UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

User-Specific Bicluster-based Collaborative Filtering

Miguel Miranda Garção da Silva

Mestrado em Ciência de Dados

Dissertação orientada por:
Prof. Doutor Sara Alexandra Cordeiro Madeira

2020

Agradecimentos

O caminho para a realização de uma tese de mestrado é composto por vários e diversos momentos. Inúmeros momentos de motivação e alegria, alguns de incerteza. Gostaria de agradecer a todas as pessoas que direta ou indiretamente me acompanharam durante estes momentos, e que, com esse apoio, tornaram possível a realização deste trabalho.

Primeiramente, agradeço à minha orientadora, Professora Sara Madeira. Começo por agradecer por me ter escolhido e confiado em mim para trabalharmos em conjunto neste projeto. Gostaria de agradecer também o papel vital que teve neste nosso trabalho, onde destaco a orientação, incentivo, disponibilidade e apoio que sempre transmitiu durante estes meses. Sinto-me grato por esta experiência, que me fez crescer e proporcionou a oportunidade de aprender e trabalhar com alguém que respeito e admiro.

Em segundo lugar, um obrigado à minha família, em especial aos meus pais, avô e Simões, que me transmitem, diariamente, os ideais e valores que me acompanham e moldam enquanto pessoa. Sinto-me eternamente agradecido por me darem todas as condições necessárias para que eu conseguisse estar nesta posição onde me encontro hoje.

Num tom de brincadeira, gostaria de agradecer a todos os meus amigos. Se dizem que somos uma combinação dos nossos amigos mais próximos, então eu tenho de ser perfeito. Duarte, amigo Engenheiro, o mais correto talvez seria destacar os momentos de trabalho, esforço e dedicação académica, mas sabes que o politicamente correto não é o meu forte, prefiro relembrar as nossas parvoíces. Lobo, amigo Data Scientist, incrível como partilhámos tantas experiências nos últimos anos, influenciaste-me e aprendi muito contigo, espero que tenha conseguido retribuir. Gil, amigo Hacker, não sei se alguma vez irás penetrar e deixar a tua marca no meu computador, mas tenho a certeza de que a mim já me marcaste. Paulo, amigo dos caracóis, sei que mal me referenciaste na tua tese, mas não faz mal, estás perdoado melhor amigo, não aconteceu nada, eu perdoo-te :). Obrigado a todos, acredito que foi apenas o início de uma grande amizade. Ou então não, e daqui a dois meses já somos desconhecidos (mais provável).

Termino agradecendo mais uma vez, a todos os que, mesmo que não mencionados, contribuíram para a realização deste trabalho, que me permite alcançar um grande objetivo na minha vida.

Dedico esta tese a todos vocês, que acompanham o meu percurso.

Resumo

Os sistemas de recomendação são um conjunto de técnicas e software que têm como objetivo sugerir itens a um determinado utilizador. Sugestões essas que têm como objetivo ajudar os utilizadores durante a tomada de decisão. O processo para uma tomada de decisão pode ser difícil, especialmente quando existe um enorme número de opções para escolher. Grandes empresas tiram partido dos sistemas de recomendação para melhorar o seu serviço e aumentar as suas receitas. Um exemplo é a plataforma de streaming Netflix que, utilizando um sistema de recomendação, personaliza os filmes ou séries destacados para cada cliente. As recomendações personalizadas normalmente têm como base os dados que as empresas recolhem dos utilizadores, que vão desde reações explícitas, por exemplo através avaliações do utilizador a produtos, a reações implícitas, examinando a forma como o utilizador interage com o sistema.

Uma das abordagens mais populares dos sistemas de recomendação é a filtragem colaborativa. Os métodos baseados em filtragem colaborativa produzem recomendações personalizadas de itens, tendo por base padrões encontrados em dados de uso ou avaliações anteriores. Os modelos de filtragem colaborativa normalmente usam uma simples matriz de dados, conhecida como matriz de interação U-I, que contém as avaliações que os utilizadores deram aos itens do sistema. Explorando os dados da matriz U-I, a filtragem colaborativa assume que, se um determinado utilizador teve as mesmas preferências que outro utilizador no passado, é provável que também venha a ter no futuro. Desta forma, os modelos de filtragem colaborativa têm como objetivo recomendar uma lista de N itens a um utilizador (denominado utilizador ativo), ou prever o rating que esse utilizador iria dar a um item que ainda não avaliou. Na literatura, os métodos de filtragem colaborativa são divididos em duas classes: os baseados em memórias e os baseados em modelos.

Os algoritmos baseados em memória, também conhecidos como algoritmos de vizinhança, usam toda a matriz U-I para realizar as tarefas de recomendação. Os dois principais métodos são conhecidos como “User-based” e “Item-based”. O User-based tenta encontrar utilizadores com preferências parecidas ao utilizador a que se pretende fazer recomendações e usa os dados dessa vizinhança de utilizadores similares para fazer as previsões ou recomendações. Por outro lado, os algoritmos Item-based utilizam os itens já avaliados pelo utilizador ativo, calculam a similaridade entre esses itens e o item que

se quer avaliar, construindo assim uma vizinhança de itens. A partir dessa vizinhança de itens, prevê-se uma futura avaliação do utilizador a esse mesmo item.

Apesar de os algoritmos de vizinhança obterem bom resultados de previsão e recomendação, apresentam duas grandes debilidades que limitam o seu uso em ambientes de recomendação do mundo real. Os dados de recomendação são normalmente de grandes dimensões e esparsos, isto é, com muitos valores em falta. Dada a complexidade resultante do facto de terem de comparar todos os utilizadores ou itens entre si, o que se traduz em n^2 comparações, torna-se impraticável o uso de algoritmos deste género em sistemas com grande quantidade de users e itens. Além disso, o facto de haver muitos valores em falta, faz que seja recorrente alguns utilizadores/itens terem pequenas vizinhanças.

Para tentar lidar com as fraquezas dos algoritmos baseados em memórias, surgiram os algoritmos baseados em modelos. Estas abordagens utilizam modelos que aprendem com os dados e reconhecem padrões para realizar as tarefas de filtragem colaborativa. Técnicas de redução de dimensionalidade como "Singular Value Decomposition" e "Latent Semantic Analysis" são agora as abordagens standard para reduzir a natureza esparsa da matriz de interação. Existem ainda abordagens baseadas em aprendizagem automática, como redes bayesianas, agrupamento de dados, entre outras. Estes modelos de redução de dimensionalidade, apesar de perderem informação que geralmente resulta em piores resultados em termos de previsão/recomendação, conseguem lidar com o problema da escalabilidade apresentado pelos modelos baseados em memória.

O agrupamento de dados, conhecido como "Clustering", é o processo que agrupa um conjunto de objetos de dados em múltiplos grupos, sendo que os objetos dentro do mesmo grupo têm uma grande semelhança entre si, mas são suficientemente dissemelhantes dos restantes objetos fora do seu grupo. Os algoritmos tradicionais de agrupamento conseguem encontrar padrões globais nos dados, por exemplo numa matriz, conseguem criar grupos de linhas que são similares entre si, tendo em conta toda a dimensão das colunas. Contudo, existe uma técnica avançada de agrupamento de dados, denominada "Biclustering" que, para além dos padrões globais, consegue ainda encontrar padrões locais, como grupos de linhas que são apenas similares em algumas colunas da matriz. Neste trabalho, explorámos as potencialidades desta técnica de biclustering na área de filtragem colaborativa, considerando uma matriz onde as linhas são utilizadores e as colunas são itens. O objetivo é encontrar subconjuntos de utilizadores com gostos similares considerando subconjuntos de itens.

Na literatura, já existem alguns trabalhos que combinam técnicas de biclustering com filtragem colaborativa. Alguns autores usam o biclustering como uma etapa de pré-processamento dos dados de forma a reduzir a sua dimensionalidade e, posteriormente, utilizam abordagens clássicas de filtragem colaborativa para realizar as tarefas de recomendação. No entanto, existem também abordagens que realizam as tarefas de recomendação, usando apenas biclustering como técnica para prever os valores em falta na matriz de interação.

Usando como referência o estado da arte da literatura nas abordagens de filtragem colaborativa incorporando técnicas de biclustering, desenvolvemos a nossa própria abordagem. Começámos por avaliar as potencialidades de um algoritmo de biclustering, BicPAM, reconhecido como a principal referência dos algoritmos de biclustering baseados em padrões. De seguida, comparámos a sua performance com outros dois algoritmos de biclustering, o xMOTIFs e QUBIC, usados anteriormente em abordagens de filtragem colaborativa, onde concluímos que tanto o BicPAM como o QUBIC conseguiam, eficazmente, encontrar grupos de utilizadores e itens correlacionados numa matriz de interação.

Propomos uma abordagem de filtragem colaborativa que usa biclustering para reduzir a dimensão dos dados da matriz de interação, aumentando a escalabilidade dos algoritmos Item-based. A nossa abordagem, denominada "User-Specific Bicluster-based Collaborative Filtering" (USBCF), usa o QUBIC como algoritmo de biclustering para encontrar grupos de utilizadores com preferências similares em conjuntos de itens. Através dos grupos encontrados, criamos, para cada utilizador, uma submatriz da matriz U-I original. Esta submatriz contém um pequeno conjunto de dados, personalizado para cada utilizador, que consideramos conter informação essencial sobre as preferências do utilizador em questão. Após a criação de uma matriz individual para cada utilizador, um algoritmo de filtragem colaborativa Item-based é treinado nesse conjunto de dados, criando assim um modelo de recomendação exclusivo para cada utilizador do sistema.

