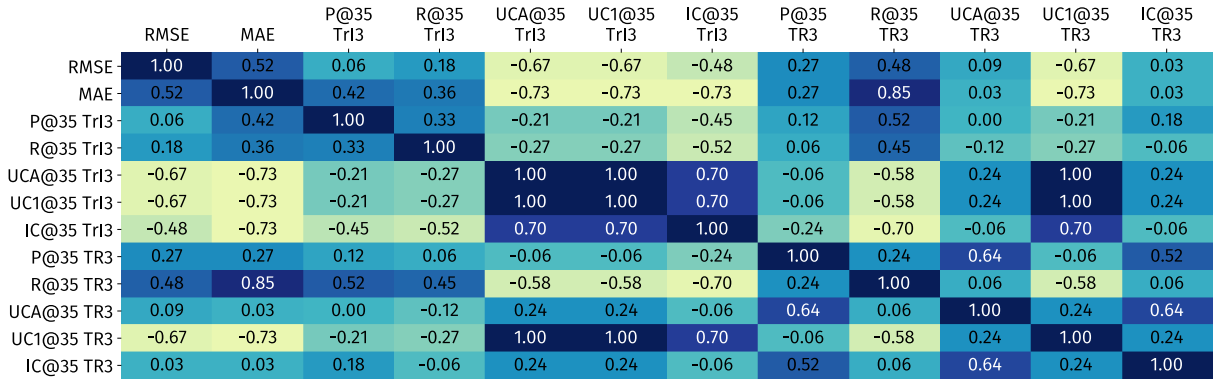
(a) RMSE, MAE, Precision@40 and Recall@40 for several methodologies.



(b) Precision@N with TestRatings.



(c) Precision@N with TrainingItems.

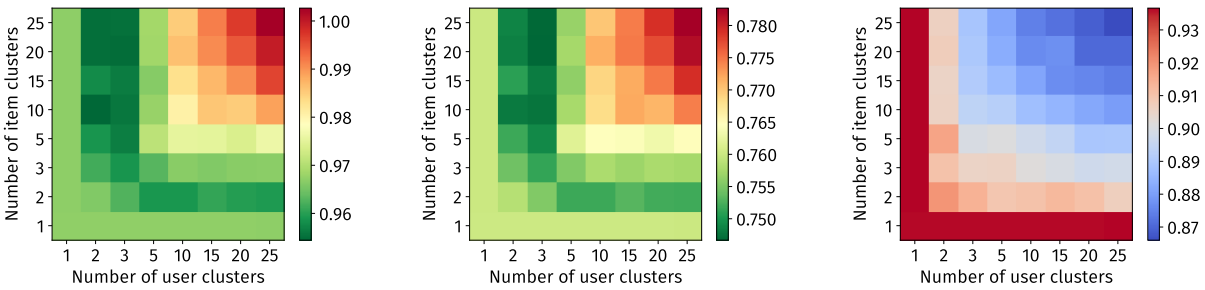


(d) RMSE, MAE, Precision@35, Recall@35, ReachAll@35 (UCA), ReachAtLeastOne@35 (UC1) and ItemCoverage@35 (IC) with TrainingItems and TestRatings.

Figure 16: Kendall's tau correlation coefficient results with the ML-100K dataset ($th = 3$ to compute relevant items).

Algorithm	*Prediction*	*Precision*	*User coverage*	*Item coverage*	*Time*	*Incremental update*
IUAVG						
UBKNN						
IB						
MF SGD						
COCLUST						
NBCF						
BBCF						

Table 4: Qualitative summary of results for the ML-100K dataset (greener/longer is better).



(a) RMSE (b) MAE (c) Training error

Figure 17: Impact of the number of user and item clusters in COCLUST, with 30 iterations and the ML-100K dataset.