Seguindo os bons princípios para avaliação de uma nova abordagem de filtragem colaborativa, testámos a nossa abordagem no famoso conjunto de dados para recomendação de filmes, MovieLens. Comparámos a performance da nossa abordagem com os principais métodos de filtragem colaborativa, utilizando diferentes métricas, incluindo um método considerado o estado da arte na área de filtragem colaborativa com biclustering (BBCF).

Os resultados da nossa avaliação demonstram que conseguimos superar a maior limitação do método BBCF, uma vez que este método conseguia devolver uma previsão apenas para uma pequena fração dos utilizadores e itens. Além disso, a nossa abordagem consegue ainda uma qualidade de previsão de avaliações comparáveis aos melhores algoritmos de filtragem colaborativa.

Concluímos com uma análise crítica à nossa abordagem, identificando atributos que podem ser adaptados para, no futuro, aperfeiçoar a nossa abordagem. O nosso projeto demonstra as potencialidades das técnicas de agrupamento avançadas na área de sistemas de recomendação e acreditamos que produz valor para ajudar a expandir a literatura da área de filtragem colaborativa, tirando proveito de biclustering.

Palavras-chave: Sistemas de Recomendação, Filtragem Colaborativa, Ténicas de agrupamento de dados

Abstract

Collaborative Filtering is one of the most popular and successful approaches for Recommender Systems. However, some challenges limit the effectiveness of Collaborative Filtering approaches when dealing with recommendation data, mainly due to the vast amounts of data and their sparse nature. In order to improve the scalability and performance of Collaborative Filtering approaches, several authors proposed successful approaches combining Collaborative Filtering with clustering techniques. In this work, we study the effectiveness of biclustering, an advanced clustering technique that groups rows and columns simultaneously, in Collaborative Filtering. When applied to the classic U-I interaction matrices, biclustering considers the duality relations between users and items, creating clusters of users who are similar under a particular group of items. We propose USBCF, a novel biclustering-based Collaborative Filtering approach that creates user-specific models to improve the scalability of traditional CF approaches. Using a real-world dataset, we conduct a set of experiments to objectively evaluate the performance of the proposed approach, comparing it against baseline and state-of-the-art Collaborative Filtering methods. Our results show that the proposed approach can successfully suppress the main limitation of the previously proposed state-of-the-art biclustering-based Collaborative Filtering (BBCF) since BBCF can only output predictions for a small subset of the system users and item (lack of coverage). Moreover, USBCF produces rating predictions with quality comparable to the state-of-the-art approaches.

Keywords: Recommender Systems, Collaborative Filtering, Clustering, Biclustering, Coclustering, Time information

Contents

List of Figures **18**

List of Tables **20**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objectives and Contributions . . . 2
1.3 Context . . . 3
1.4 Structure of the document . . . 3

2 Background **5**
2.1 Collaborative Filtering . . . 5
2.1.1 Classic CF setting and tasks . . . 6
2.1.2 Methods and Categorization . . . 8
2.1.3 Challenges . . . 14
2.2 Evaluating CF-based Recommender Systems . . . 15
2.2.1 Evaluation Protocols . . . 15
2.2.2 Benchmark datasets . . . 16
2.2.3 Evaluation Metrics . . . 17
2.3 Temporal Dynamics in Collaborative Filtering . . . 19
2.3.1 Time-aware . . . 20
2.3.2 Time-dependent . . . 21
2.4 Biclustering . . . 22
2.4.1 Clustering and biclustering . . . 22
2.4.2 Bicluster Type . . . 23

3 Related Work **27**
3.1 Biclustering Collaborative Filtering . . . 27
3.1.1 Rating Prediction Approaches . . . 27
3.1.2 Top-*N* Recommendation Approaches . . . 31
3.2 Time Dimension in Neighborhood-based Collaborative Filtering . . . 33
3.2.1 Temporal CF . . . 33

3.2.2 Time-aware CF . . . 35

4 Evaluating the Potentialities of BicPAM 37
4.1 BicPAM - State-of-the-art Pattern-based Biclustering . . . 37
4.2 Generating Synthetic Biclustering Benchmark Data - G-Tric . . . 40
4.3 Biclustering Evaluation Methodology . . . 44
4.3.1 Biclustering Methods . . . 44
4.3.2 Synthetic Data . . . 46
4.3.3 Performance Metrics . . . 46
4.3.4 Results and Discussion . . . 49
4.4 Final Remarks . . . 53

5 USBCF - A User-Specific Bicluster-based Approach for Collaborative Filtering 55
5.1 Motivation and Main Concept . . . 55
5.2 USBCF Overview . . . 56
5.2.1 Discover Biclusters in U-I Matrix . . . 56
5.2.2 Create User-Specific Bicluster-based Matrix . . . 57
5.2.3 Learning User-Specific Recommendation Models . . . 63
5.3 Final Remarks . . . 63

6 USBCF - A Case Study on Movie Recommendation 65
6.1 Evaluation Methodology . . . 65
6.1.1 Experimental Platform and Software . . . 66
6.1.2 Benchmark Dataset . . . 66
6.1.3 Evaluation Metrics . . . 66
6.1.4 Approaches for Comparative Analysis . . . 67
6.2 Results and Discussion . . . 68
6.2.1 Baseline Approaches - Models and Parameterization . . . 68
6.2.2 BBCF . . . 70
6.2.3 USBCF . . . 72
6.3 Final Remarks . . . 77

7 Conclusions and Future work 79

A Biclustering Algorithms Study 83

B USBCF 85
B.1 Cross-Validation Results . . . 85
B.2 User-based split Cross-Validation Results . . . 86

Bibliography **100**

List of Figures

2.1 Collaborative Filtering principle illustration. 6
2.2 Sample U-I numeric rating matrix (on a 5-star scale). 6
2.3 The Collaborative Filtering Process (adapted from [96]). 7
2.4 Overview of Collaborative Filtering categorization. 8
2.5 Singular Value Decomposition of the U-I example rating matrix. 13
2.6 $User \times Item \times Time$ recommendation space. 20
2.8 Clustering and biclustering example. 23
2.9 Examples of different types of biclusters. (a) Constant value, (b) constant-values on rows, (c) constant-values on columns, (d) coherent values (additive model), (e) coherent values (multiplicative model), (f) overall coherent evolution, (g) coherent evolution on the rows, (h) coherent evolution on the columns (adapted from [76]). 24

3.1 Outline of the rating prediction approach (ITCCCF) proposed by Liang and Leng [70]. 29
3.2 Outline of the rating prediction approach (NBCFu) proposed by Kant and Mahara [60]. 30
3.3 Outline of the rating prediction approach proposed (BBCF) by Singh and Mehotra [101]. 31
3.4 Outline of the Top-*N* recommendation approach (NBCF) proposed by Symeonidis et al. [105]. 32
3.5 Concept Drift in CF (adapted from [25]) 34

4.1 Methodology of the BicPAM algorithm. 38
4.2 Example of mining constants biclusters from a matrix. 39
4.3 Distribution of the rating values in the MovieLens-1M and Netflix datasets. 41
4.4 Example of two biclustering solutions that we wish to compare using biclustering similarity measures. 47
4.5 Performance of the biclustering algorithms in each synthetic dataset. . . . 51

5.1 USBCF approach applied on a small U-I matrix. 56
5.2 Example of a bicluster and an active user with distinct rating pattern. . . . 58
5.3 Example of aggregating two biclusters to create a new dataset. 62

6.1 User-based split cross-validation. 67
6.2 Sensitivity of UBCF and IBCF to the size of the neighborhood. 70
6.3 Sensitivity of BBCF to the size of the biclusters' neighborhood. 71
6.4 Effect of the number biclusters in the neighborhood on the average size of the personalized dataset. 72
6.5 Sensitivity of USBCF-CMR to the size of the biclusters' neighborhood. . 75
6.6 Effect of the number biclusters in the neighborhood on the average size of the personalized dataset. 76

A.1 Evolution of the biclusters found according to the number of rows and columns of the input data. 83

List of Tables

2.1 Predictive error metrics. 18

4.1 Framework to ensure exhaustive search from the BicPAM algorithm. . . . 42
4.2 Properties of the constant columns biclustering solutions varying the size of the input data. 43
4.3 Properties of the order-preserving biclustering solutions varying the size of the input data. 43
4.4 Overlapping statistics from the biclustering solutions of the subsets from Netflix dataset. 43
4.5 Characteristics of the generated synthetic datasets. 46
4.6 Similarity scores and efficiency of the QUBIC algorithm. 50
4.7 Similarity scores and efficiency of the xMOTIFs algorithm. 50
4.8 Similarity scores and efficiency of the BicPAM algorithm. 51
4.9 General information of the QUBIC's solutions. 52
4.10 General information of the xMOTIFs' solutions. 52
4.11 General information of the BicPAM's solutions. 53

6.1 Results of the basic models. 69
6.2 Results of the User-based CF and Item-based CF varying the maximum number of neighbors' parameter 69
6.3 Results of the Matrix Factorization models with best performance. 70
6.4 Sensitivity of BBCF to the number of biclusters in the neighborhood. . . . 71
6.5 Results of the BBCF-NoWeight model. 73
6.6 Results of the BBCF-CombinedSimilarity-MSR model. 73
6.7 Results of the USBCF-MSR and the USBCF-CMR models. 74
6.8 Sensitivity of the USBCF-MCR to the number of biclusters in the neighborhood 75
6.9 Results of the Adaptive Bicluster Neighborhood Dimension version of the USBCF-CMR. 76

A.1 Properties of the constant columns biclustering solutions varying the size of the input data 84

A.2 Properties of the order-preserving biclustering solutions varying the size of the input data 84

B.1 Results of the User-based CF and Item-based CF on the cross-validation varying the maximum number of neighbors' parameter. 85
B.2 Results of the Matrix Factorization models on the cross-validation varying the number of features. 86
B.3 Sensitivity of the BBCF to the number of biclusters in the neighborhood. . 86
B.4 Results of the basic models on the user-based cross-validation. 86
B.5 Results of the User-based CF and Item-based CF on the user-based split cross-validation varying the maximum number of neighbours' parameter. 87
B.6 Results of the Matrix Factorization models on the user-based cross-validation varying the number of features. 88
B.7 Results of the BBCF on the user-based cross-validation varying the number of nearest biclusters' parameter. 88
B.8 Results of the BBCF-NoWeight model on the user-based cross-validation. 88

Chapter 1

Introduction

1.1 Motivation

Recommender systems (RS) are software tools and techniques that support users in the decision-making process by suggesting products, services, or other types of items from a collection [93]. The decision-making process can be complicated, especially for people who lack sufficient personal experience to evaluate the potentially overwhelming number of options. Thus, recommender systems can be found in many industries and applications and are responsible for a big piece of the revenues of many companies. Companies like Spotify, Netflix, and Amazon depend on their recommendations to allow users to discover new content and improve the user experience in their services.

This work focuses on *Collaborative Filtering* (CF), one of the most popular and successful classes of the methods used by recommender systems, which recommend items to users based on the preferences other users have expressed for those items. The information domain in CF usually consists of triplets *<user, item, rating>*, representing how the user rated an item. The set of triplets forms a sparse matrix referred to as User-Item rating interaction matrix (U-I matrix), containing all the users and items in the system, together with the respective ratings. The sparse nature of this matrix results from the unknown ratings, in cases the user has not (yet) rated the item.

Collaborative Filtering methods are usually divided into two main classes, the *memory-based* and the *model-based*. The memory-based algorithms use the user-item system ratings directly to predict ratings for new items. This can be done by *User-based* or *Item-based* recommendation. User-based systems predict the preference of a particular user for an item, using the ratings that similar users, classed neighbors, gave to that item. On the other hand, item-based approaches predict the rating based on how the active user rated similar items.

Some challenges limit memory-based CF effectiveness when dealing with recommendation data, mainly due to the vast amounts of data and their sparse nature. Model-based approaches try to overcome the limitations of memory-based CF by training models using

the available data and later use them to predict users' ratings for new items. Dimensionality reduction, such as Matrix Factorization and *Clustering* are examples of techniques adopted by model-based approaches.

Cluster analysis or simply Clustering is the process of partitioning a set of data objects into groups. Each group is a cluster, such that objects in a cluster are similar to one another yet dissimilar to objects in other clusters. Clustering is fundamental in many applications, such as business intelligence, image pattern recognition, Web search, biology, and security [42]. However, traditional clustering techniques can only be applied to either the row dimension or the column dimension of the data matrix separately. Moreover, when computing the clusters of objects, these algorithms use the entire dimension (all the columns or all the rows). In real-world scenarios, the correlation of a subset of rows is frequently only significant and meaningful for a subset of the overall columns, and vice versa [100]. *Biclustering* is an advanced clustering technique that performs Clustering in two dimensions simultaneously, being able to find these local pattern patterns of correlated sub-spaces in the data (biclusters). Many biclustering approaches have been proposed, particularly in the context of gene expression data analysis [16, 41, 80, 107]. Nonetheless, although the massive impact biclustering is having in biological applications, it is also showing promising results in many other domains, including in Collaborative Filtering [34, 60, 101, 105, 123].

In this context, we propose a novel biclustering-based collaborative filtering approach that uses biclustering to improve the scalability of memory-based CF methods. The approach, named "User-specific Bicluster-based Collaborative Filtering" (USBCF), uses biclustering to find groups of users with similar preferences under a particular group of items (biclusters). Then, it uses these biclusters to create user-specific small and denser U-I matrices used to train traditional CF models.

We evaluated the proposed approach against baseline CF methods as well as the state-of-the-art approach of biclustering-based Collaborative Filtering (BBCF) [101]. The evaluation results that the proposed approach can successfully suppress the main limitation of the previously proposed state-of-the-art biclustering-based Collaborative Filtering (BBCF), since BBCF can only output predictions for a small subset of the system users and item (small coverage). Moreover, USBCF produces rating predictions with quality comparable to the state-of-the-art CF approaches.

1.2 Objectives and Contributions

The main goal of this thesis is to investigate and highlight the potentialities of biclustering when applied to the Collaborative Filtering domain, while overcoming limitations presented by the state-of-the-art of the biclustering-based CF approaches.

Bellow, we list the central contributions of this work:

- Presents a comprehensive survey of the approaches developed in the field of biclustering-based CF. It also reviews significant research contributions regarding temporal dynamics in neighborhood-based CF that can easily be included in the biclustering-based CF methods.

- Evaluates the potentialities of three biclustering algorithms to discover hidden biclusters in synthetic data resembling U-I matrices.

- Proposes a new biclustering-based CF methodology, USBCF, that overcomes the coverage capability limitation presented by previously proposed related-work.

- Revises the popular bicluster quality measure, Mean-Square-Residue (MSR), so that the missing-values on the bicluster do not contribute to the residue. Moreover, it proposes a new quality measure, Mean-Column-Residue, to measure the quality of biclusters with constant-values on biclusters.

- Exhaustively evaluates the proposed approach on a movie-recommendation real-world benchmark dataset, comparing it against state-of-the-art CF methods.

- Discusses the challenges of biclustering when applied to recommendation scenarios and presents potentially relevant research avenues for future work in the field.

1.3 Context

This work was carried out at the research line of excellence of Data and Systems Intelligence at the Laboratory of Large Scale Systems (LASIGE). This work could not have been done without the guidance of Prof. Sara Madeira.

Fundação para a Ciência e Tecnologia (FCT) partially funded this work through projects Neuroclinomics2 (PTDC/EEI-SII/1937/2014) and iCare4U (PTDC/EME-SIS/31474/2017), and plurianual funding to LASIGE (UIDB/00408/2020).

1.4 Structure of the document

This document is organized as follows:

- Chapter 2 introduces the Collaborative Filtering paradigm, focusing on the main tasks, traditional approaches, and evaluation methodologies. This chapter also presents the biclustering task and defines its main concepts.

- Chapter 3 reviews the state-of-the-art concerning biclustering-based CF approaches and time-exploiter neighborhood-based CF algorithms.

- Chapter 4 studies the potentialities of three biclustering algorithms to discover biclusters in an U-I interaction matrix, with special focus on BicPAM, the state-of-the-art pattern-based biclustering algorithm.
- Chapter 5 proposes a new CF approach that uses biclustering to create user-specific U-I matrices, improving scalability of traditional CF algorithms.
- Chapter 6 validates and evaluates the proposed approach using a movie recommendation real-world dataset, comparing the approach against baseline and state-of-the-art CF approaches.
- Chapter 7 concludes the work, drawing conclusions and highlighting major directions for future work.

Chapter 2

Background

This chapter introduces the theoretical background of this work and is organized as follows. Section 2.1 provides an overview of the Collaborative Filtering paradigm. Section 2.2 introduces the key principles to address the evaluation of Collaborative Filtering approaches. Section 2.3 explores the benefits of exploiting time information in CF. Finally, Section 2.4 covers the biclustering technique, defining its main concepts.

2.1 Collaborative Filtering

Decision making plays a vital role in everyone's lives. Every single day we are confronted with options and choices for which our decisions determine the outcome of our lives. What to wear? What to eat? What to buy? What show should I watch? Effective decision-making can be difficult, especially in domains where the pool of options is massive. There has been both academic and industry interest in how to automatically recommend items to individuals. Spotify, Amazon, Netflix, and Facebook are some popular platforms who actively use recommender systems [28]. From e-commerce to online advertisement, these systems are unavoidable in our daily online journeys to suggest items in a personalized way.

A wide variety of methods have been proposed and adopted [93], but this thesis focuses on the class of Collaborative Filtering methods. Collaborative Filtering (CF), firstly proposed in 1992 [36], is currently the most familiar, widely implemented, and mature of the technologies among the traditional ones (e.g., content-based and hybrid-based techniques) used to build recommender systems. It is based on the assumption that users who had similar likings in the past will have similar likings in the future. Meaning that if a person P1 and a person P2 both like an item I1, then it assumes it is more likely P1 to have a similar opinion to P2 on an item I2 than having a similar opinion to a person P3 who does not share opinions with P1. Figure 2.1 illustrates the principle of Collaborative Filtering.

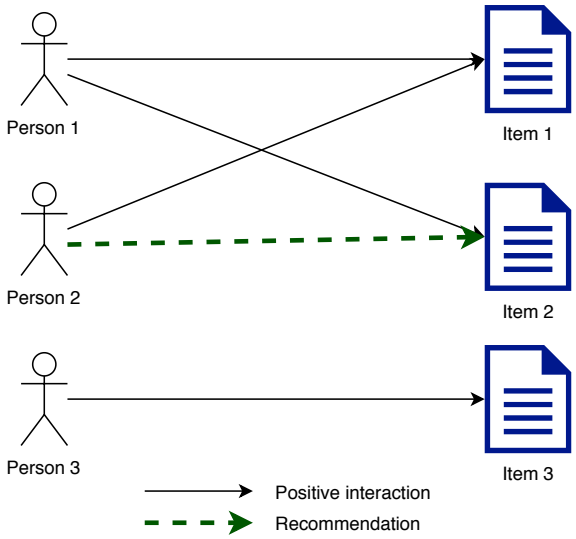


Figure 2.1: Collaborative Filtering principle illustration.