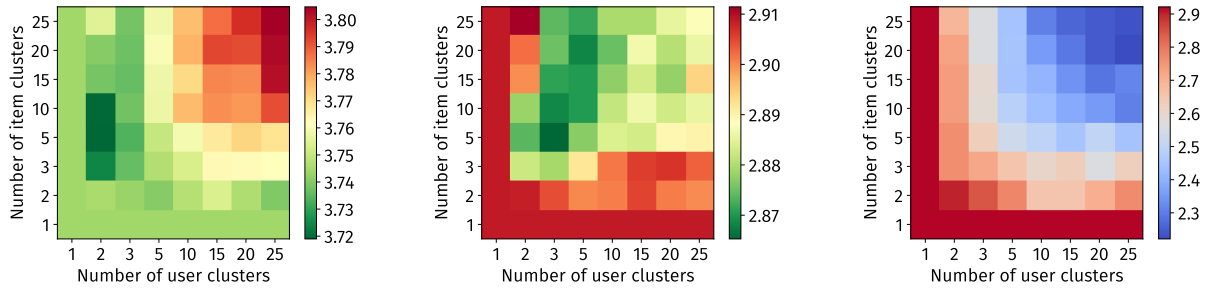
same time. Each user/item couple is associated with a single biclusters which can be retrieved in constant time once the model build. This ensures a high coverage since no user is left over. With efficient data structures, the prediction is a $O(1)$ operation which is even better than the $O(K)$ complexity of the prediction task for matrix factorization techniques. Tuning the parameters (the numbers of user and items clusters) is no different from the search of the ideal number of hidden features, although it is slightly more cumbersome since this is a two-dimensional space. We show in Figures17 and 18 how the parameters of COCLUST effect the prediction score on the testing set, but also the prediction error on the training set. We see that the training error and the accuracy scores do not follow the same pattern. High numbers of clusters do not yield the best RMSE and MAE results despite a lower training error, which can be explained by overfitting. Another reason might be that biclusters are then smaller: predictions are computed with local data where some more global knowledge would also be beneficial.

The Ml-100K and BookCrossing datasets exhibits similar behaviors. We suspect this is also the case for the ML-1M dataset, but we would need to run experiments with even greater numbers of cluster to obtain a similar plot. The Jester-1 dataset however shows very different results because of its density and its low number of items (only 100). That explains why too many item clusters yield a poor accuracy unless there are enough user clusters to get groups with similar interests.

Other biclustering-based approaches such as NBCF or BBCF suffer from coverage issues: be-

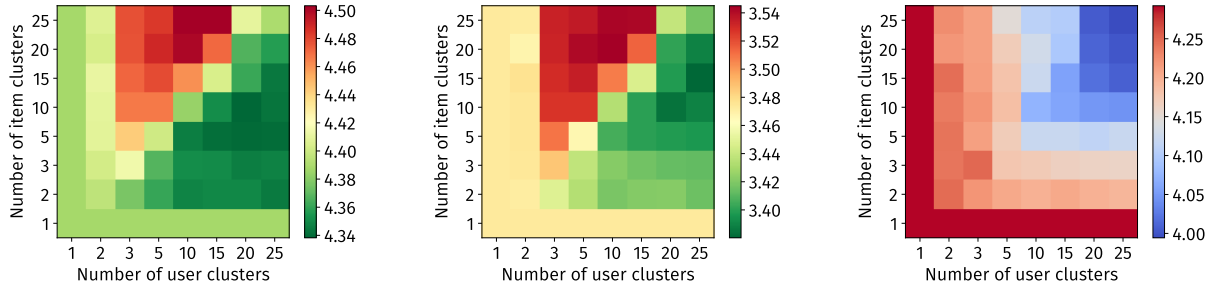
(a) RMSE (b) MAE (c) Training error

Results with the ML-1M dataset.



(d) RMSE (e) MAE (f) Training error

Results with the BookCrossing dataset.



(g) RMSE (h) MAE (i) Training error

Results with the Jester-1 dataset.

Figure 18: Impact of the number of user and item clusters in COCLUST, with 30 iterations and several datasets.

cause some items or users are not covered by the biclustering, it is not possible to recommend these items or to provide recommendations to those users. However, results suggest there is still a small set of users for which these approaches are of interest. We thus believe, despite their high computational cost, that they could be beneficial to use in the context of a hybrid system running multiple algorithms.