2.1.1 Classic CF setting and tasks

In a classic Collaborative Filtering scenario, we have a set of m users $U = \{u_1, ..., u_m\}$ and a set of n items $I = \{i_1, ..., i_n\}$ which compose a User-Item matrix $\boldsymbol{R}_{m \times n}$ (U-I matrix). Each entry $\boldsymbol{R}_{ij}$ indicates the inclination of the user i towards the item j, usually in the form of rating, so the matrix $\boldsymbol{R}$ expresses the preference of each user for the n items if $\boldsymbol{R}_{ij} > 0$. Figure 2.2 shows an example of a U-I matrix for five users and four items. The blank cells indicate unknown values, where the user has yet not rated the item. Equivalently, since the rating matrix is usually very sparse, it is frequently expressed as a set of <*user, item, rating*> triples, instead of a full matrix representation.

Users \ Items	i_1	i_2	i_3	i_4
u_1	4		2	2
u_2	5	5		2
u_3	4	5	2	
u_4			5	5
u_5	3	3	5	

Figure 2.2: Sample U-I numeric rating matrix (on a 5-star scale).

Depending on the systems in question, the ratings can take many forms, differing in connotation. Some ratings are obtained by asking feedback from the users explicitly, while others by observing their interactions implicitly with the system. According to the type of user feedback, the rating data is usually classified as one of two types [110]:

- **Numeric rating feedback.** Usually in the form of triples <*u, i, r*>, which expresses the event of the rating r given by the user u to the item i. This type of data is usually gathered from explicit feedback of the user. Some systems use real-valued rating such as 0-5 stars scale, while others use binary (like/dislike) scales [28]. The U-I matrix previously presented in Figure 2.2 is an example of a numeric rating feedback matrix.

- **Positive-only/unary feedback.** For the most part, a set of *(u,i)* tuples, represents a positive interaction between a user u with an item i. This type of data is usually obtained from implicit interactions of the user, such as users' views/clicks in videos of a video-sharing platform.

Under this setting, the main problem of CF can be defined as: Given a U-I matrix $\boldsymbol{R}$ that represents a known set of preferences of M users to N items, perform a recommendation task to a user u, called the *active user*. Recommendation systems, including Collaborative Filtering, are primarily used for **prediction** or **recommendation tasks** [79, 116]. Figure 2.3 shows a schematic diagram of the CF process.

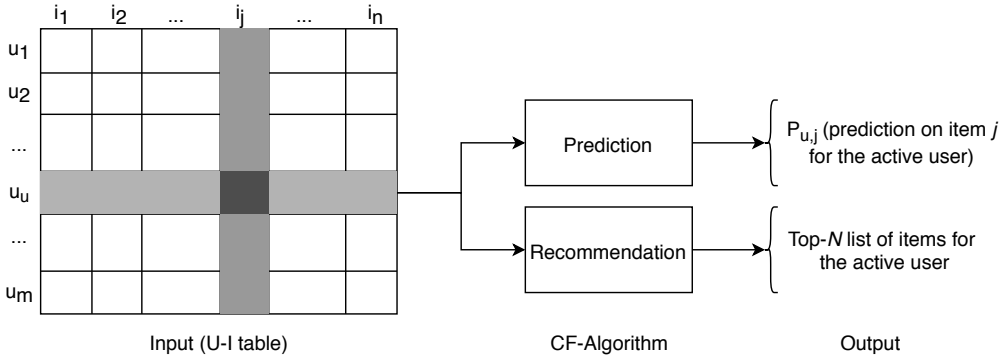


Figure 2.3: The Collaborative Filtering Process (adapted from [96]).

The goal of the *prediction* task, otherwise called "Annotation in Context" [51], is to predict the rating a user would give to a *target item*. In other words, anticipate how much the user might like a particular item. These systems use known values of the rating matrix and try to predict the missing ones.

As for the *recommendation* tasks, the system usually generates and provides an ordered list of *n* items, known as Top-*N* recommendation list, which is a list including the most relevant/useful items for the user. This classic recommendation task is also referred to as "Find Good Items" [51]. There are many subcategories of tasks within the "Find Good Items" class because, in some contexts, we might not want the n items with the highest predicted preferences, as preference may not be the only criteria relevant to produce the recommendation list [28]. This is particularly true in systems that want users to explore items that differ from their usual choices. However, it is a common practice to use

predictive Collaborative Filtering to perform recommendation by generating predictions and returning the items with the highest ratings.

2.1.2 Methods and Categorization

In the literature, the authors usually group CF approaches as one of the three major categories: *memory*, *model*, and *hybrid-based* [102], as illustrated in Figure 2.4. This Section gives a brief explanation about each one of the categories, discussing their pros and cons and indicating some popular baseline and state-of-the-art algorithms.

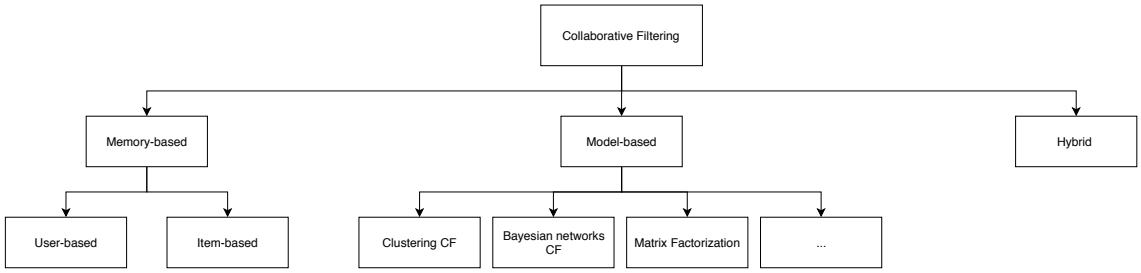


Figure 2.4: Overview of Collaborative Filtering categorization.

Memory-based CF

Memory-based CF algorithms use the entire U-I matrix, or a sample of it, to generate predictions for new items [102]. *User-based* and *item-based* are the two known techniques that follow this principle effectively.

A traditional *user-based* algorithm, such as GroupLens [92], predicts the rating $r_{u,i}$ of a user u for an item i using the ratings given to i by the k users most similar to u, called the k-nearest neighbours. The *user-based* approaches use similarity functions to estimate how similar the two users u and v are. Popular examples of similarity functions are the cosine vector similarity:

$$sim(u,v)_{cos} = \frac{\vec{r}_u \cdot \vec{r}_v}{\|\vec{r}_u\| \, \|\vec{r}_v\|} = \frac{\sum_{i \in I} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I} r_{u,i}^2} \sqrt{\sum_{i \in I} r_{v,i}^2}}, \tag{2.1}$$

and the Pearson correlation as:

$$sim(u,v)_{corr} = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}. \tag{2.2}$$

Then, the ratings given by the neighbors to the item are aggregated and used to estimate a rating prediction [116]. This aggregation is typically done by computing the weighted average of the neighbouring users' ratings on item i, using similarity as the weights:

$$\hat{r}_{u,i} = \overline{r}_u + \frac{\sum_{u' \in N} sim(u,u')(r_{u',i} - \overline{r}_{u'})}{\sum_{u' \in N} |sim(u,u')|}, \tag{2.3}$$

where $\overline{r}_u$ corresponds to the active user's mean rating, and $sim(u, u')$ is the similarity between the active user and a neighbour. This is an enhancement of the original weighted average, since the ratings are mean-centred by the users' ratings. This idea compensates for differences in users' use of the rating scale because some users tend to give higher ratings than others [28]. There are some variations of this formula and other types of aggregating functions, but this is the most common as it is known for producing consistent results.

Example 2.1.1. Consider the ratings from Figure 2.2 as an example. We want to predict the rating that the user u_1 would give to the item i_2, using *user-based* approach with the following configuration:

- Similarity measure: Pearson correlation (Equation 2.2).
- Neighbourhood size: 2 users.
- Aggregation function: Weighted average with mean offset. (Equation 2.3).

Users u_2, u_3, and u_5 were the ones who rated the item i_2, so if we calculate the similarities of each one of them with u_1, we obtain: $sim(u_1, u_2) = 0.800$, $sim(u_1, u_3) = 0.614$, and $sim(u_1, u_5) = -0.800$. Then, u_2 and u_3 are selected since they are the two most similar users to u_1 (neighborhood size = 2), and the rating is predicted as:

$$\begin{aligned}\hat{r}_{u_1,i_2} &= \bar{r}_{u_1} + \frac{sim(u_1,u_2)\,(r_{u_2,i_2} - \bar{r}_{u_2}) + sim(u_1,u_3)\,(r_{u_3,i_2} - \bar{r}_{u_3})}{|sim(u_1,u_2)| + |sim(u_1,u_3)|} \\ &= 2.667 + \frac{0.800 \times (5-4) + 0.614 \times (5-3.667)}{0.800 + 0.614} \\ &= 3.811.\end{aligned}$$

User-based CF, despite usually producing good results, it suffers from scalability problems when the number of users grows. This scenario is quite common in real-world applications, and for that reason, Amazon introduced the *item-based* CF in 1998, but the first work describing the approach was only published in 2001 by Sarwar et al. [96]. The approach introduced by Sarwar et al. popularized the idea of pre-computing and storing all the similarities before the prediction/recommendation step. In most user-based systems, this idea is not possible because users are frequently entering and exiting the system, rating and re-rating items, which results in unstable similarities between users. For this reason, user-based algorithms usually search for neighbours only during prediction/recommendation phase. On the other hand, this volatility on the users' ratings affects the similarity between two items significantly less than the similarity between two users in systems with a sufficiently high user to item ratio [28]. This is the typical case of real-world systems, therefore, it is reasonable to pre-compute all the similarities between all the items before the prediction/recommendation step. Moreover, with an item-based method, the system can easily justify a recommendation to the user, explaining that it is recommending an

item because he/she in the past positively interacted with similar ones. Whereas in user-based methods, the active user usually does not know that other users are being used as neighbours [93].

The *item-based* algorithms' main concept is identical to the one we have in the user-based CF. However, the recommendation of an item depends only on the information about other items previously rated by the active user. The item-based approach looks into a set of items the active user has rated, computes the similarities between those items and the target item i, and then selects the k most similar items. Similarly to user-based CF, there are many possible functions to compute the similarities. Given two items i and j, common methods include:

- **Pearson correlation.**

$$sim(i,j)_{corr} = \frac{\sum_{u \in U} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{uj} - \bar{r}_j)^2}}. \tag{2.4}$$

  Computes statistical correlation between two item's common ratings. It is more frequently used in user-based CF, as it usually performs worse than cosine similarity in item-based CF [96].

- **Cosine similarity.**

$$sim(i,j)_{cos} = \frac{\vec{r}_i \cdot \vec{r}_j}{\|\vec{r}_i\| \, \|\vec{r}_j\|} = \frac{\sum_{u \in U} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U} r_{u,i}^2} \sqrt{\sum_{u \in U} r_{u,j}^2}}. \tag{2.5}$$

  Cosine distance between two items' ratings vectors. It is very efficient and produces good accuracy results.

- **Adjusted-cosine similarity.**

$$sim(i,j)_{adjcos} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_u)^2}}. \tag{2.6}$$

  Subtracting the user's mean rating makes the function properly deal with users that use different rating values to quantify the same level of appreciation for an item. This is particularly important, as different users have their own underlying scale to express their preferences.

- **Item-mean-centered cosine similarity.**

$$sim(i,j)_{itemcenteredcos} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}. \tag{2.7}$$

Despite adjusted-cosine solving the obvious problem of different rating scales between users, subtracting the item's mean rating instead of the user's mean rating seems to be more effective [27].

Once found the k most similar items, the predicted rated is computed by aggregating the ratings the active user gave to those items, typically weighting averaging as [96]:

$$\hat{r}_{u,i} = \frac{\sum_{j \in N} sim(i,j) r_{u,j}}{\sum_{j \in N} |sim(i,j)|}, \tag{2.8}$$

where N is the set of the k most similar items to j that the user u had previously rated.

Example 2.1.2. Consider again the ratings from Figure 2.2 as an example. We want to predict the rating that the user u_1 would give to the item i_2, using the *item-based* approach with the following configuration:

- Similarity measure: Cosine similarity (Equation 2.5).
- Neighborhood size: 2 items.
- Aggregation function: Weighted average (Equation 2.8).

If we calculate the similarities of each one of items with i_2, we obtain: $sim(i_2, i_1) = 0.865$, $sim(i_2, i_3) = 0.427$, and $sim(i_2, i_4) = 0.227$. Then, i_1 and i_3 are selected since they are the two most similar items to i_2 (neighborhood size = 2), and the rating is predicted as:

$$\begin{aligned} \hat{r}_{u_1,i_2} &= \frac{sim(i_2,i_1) r_{u_1,i_1} + sim(i_2,i_3) r_{u_1,i_3}}{|sim(i_2,i_1)| + |sim(i_2,i_3)|} \\ &= \frac{0.865 \times 4 + 0.427 \times 2}{0.865 + 0.427} \\ &= 3.339. \end{aligned}$$

Two drawbacks are typical for memory-based CF approaches. First, the algorithms usually do not scale due to the computation of similarities between all pairs of users or items (quadratic time complexity). Secondly, the choice of the similarity function is crucial because it affects the results of the model [116]. In order to compensate for this dependency on the similarity, many modifications have been proposed to improve the traditional algorithm. For instance, some models incorporate extra features such as information of the users and items order to improve the recommendations [116]. There is a panoply of works exploring this idea, but using social networks as side information (i.e., friendship graph, followers, and trusted users) has been successful in many works because people tend to share opinions with individuals who are related to them [35, 77].

Model-based CF

Model-based approaches, in contrast to neighbourhood-based systems, use the rating data to learn a predictive model. The trained models are then able to recognize the patterns in the data and perform the Collaborative Filtering tasks. There is a vast amount of works proposing and exploring different model-based techniques, such as Bayesian networks [14], clustering/biclustering (see Section 3.1), latent semantic [52, 106], mixture models [62], and transfer learning [68].

State-of-the-art model-based approaches address, for the most part, scalability and sparsity problems better than memory-based approaches and, also, can produce better recommendations. However, there is a trade-off between performance and scalability since the training phase of complex models is usually more costly. Many of these approaches use dimensionality reduction techniques, but this can come with the cost of potentially losing useful information, leading to less accurate recommendations.

Latent semantic analysis (LSA) is a natural language processing technique that some CF models use to find latent/hidden features in the rating data [52]. In these models, *Singular Value Decomposition* (SVD) is usually the key dimensionality reduction tool responsible for uncovering the latent factors in the data. SVD-based models also referred to as matrix factorization models, became a top option for implementing CF systems, after becoming popular for playing a significant role in the Netflix Prize competition [11].

For an $\boldsymbol{R}_{m\times n}$ input matrix, SVD factorizes $\boldsymbol{R}$, using r concepts/topics, into an approximation of a product of three matrices with as:

$$\boldsymbol{R}_{m\times n} \approx \boldsymbol{U}_{m\times r}\boldsymbol{\Sigma}_{r\times r}\left(\boldsymbol{V}_{n\times r}\right)^{\mathsf{T}}, \tag{2.9}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices that store the left and right singular vectors, respectively, and $\boldsymbol{\Sigma}$ is a diagonal matrix whose values are the singular values of the decomposition. It is possible to reduce the $r \times r$ matrix $\boldsymbol{\Sigma}$ to have only the k largest diagonal values, obtaining a matrix $\boldsymbol{\Sigma}_k$. Given that, if the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are reduced accordingly, then the reconstructed matrix $\boldsymbol{R}_k = \boldsymbol{U}_k.\boldsymbol{\Sigma}_k.\boldsymbol{V}_k^{\mathsf{T}}$ is the closest rank-k matrix to $\boldsymbol{R}$.

When $\boldsymbol{R}$ is a rating matrix, the matrix factorization models map both users and items to a joint latent factor space of dimensionality r, where r is the number of factors/concepts that try to explain those ratings. If the items of the matrix are movies, the latent factors could measure, for instance, the amount of action, amount of comedy, character development, orientation to adults, or completely uninterruptible factors [93]. In this case, we could interpret the SVD as relationships between "movies", "users", and "concepts/-factors", where $\boldsymbol{U}$ is the user-to-concept similarity matrix, $\boldsymbol{V}$ is the movie-to-concept similarity matrix, and $\boldsymbol{\Sigma}$ stores the "strength" of each concept. Figure 2.5 shows the use of SVD in the U-I rating matrix from Figure 2.2.

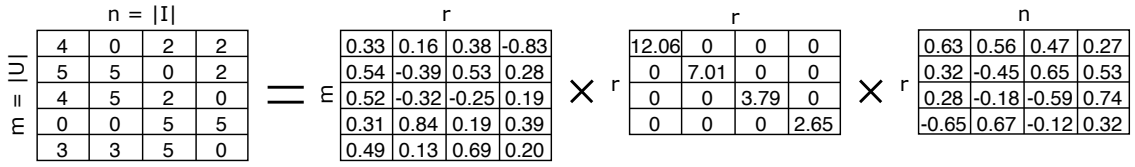


Figure 2.5: Singular Value Decomposition of the U-I example rating matrix.

The purest SVD implementation is undefined when the input matrix has missing values. In the SVD example of Figure 2.5, we considered the missing values as 0, which is unreasonable for recommendation scenarios since SVD treats those values as vulgar rating scores. Sarwar et al. [95] studied the imputation of the item's average ratings and found it to work better than the imputation of the user's average. However, imputation significantly increases the amount of data, leading to a computationally expensive model with possibly distorted predictions due to inaccurate imputations. In order to overcome the drawbacks of the imputation, several approaches have been proposed [10, 32, 63, 121]. Those approaches compute an estimation of the SVD but only using the known ratings. Alternating Least-Squares method (ALS) [121] and gradient descent-based [32] are two popular approaches. The gradient descent method for estimating SVD usually has a regularization factor in order to prevent the model from overfitting. The regularized SVD tends to predict ratings more accurately than the unregularized SVD [28]. Moreover, this method can be updated iteratively as new users, items and ratings enter the system. The baseline rule to predict the preference of a user *u* for and item *i* is:

$$\hat{r}_{u,i} = \sum_{f=1}^{F} u_f \times o_f \times i_f. \tag{2.10}$$

However, as in the Memory-based CF, it is common practice to normalize the rating data before computing the model by subtracting a user and item-bias, $b_{u,i}$, in order to optimize the recommender's accuracy [28]. In this case, the normalization bias has to be taken into account in the prediction rule as:

$$\hat{r}_{u,i} = b_{u,i} + \sum_{f=1}^{F} u_f \times o_f \times i_f. \tag{2.11}$$

Hybrid-based CF

There are recommender systems that combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one. Pure hybrid Collaborative Filtering systems combine algorithms of both memory and model-based classes in order to overcome the limitations of the individual approaches. The recommendation performance of hybrid systems is generally better than most pure memory-based and model-based CF algorithms because they combine the strengths of the component

algorithms [15]. Some systems combine not only Collaborative Filtering algorithms but also techniques of other types of recommendation systems, such as content-based.