6 Extending the COCLUST Algorithm

We now present some ongoing work as well as some ideas to improve the COCLUST algorithm. We give explanations in specific subsections for the ease of readability, but all the modifications we propose can be combined into a single recommender algorithm.

6.1 Self-tuning parameters

Since it can be hard to identify ideal values for the number of user clusters and items clusters in COCLUST (an extensive empirical exploration of the domain is computationally expensive and cumbersome), we wish to propose a self-tuning methodology that we describe in Algorithm 1. We start with initial parameters and we navigate in the domain by keeping values that appear more relevant. To this end, we divide the dataset into a training set and a validation set, and we use the RMSE score.

Algorithm 1 Self-tuning of k and l parameters for COCLUST

Input: Dataset $\mathcal{D}$

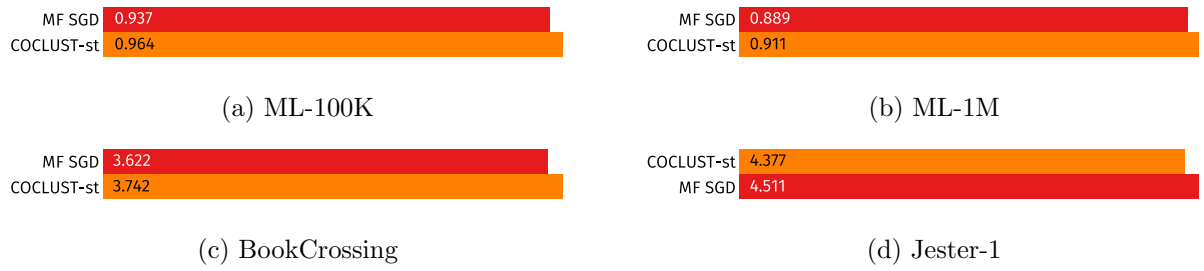
Divide $\mathcal{D}$ into a training set $\mathcal{T}_T$ and a validation set $\mathcal{T}_V$
Set $k = 1$ and $l = 1$
Build a model $M = \text{COCLUST}(k, l, \mathcal{T}_T)$
Compute $r = \text{RMSE}(M, \mathcal{T}_V)$
repeat
    Increase k and/or l by a random step to obtain $\tilde{k}$ and $\tilde{l}$ ▷ Explore the domain
    Build a model $\tilde{M} = \text{COCLUST}(\tilde{k}, \tilde{l}, \mathcal{T}_T)$
    Compute $\tilde{r} = \text{RMSE}(\tilde{M}, \mathcal{T}_V)$
    if $\tilde{r}$ better than r **then**
        $k = \tilde{k}$, $l = \tilde{l}$, $r = \tilde{r}$ ▷ The new parameters seem better so let us keep them
until convergence or maximum number of iterations reached
return k and l

We present preliminary results we obtained in Figure 19. The input parameters for the COCLUST recommender are computed with the protocol previously described. We see that results are very similar with those of Figures 9 and 10 where k and l were tuned by hand. For some datasets, we even obtained better results, which is not that surprising: tuning by hand is a cumbersome iterative process and the self-tuning protocol was able to find better values for the parameters. We can also compare with Figures 17 and 18 to notice that the RMSE score remained into areas with reasonable values.

These results constitute an interesting first step for the methodology we have proposed. We however believe further work in necessary for this self-tuning protocol to achieve maturity. We now discuss its limits and present several ideas of improvements.

Heuristics could be used to drive the exploration of the domain, perhaps by exploiting precise characteristics of the dataset considered, which would first require investigating if some correlations can be identified with the ideal values of k and l. For instance, if the items are movies that can be clustered in three main types, we can expect l to be at least greater than three.

In our protocol, we use RMSE as a starting point, but other metrics or a combination of different metrics may be more relevant. The ideal way to decide if new parameters are better than the former



(a) ML-100K

(b) ML-1M

(c) BookCrossing

(d) Jester-1

Figure 19: RMSE score with the ML-100K dataset.

is tightly correlated with the best way to evaluate recommendation systems, and can also be highly dependent on the purpose of the recommender as an application in a real-world context.