Burke [15] surveyed hybrid recommender systems and grouped them into seven classes:

- **Weighted.** Combining "votes" of different recommenders into a single output.
- **Switching.** Switching between algorithms depending on the current context.
- **Mixed.** Returning the results of the different recommenders.
- **Feature-combining.** Using various inputs from different data sources.
- **Cascade.** Chain of recommenders, where one recommender uses the output of the other as input, in order to refine it.
- **Meta-Level.** A model trained by one algorithm is used as input by another algorithm.

Each category has pros and cons, and so it is not possible to declare that a specific category is the best. The choice of the technique and algorithm for the recommender system should always take into account the environment and context in which we want to apply it.

2.1.3 Challenges

The primary purpose of a recommender system is to provide fast and accurate predictions to the user, contributing to better user experience and bringing value for the company. However, recommendation algorithms are usually applied in challenging environments, impacting the scalability and accuracy of the systems [79, 102]. The value of a recommender system is directly linked to how well it addresses these challenges:

- **Data Sparsity.** Recommender systems are usually used in contexts with a large number of items. When the number of the users is low compared to the number of items, this leads to a sparse U-I matrix, profoundly affecting the performance of the algorithms. An example of a data sparsity problem is the cold-start issue that happens when a user just entered the system. The user's preferences are not yet known, so it is not easy to make a trustworthy recommendation.
- **Scalability.** Scalability problems may occur when the number of users and items grow. Ideally, the system should maintain the performance even when the size of the context (U-I matrix) grows tremendously.
- **Synonymy.** Due to the natural properties of the natural language, sometimes different names can address the same item. For example, "horror movie" and "horror

film" both have the same meaning. However, most recommender systems are not able to detect this association between items, and so they are treated as different items, affecting the performance.

- **Grey Sheep Users.** Users whose behaviour is unpredictable. These users do not consistently agree with any group of people, which goes against the main principle of Collaborative Filtering. An example of a grey sheep is a user who loves the first Harry Potter movie but hates the second movie of the saga.

- **Shilling Attacks.** Some users deliberately try to trick the system. When there are no restrictions on who can make feedback, some users may want to boost or depress an item in the recommendation list. An example of a shilling attack is workers of a company giving excellent feedback to their product and poor feedback to products of rival companies.

- **Explainability.** The ability to justify the user why the recommendation system is recommending him a specific item is valuable.

2.2 Evaluating CF-based Recommender Systems

As previously mentioned, different types of recommender systems are used for different tasks and contexts, which makes it challenging to introduce a consensual evaluation framework. Since an algorithm that is considered adequate for a particular context may be completely inappropriate in another context, to properly evaluate a recommender system, it is essential to take into consideration the tasks for which it is going to be used. In this section, we present an overview of the critical aspects to consider in order to evaluate a recommender system properly.

2.2.1 Evaluation Protocols

In the recommender systems' literature, **offline experiments** play a significant role in measuring the predictive and classification accuracy of the systems because it is quick and economical [28]. The models are evaluated using data splitting techniques such as the train-test-split or cross-validation. There are, however, two significant drawbacks of offline analysis. The sparsity nature of the data limits the set of ratings that can be evaluated since it is not possible to test the prediction without the "real" rating values. Secondly, this type of evaluation does not take into consideration environmental variables that may affect the decision of the user, for example, the aesthetics of the user interface might affect how the user interacts with the system [51]. Thus, we can not guarantee that an algorithm with excellent offline performance will have a similar performance in a real-world scenario.

Besides the flaws previously mentioned, when the data is partitioned into the train and test sets, the data is usually shuffled, which means that we could potentially be using future ratings to predict older ones. Because of this, some works take advantage of the rating timestamps in order to evaluate the algorithms more faithfully. One solution to deal with this problem is treating the dataset as a data stream, which means trying to "replay" a series of ratings and recommendations. Each time a rating enters the system, the model can make the recommendation task based only on the data before that "moment". After evaluating the result, the actual rating is inserted, and the next moment can be evaluated [51]. Having this into consideration, Vinagre et al. [109] proposed a framework able to monitor the accuracy evolution of an algorithm, allowing the detection of phenomena that the traditional offline evaluations do not consider.

An alternative in order to tackle the downsides of offline evaluations is to conduct **live/online experiments**, such as field trials and virtual lab studies [28]. Online evaluation techniques allow us to evaluate and compare algorithms by simulating user behaviour. Field trials, such as A/B testing, consist in observing, in sites with already established user activity, a new feature/algorithm on a subset of the users. For testing in non-deployed applications, virtual labs can be used, but there is always an artificial interaction with the users that may affect the results of the study. Although live experiments are more informative about the real interaction of the user and the performance of the system, in many cases, they require creating online testing systems that are costly. Moreover, the reproducibility of the results is not possible [110].

2.2.2 Benchmark datasets

Regarding datasets for CF recommender systems evaluation, some are considered benchmarks. The majority of studies related to Collaborative Filtering recommender algorithms have used one of the datasets described below. A few other datasets have been used [110], but most of them are not publicly available.

- **Movielens.**[1] GroupLens Research has made available rating datasets from the MovieLens, a movie recommendation service [43]. The Movielens-20M dataset is the one recommended for research purposes. It contains 20,000,263 ratings from 138,493 users for 27,278 movies. The ratings were registered from January 09, 1995, to March 31, 2015, and the records have timestamp information. All the users in this dataset had rated at least 20 movies using a 0-5 stars scale with a 1/2 start granularity.

- **Jester.**[2] This dataset is a collection of 4.1 million continuous ratings (-10.00 to +10.00) of 100 jokes from 73,421. The data was collected between April 1999 to

[1] `https://grouplens.org/datasets/movielens/20m/`
[2] `https://goldberg.berkeley.edu/jester-data/`

May 2003 from the Jester joke recommender system [37]. This dataset does not contain any timestamp information.

- **BookCrossing.**[3] It contains 1,149,780 book ratings crawled from the book sharing and discussion platform bookcrossing.com [122]. The ratings of this dataset are a mix of explicit real-valued (1-10) and positive-only ratings from 278,858 users to 271,379 different books. This dataset does not contain any timestamp logs.

- **Netflix.**[4] In October 2006, Netflix released a movie rating dataset and challenged, for a $1M Grand Prize, the research community to develop systems that could beat the accuracy of Cinematch [11]. The released dataset contains ratings of 480,000 users to 17,770 movies, a total of over 100 million movie ratings on a scale from 1 to 5 stars. The dataset also contains information about the registering of each rating, in the form of YYYY-MM-DD date, ranging from 1998-11-01 to 2005-12-31.

- **Lastfm.**[5] Last.fm is a music website [6], which provides access to their data through their web services. Last.fm Dataset - 1K users is a dataset collected from Last.fm API for research purposes, and represents the whole listening habits (until May, 5th 2009) for 992 users. It consists of 19,150,868 tuples in the form of user-artist-song-timestamp. Unlike the other benchmarks datasets presented, this one does not contain rating information so it can not be used for rating prediction evaluation, but it is viable for Top-*N*-recommendation evaluation.

2.2.3 Evaluation Metrics

A wide variety of metrics to evaluate Collaborative Filtering algorithms can be found in the literature because different systems with different purposes require different metrics to evaluate them. Herlocker et al. [51] divided accuracy metrics into three classes: *predictive* accuracy, *classification* accuracy, and *ranking* accuracy metrics. Apart from those, there are also some metrics that do not measure the accuracy of the system but rather the usefulness for the user, however those are not addressed in this work [51].

Predictive metrics measure the closeness of the predicted ratings to the actual user ratings, and so are mainly relevant when the task of the system is rating prediction. Mean absolute error (MAE) is the most popular predictive metric, and it measures the average of the absolute deviation between the predicted and the real rating. There are several metrics related to MAE, such as the mean square error (MSE), the root mean squared error (RSME), and the normalized mean absolute error (MAPE) [51]. Table 2.1 shows the formulas to calculate each of the mentioned predictive errors.

[3] `http://www2.informatik.uni-freiburg.de/~cziegler/BX/`
[4] `https://archive.org/download/nf_prize_dataset.tar`
[5] `http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html`
[6] `http://last.fm`

Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert e_i\rvert$
Mean squared error	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} e_i^2$
Root mean square error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$
Mean absolute percentage error	$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left\lvert\frac{e_i}{y_i}\right\rvert$

Table 2.1: Predictive error metrics.

It is a widespread practice to evaluate an algorithm with both MAE and RMSE at the same time since the RMSE penalizes large errors more than the MAE, which gives more extra information about the performance of the system.

Classification metrics quantify the frequency with which a system makes the correct or erroneous classification of whether an item should be recommended. So these metrics are appropriate when the goal is to recommend a list of n items to the user. The precision-recall framework of the information retrievals is widely used for this type of task [102, 116]. Precision represents the probability that a selected item is relevant [51] as:

$$precision = \frac{t_p}{t_p + f_p}, \tag{2.12}$$

where t_p is the number of items retrieved that were relevant, and f_p is the number of items retrieved which were not relevant for the user. Recall, on the other hand, represents the probability that a relevant item will be selected:

$$recall = \frac{t_p}{t_p + f_n}, \tag{2.13}$$

where f_n is the number of items that were not retrieved but were considered relevant. Some works, in addition to the recall and precision metric, also use the F_1 score, which combines both precision and recall into a single score:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}. \tag{2.14}$$

An alternative to these metrics is the usage of the *ROC model*, which measures the capability of the system to distinguish between relevance and noise [51].

Ranking metrics measure how well the recommendation algorithm produces an ordered list of items. There are two popular ways to measure the accuracy of the ranking.

The first approach uses a reference ranking (a correct order) and evaluates how similar the ordering of the list produced is when compared to the reference one. Ranking correlation measures such as Spearman's p, and Kendall's Tau are are popular choices [51, 93].

The other popular approach is to assume that the utility of a list of recommendations is additive, given by the sum of the utilities of the individual recommendations. The utility of each recommendation is the utility of the recommended item discounted by a factor that depends on its position in the ranked list. The half-life utility is an example of a utility-based ranking metric [14, 45]. The half-life metric is an interesting metric that measures the utility of the returned ranked list by assuming that the likelihood that a user will view an item in the list decays exponentially for every successive item. This is done with the usage of a decay weight function that give most of the weight to the items from the top of the list, and successively less to the ones bellow. Therefore, this metric is incredibly convenient in domains where this assumption does verify, such as in e-commerce recommender systems [28]. Another option, for applications in which the user is expected to read a relatively large portion of the list, is the Normalized Cumulative Discounted Gain (nDCG) [57], a measure from the information retrieval field, where positions are discounted logarithmically. Assuming each user u has a "gain" g_{ui_j} from the recommendation of the item i. The average Discounted Cumulative Gain (DCG) for a list of items L is defined as:

$$\text{DCG} = \frac{1}{N}\sum_{u=1}^{N}\sum_{j=1}^{L}\frac{g_{ui_j}}{\max\left(1, \log_b j\right)}, \tag{2.15}$$

where the logarithm base is parameter, but a logarithm with base 2 is commonly used to ensure all positions are discounted. nDCG is the normalized version of DCG given by:

$$\text{nDCG} = \frac{\text{DCG}}{\text{DCG}^*}, \tag{2.16}$$

where DCG^* is the ideal discounted cumulative gain.

2.3 Temporal Dynamics in Collaborative Filtering

Besides the typical U-I rating data, it is possible to take into account the time information to track user interests and item popularity over time. Tracking these features is particularly meaningful in recommendation algorithms because the preferences of the users usually evolve throughout time. Trends, contexts, and moods are some factors that can cause variations in the preferences of the users. For instance, it is reasonable to assume music taste is profoundly affected by the current mood of the user, if the user is a student listening to music during the final exams period, the chances of him picking music suitable for studying are higher during that period. Therefore, to produce more accurate recommendations, the system should be able to detect those types of time drifting patterns, which is

not possible with traditional CF algorithms that discard the time dimension. The winning solution for the Netflix competition had a big part highlighting the benefit of exploiting the time context information in Collaborative Filtering [65]. Moreover, Vinagre et al. [110] recently reviewed time-exploiters contributions and concluded that despite most approaches not being able to outperform the non-time exploiters' state-of-the-art algorithms, it is clear that the usage of time significantly enhances the recommendation capability of the baseline time-agnostic algorithms.

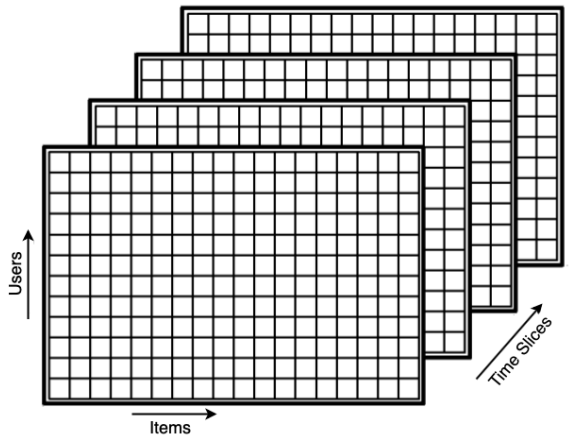


Figure 2.6: $User \times Item \times Time$ recommendation space.

Collaborative filtering algorithms that exploit time can be classified into two categories: *time-aware* and *time-dependent* [99, 110]. In the literature, many adaptations of the traditional Collaborative Filtering algorithms try to capture the temporal dynamics, especially memory-based and matrix factorization algorithms [5, 66, 67]. For the sake of simplicity, in this work, we will only focus on memory-based and matrix factorization CF techniques that use time as contextual information.

2.3.1 Time-aware

Time-aware algorithms generate recommendations for periods (e.g., hours, days, weeks, months) in a temporal cycle. For instance, if a user is considering watching a movie on the weekend, the algorithm would only recommend items which are considered appropriate to watch on the weekends. According to Adomavicius et al. [1] there are three different methodologies for using contextual information (in our case, temporal information):

1. **Pre-filtering.** Filtering the input by considering the time phase of the recommendation. For instance, ignoring ratings of movies that were given during the week if the recommendation is during the weekend.

2. **Post-filtering.** Filtering the unsuitable items from the output recommendations, for example, using a traditional time-agnostic model and then removing the items that

are not in the appropriate time phase.

3. **Modeling.** Including the time features during the learning phase of the model. The final model should be able to produce relevant recommendations considering the time phase for which the recommendations are needed.

2.3.2 Time-dependent

Time-dependent or *temporal* CF refers to algorithms that use time context information as a chronological sequence, such as a data stream. These algorithms try to capture the temporal effects that underlie the sequential data. The main objective of *time-dependent* algorithms is to adjust rapidly to changes, such as user preference changes or drifts in item popularity [110]. An example of a simple time-dependent approach is to give more importance to recent ratings rather than using all the ratings equally.

Memory-based models which were adapted to become time-dependent are mainly based on the idea of giving more relevance to the recent observations. The proposed approaches are usually adaptations of the traditional algorithms incorporating decay functions or sliding-time-windows [110].

- *Decay functions* reduce the importance of past data gradually. An example of a decay function is:

$$f(t) = e^{-\alpha t} \qquad 0 < \alpha \leq 1, \tag{2.17}$$

  where t is the time elapsed since a moment in time, and α is a parameter that controls the decay rate.

  The value of the decay function is then used as a weight in the similarity computation step, causing items to be less similar if their ratings are far apart in time.

- *Sliding-time-windows* technique instead of considering all the available data, it only uses a "window" of data that contains K records, for instance, the 50 most recent ratings, if K is equal to 50. Alternatively, a window can be defined by a time interval, for example, all the ratings from the past two weeks.

Matrix factorization models have also been modified to capture temporal effects in sequential rating data. In 2009, Koren [66] proposed an extension to his popular SVD++ model, to deal with temporal effects. In the proposed TimeSVD++ model, the author uses time-variant biases with decay functions and time-windows techniques. This model has a big impact in the literature because it played a significant role in the solution that won the Netflix contest. Xiong et al. [113] presented a Temporal CF based on Bayesian probabilistic tensor factorization (BPTF). In this approach, Xiong et al. split the time dimension into w time slices, creating a three-dimensional tensor. Finally, the authors applied the BPTF model to the tensor.

2.4 Biclustering

2.4.1 Clustering and biclustering

Clustering, or **cluster analysis**, is the process of grouping a set of data objects into multiple groups or clusters so that objects within a cluster have high similarity (high intra-cluster similarity), but are very dissimilar to objects in other clusters (low inter-cluster similarity). These (dis)similarities are estimated based on the attribute values which describe the objects [42].

In Machine Learning, clustering is considered an unsupervised task, as it looks for previously undetected patterns in a dataset with no pre-existing labels/outcomes. Hence, clustering can lead to the discovery of previously unknown groups of objects inherent in the data. Clustering can also be used for outlier detection, as it permits to identify values that significantly deviate from any of the discovered clusters. Clustering has been widely used in countless applications from different fields (e.g., bioinformatics, social science, business marketing, fraud detection) [42]. However, traditional clustering methods exhibit some limitations when applied to specific problems. Some of these limitations result from the fact that traditional clustering algorithms mislay some valuable information because they can only be applied either to the rows or the columns of a data matrix, separately, disregarding the other dimension. For instance, considering a U-I matrix from the recommendation scenario, traditional clustering methods are only able to group users based on their ratings for all the items in the matrix or to cluster items based on the ratings from all users, which means that these methods can not find groups of users that only share preferences under a subset of the items.

In this context, clustering algorithms find global patterns in data, such as users who have similar taste across all the items, but, they miss local patterns, as a subset of users who rated a similarly a subset of items. To deal with this limitation, an advanced clustering technique, called **biclustering**, was developed.

As opposed to one-way clustering techniques that can be applied to either the rows or the columns of the data matrix, separately, biclustering is a technique that clusters rows and columns simultaneously. Consequently, biclustering produces local models, instead of a global model, and as a result, can identify subgroups of objects that are similar only under a specific subgroup of attributes. Figure 2.8 illustrates the main differences between clustering and biclustering methods. Figure 2.8 a) and b) shows an example of applying clustering to the rows and columns of a data matrix, individually. We can see that clustering was able to discover a cluster of rows ($\{x_1, x_2\}$), and a cluster of columns ($\{y_3, y_5\}$). Whereas, the biclustering technique in Figure 2.8 c), can discover different sub-matrices (biclusters) in the matrix, that show similar or coherent behaviour, such as $(\{x_2, x_4\}, \{y_1, y_3, y_5\})$.