Another idea worth investigating is to get inspiration from the different techniques used in clustering such as X-means [38] or G-means [23] which try to infer the ideal value of k for the k-means problem.

The computation cost of this approach can be controlled with a maximum number of iterations, which means the final complexity is the original complexity of the COCLUST algorithm times a constant of the user choice. We thus believe the time performance would still be of the same order of magnitude than matrix factorization where the number of hidden features also has to be determined, either through an exhaustive search of a similar heuristic-based approach.

6.2 Adding regularization

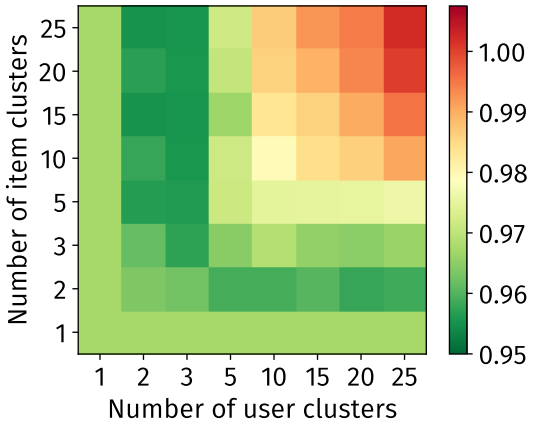
We mentioned earlier that the higher number of biclusters, the worse the results with COCLUST, as illustrated in Figure 17. The numbers of clusters that yield the best results are surprisingly small given the characteristics of the ML-100K dataset. For instance, the ideal values we can deduce from the figure are somewhere along $k = 3$ and $l = 20$ which gives around 300 users and 80 items per bicluster with a uniform distribution.

We propose to add a regularization term to the optimization problem of Definition 7 to prevent overfitting and be able to benefit from higher numbers of clusters. Given a user u, our intuition is to spread the profile of u over all the different item clusters. If all the profile belongs to the same item cluster, then highly accurate recommendations would be provided within the same cluster, but predictions are likely to be poor for other item clusters, which damages coverage and serendipity. A similar reasoning is possible for item profiles. In the end, we propose the following, where the second term compute the maximum number of non-empty cells in a bicluster.

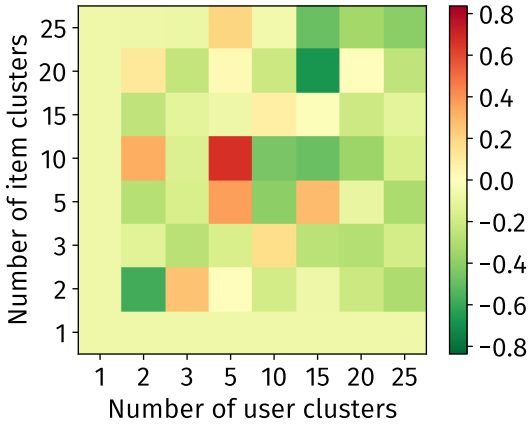
$$\min_{(\rho,\gamma)} \sum_{\substack{(u,j)\in\mathcal{U}\times\mathcal{I} \\ \mathbf{R}_{u,j}\neq\perp}} (\mathbf{R}_{u,j} - \widehat{r_{u,j}})^2 + \lambda \cdot \max_{\substack{1\leq g\leq k \\ 1\leq h\leq l}} |\{\mathbf{R}_{u,i} \neq \perp \mid \rho(u) = g \wedge \gamma(j) = h\}| \tag{1}$$

We also investigate another regularization term which is simply the standard deviation of the prediction errors to avoid scattered distributions.

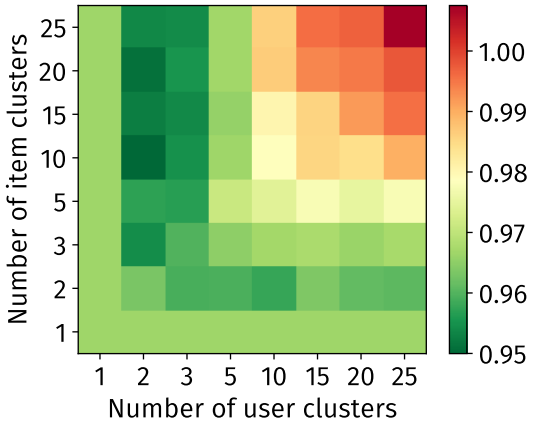
$$\min_{(\rho,\gamma)} \sum_{\substack{(u,j)\in\mathcal{U}\times\mathcal{I} \\ \mathbf{R}_{u,j}\neq\perp}} (\mathbf{R}_{u,j} - \widehat{r_{u,j}})^2 + \lambda \cdot \sigma\left([\mathbf{R}_{u,j} - \widehat{r_{u,j}}]^2\right)_{\substack{(u,j)\in\mathcal{U}\times\mathcal{I} \\ \mathbf{R}_{u,j}\neq\perp}} \tag{2}$$



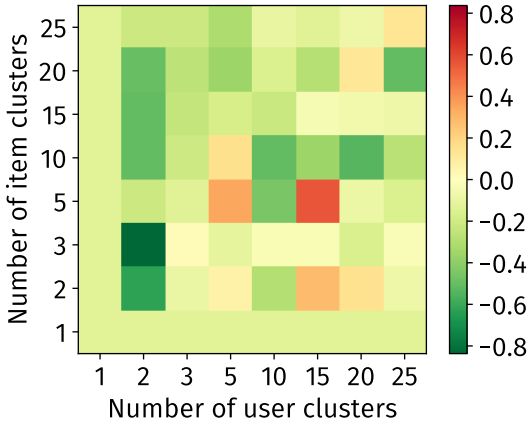
(a) RMSE (COCLUST-R-Max)



(b) Increase percentage (negative values mean COCLUST-R-Max is better than COCLUST).



(c) RMSE (COCLUST-R-Std)



(d) Increase percentage (negative values mean COCLUST-R-Std is better than COCLUST).

Figure 20: RMSE of COCLUST with regularization (COCLUST-R-Max / $\lambda = 0.009$, COCLUST-R-Std / $\lambda = 1$) and percentage improvement compared to COCLUST.

In both propositions, we have the original term along with a new term controlled by the regularization parameter λ.

We adapt the COCLUST algorithm to solve this new optimization problems and run some experiments. We denote this new recommender by COCLUST-R-Max (Equation 1) and COCLUST-R-Std (Equation 2). We show some preliminary results in Figure 20 where we focus on the RMSE score. The increase percentage is computed as follows:

$$score = \frac{\text{RMSE w/o reg.} - \text{RMSE w/ reg.}}{\text{RMSE w/o reg.}} \times 100$$

Negative values thus mean that COCLUST-R performs better than COCLUST. We see overall

that the addition of the regularization slightly improved the prediction accuracy, and that the error decrease is more apparent for higher numbers of clusters in the case of COCLUST-R-Max. Regarding COCLUST-R-Std, we get better improvements, but not with the same configurations which means we could maybe get better results by mixing the two approaches.

We plan to extend these results by devising a smarter regularization term. Our first attempts, despite being very naive, shows that there is room for improvements in this direction.

6.3 Nearest biclusters

A fallout of higher numbers of biclusters could be, as already stated earlier, that computations only use local data while a more global view would be necessary to obtain good predictions. In this respect, we propose, in cases where the matrix is partitioned in many small biclusters, to adapt the prediction formula of Definition 7 to also take into account nearby rows and columns of biclusters given some similarity metric. Different scores with several levels of neighborhood could be computed, and then aggregated with relevant weights to obtain a final prediction. We could for instance get inspiration from the "k nearest biclusters" approach of NBCF and BBCF algorithms to further investigate the potential benefits of this idea in the near future.

7 Conclusion

In this report, we have considered the field of recommendation system and the many challenges that are still to be addressed today. We have presented the most popular techniques that were investigated over the last three decades, and we have focused on biclustering, an approach inspired from work within the genomics community. Through an extensive experimental survey, we have compared some existing biclustering-based algorithms with the state-of-the-art methods.

We have found that partitional biclustering as used in COCLUST is an interesting approach that could lead to similar performances as matrix factorization, one of the leading techniques to provide recommendations. However, we believe COCLUST to have the potential for great improvements over MF in terms of incremental update which would allow the cope with the highly dynamic nature of many systems with temporal constraints. We have presented in Section 6 some ongoing work to provide improvements to the COCLUST algorithm.

Approaches that use a more general form of biclustering are subject to high computational cost, but are also able to grasp very accurately the preferences of specific users. They thus could be used as a complement to some other algorithms in a hybrid system. Because users are different, they are presented with personalized recommendations, but one could argue that it is only natural that the recommender algorithm itself should be personalized to each individual. In practice, most production system do not use a single approach but a plethora of algorithms with clever aggregation schemes to combine their output and present a final list to each user. For these reasons, we reckon biclustering-based algorithms yield promising results and should not be neglected.

Apart from the use of COCLUST in real-time systems and other approaches in hybrid systems, we also believe there is a need for standard guidelines to perform experimental evaluations of recommender systems. There exist many metrics, and even if we focus on a single metric such as precision, there are still many ways we can compute it, and they do not necessarily lead to similar conclusions. That is why we plan to extend our preliminary results of Section 5.4 by considering more datasets and more metrics (including coverage, performance and others not presented in this report such as diversity and novelty [26]). We also would like to consider results from an online evaluation scenario (through A/B testing) to better understand how results obtained from offline settings compare with real-world environments.

References

[1] https://netflixprize.com.

[2] B. Abdollahi and O. Nasraoui. Using Explainability for Constrained Matrix Factorization. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 79–83. ACM, 2017.

[3] F. Alqadah, C. K. Reddy, J. Hu, and H. F. Alqadah. Biclustering neighborhood-based collaborative filtering method for top-n recommender systems. *Knowledge and Information Systems*, 44(2):475–491, 2015.

[4] X. Amatriain and J. Basilico. Netflix Recommendations: Beyond the 5 stars, 2012.

[5] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. In *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 509–514, 2004.

[6] A. Bellogin, P. Castells, and I. Cantador. Precision-oriented Evaluation of Recommender Systems: An Algorithmic Comparison. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, pages 333–336. ACM, 2011.

[7] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proceedings of the 2007 ACM Conference on Recommender Systems, RecSys 2007, Minneapolis, MN, USA, October 19-20, 2007*, pages 9–16. ACM, 2007.

[8] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132, 2013.

[9] A. Boutet, F. De Moor, D. Frey, R. Guerraoui, A.-M. Kermarrec, and A. Rault. Collaborative Filtering Under a Sybil Attack: Similarity Metrics do Matter! In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*, pages 466–477. IEEE Computer Society, 2018.

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

[11] J. F. Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR 2002: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 11-15, 2002, Tampere, Finland*, pages 238–245. ACM, 2002.

[12] P. A. D. d. Castro, F. O. d. França, H. M. Ferreira, and F. J. V. Zuben. Applying Biclustering to Perform Collaborative Filtering. In *Seventh International Conference on Intelligent Systems Design and Applications, ISDA 2007, Rio de Janeiro, Brazil, October 20-24, 2007*, pages 421–426. IEEE Computer Society, 2007.

[13] Y. Cheng and G. M. Church. Biclustering of Expression Data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, August 19-23, 2000, La Jolla / San Diego, CA, USA*, pages 93–103. AAAI, 2000.

[14] P. Covington, J. Adams, and E. Sargin. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, pages 191–198. ACM, 2016.

[15] P. Cremonesi, Y. Koren, and R. Turrin. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 39–46. ACM, 2010.

[16] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 269–274. ACM, 2001.

[17] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 257–260. ACM, 2010.

[18] T. George and S. Merugu. A Scalable Collaborative Filtering Framework Based on Co-Clustering. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 625–628. IEEE Computer Society, 2005.

[19] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4:133–151, 2001.

[20] C. A. Gomez-Uribe and N. Hunt. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Management Inf. Syst.*, 6(4):13:1–13:19, 2016.

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

[22] C. Guérin, K. Bertet, and A. Revel. An efficient Java implementation of the immediate successors calculation. In *Concept Lattices and their Applications*, pages 81–92, Oct. 2013.

[23] G. Hamerly and C. Elkan. Learning the K in K-Means. *Advances in Neural Information Processing Systems*, 17, 2004.

[24] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, Dec. 2015.

[25] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.

[26] M. Kaminskas and D. Bridge. Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems. *ACM Trans. Interact. Intell. Syst.*, 7(1):2:1–2:42, Dec. 2016.

[27] M. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30, 1938.

[28] Y. Koren, R. M. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer*, 42(8):30–37, 2009.

[29] G. Li, Q. Ma, H. Tang, A. H. Paterson, and Y. Xu. QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research*, 37(15):e101–e101, 2009.

[30] G. Linden, B. Smith, and J. York. Industry Report: Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Distributed Systems Online*, 4(1), 2003.

[31] A. Maksai, F. Garcin, and B. Faltings. Predicting Online Performance of News Recommender Systems Through Richer Evaluation Metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 179–186. ACM, 2015.

[32] F. McSherry and I. Mironov. Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 627–636. ACM, 2009.

[33] T. M. Murali and S. Kasif. Extracting Conserved Gene Expression Motifs from Gene Expression Data. In *Proceedings of the 8th Pacific Symposium on Biocomputing, PSB 2003, Lihue, Hawaii, USA, January 3-7, 2003*, pages 77–88, 2003.

[34] A. Narayanan and V. Shmatikov. Robust de-anonymization of large datasets (how to break anonymity of the Netflix prize dataset). 2008. *University of Texas at Austin*, 2008.

[35] Y. Okada, W. Fujibuchi, and P. Horton. A biclustering method for gene expression module discovery using a closed itemset enumeration algorithm. *IPSJ Digital Courier*, 3:183–192, 2007.

[36] M.-H. Park, J.-H. Hong, and S.-B. Cho. Location-based recommendation system using bayesian user's preference model in mobile devices. In *International conference on ubiquitous intelligence and computing*, pages 1130–1139. Springer, 2007.

[37] M. J. Pazzani and D. Billsus. Content-Based Recommendation Systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321, pages 325–341. Springer, 2007.

[38] D. Pelleg and A. W. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734. Morgan Kaufmann Publishers Inc., 2000.

[39] A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, 2006.

[40] A. Said and A. Bellogín. Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 129–136. ACM, 2014.

[41] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[42] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and Metrics for Cold-start Recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260. ACM, 2002.

[43] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 23-29 April 2006, Barcelona, Catalunya, Spain*. IEEE, 2006.

[44] M. Singh and M. Mehrotra. Impact of biclustering on the performance of Biclustering based Collaborative Filtering. *Expert Syst. Appl.*, 113:443–456, 2018.

[45] P. Symeonidis, A. Nanopoulos, A. N. Papadopoulos, and Y. Manolopoulos. Nearest-biclusters collaborative filtering based on constant and coherent values. *Inf. Retr.*, 11(1):51–75, 2008.

[46] N. T. Thong and L. H. Son. HIFCF: An effective hybrid model between picture fuzzy clustering and intuitionistic fuzzy recommender systems for medical diagnosis. *Expert Systems with Applications*, 42(7):3682 – 3701, 2015.

[47] D. Valcarce, A. Bellogín, J. Parapar, and P. Castells. On the Robustness and Discriminative Power of Information Retrieval Metrics for top-N Recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 260–268. ACM, 2018.

[48] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative Deep Learning for Recommender Systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[49] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Proc. 4th Int'l Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.

[50] C.-N. Ziegler, C.-N. , S. McNee, S. M, K. , J. A, L. , and G. . Improving recommendation lists through topic diversification. 2005.