Co-clustering, simultaneous clustering, block clustering, subspace clustering, bidi-

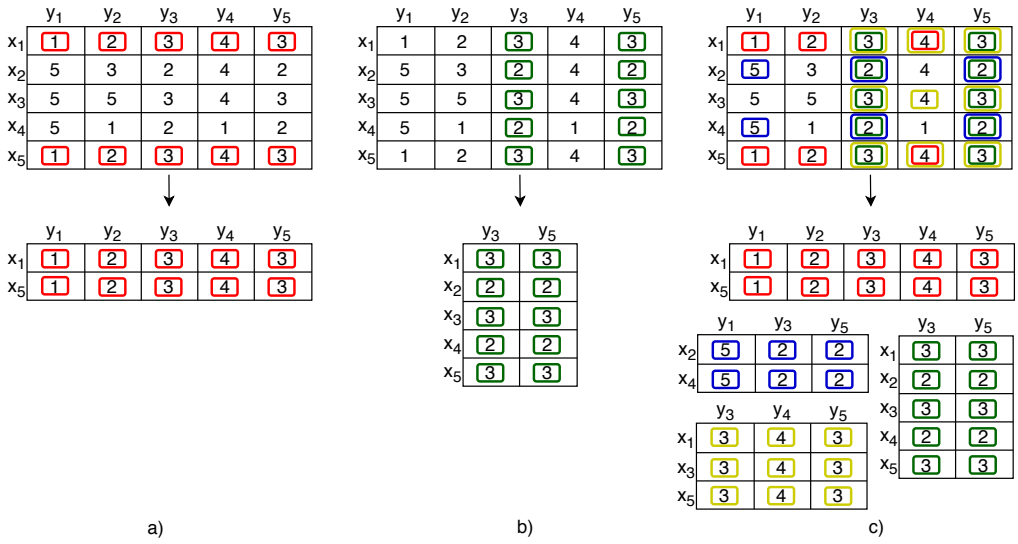


Figure 2.8: Clustering and biclustering example.

mensional clustering, and two-way clustering are some examples of terms that are often used to address the same task as biclustering. Bellow, we present a more formal definition of this advanced clustering technique.

Definition 2.4.1. Given a matrix $A = (X, Y)$, with a set of rows $X = \{x_1, ..., x_n\}$ and a set of columns $Y = \{y_1, ..., y_m\}$, where the element a_{ij} relates row i and column j, the **biclustering task** is to identify a **biclustering solution** which is a set of biclusters $B = \{B_1, ..., B_p\}$ so that each $B_k = (I_k, J_k)$ satisfies a particular criteria of homogeneity and significance, where $I_k \subseteq X$, $J_k \subseteq Y$, and $k \in \mathbb{N}$ [47, 48].

Definition 2.4.2. A **bicluster** $B = (I, J)$ is a $r \times s$ submatrix of a matrix $A = (X, Y)$, where $I = (i_1, ..i_r) \subseteq X$ is a subset of rows and $J = (j_1, ..., j_s) \subseteq Y$ is a subset of columns.

In Figure 2.8 c) we we highlighted four different biclusters: $(\{x_1, x_5\}, \{y_1, y_2, y_3, y_4, y_5\})$; $(\{x_2, x_4\}, \{y_1, y_3, y_5\})$; $(\{x_1, x_2, x_3, x_4, x_5\}, \{y_3, y_5\})$; $(\{x_1, x_3, x_5\}, \{y_3, y_4, y_5\})$. However, it is possible to identify many others, even for this simplistic example, for instance, all the possible sets, with size larger than 1, obtained through the combinations of the elements that compose the found biclusters. The complexity of the biclustering problem may depend on the problem formulation, however, finding all the biclusters through an exhaustive search is an NP-complete task [76]. Given this, the majority of the biclustering algorithms use heuristic mechanisms to reduce its complexity.

2.4.2 Bicluster Type

Different biclustering algorithms with different parameterizations usually find different clustering solutions. Thus, whenever one wants to apply a biclustering technique to a

specific domain, it needs to ao consider the type of biclusters relevant to solve the problem. Madeira and Oliveira [76] identified four major classes of biclusters:

1. Biclusters with constant values.
2. Biclusters with constant values on rows or columns.
3. Biclusters with coherent values.
4. Biclusters with coherent evolutions.

The first three classes evaluate the numeric values directly in the data matrix and try to find subsets of rows and subsets of columns with similar behaviours. These behaviours can be observed on the rows, on the columns, or in both dimensions of the data matrix, as in Figure 2.9 a), b), c), d), and e). The fourth class aims to find coherent behaviours regardless of the exact numeric values in the data matrix. As such, biclusters with coherent evolutions consider the elements in the data matrix as symbols that correspond to a given order. An example of coherent evolutions are the order-preserving biclusters, as in Figure 2.9 f), g), and h).

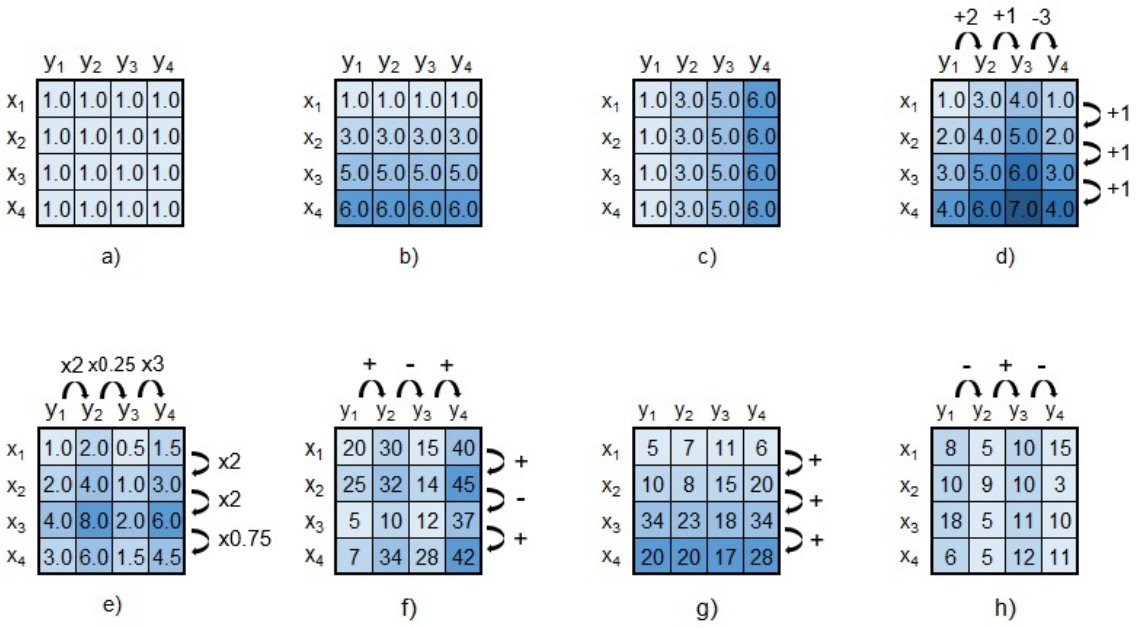


Figure 2.9: Examples of different types of biclusters. (a) Constant value, (b) constant-values on rows, (c) constant-values on columns, (d) coherent values (addictive model), (e) coherent values (multiplicative model), (f) overall coherent evolution, (g) coherent evolution on the rows, (h) coherent evolution on the columns (adapted from [76]).

In this work, we perform the biclustering task on U-I matrix from feedback data. This means that biclustering allows us to identify sets of users sharing preferences across a subset of items.

The first type of biclusters, *constant-value biclusters*, is the simplest form of a bicluster. This type of bicluster is described in Def. 2.4.3. In the U-I matrix, a constant-value bicluster exists when each user in the bicluster gave the exact same constant rating to each item in the bicluster.

Definition 2.4.3. Given a matrix $A = (X, Y)$, with a set of rows $X = \{x_1, ..., x_n\}$ and a set of columns $Y = \{y_1, ..., y_m\}$, a submatrix $B = (I, J)$ (where $I \subseteq X$, $J \subseteq Y$) is a **constant bicluster** iff $\forall x \in I$, $\forall y \in J$, the elements $a_{x,y} = c$ where c is a constant.

Biclusters with constant-values on rows or columns are another type of biclusters that can be valuable when applied in Collaborative Filtering contexts. In the case of biclusters with constant values on the columns, we identify groups of users who have the same preference about the items in the bicluster. Whereas in the case of biclusters with constant values on the rows, we can recognize items which are usually rated with the same value by some users. These types of biclusters are defined in Def. 2.4.4.

Definition 2.4.4. Given a matrix $A = (X, Y)$, with a set of rows $X = \{x_1, ..., x_n\}$ and a set of columns $Y = \{y_1, ..., y_m\}$, a submatrix $B = (I, J)$ (where $I \subseteq X$, $J \subseteq Y$) is a **constant-values on rows bicluster** iff $\forall x \in I$, $\forall y, h \in J$, the elements $a_{x,y} = a_{x,h}$; or a **constant-values on columns bicluster** iff $\forall x, u \in I$, $\forall y \in J$, the elements $a_{x,y} = a_{u,y}$.

Biclusters with coherent values, defined in Def. 2.4.5 are a more relaxed variation of the previously defined biclusters. It allows to group users that, despite having different rating standards, share coherent behaviours across the items, assuming there is an additive or multiplicative model in the pattern. For instance, in Figure 2.9 d), all the values can be obtained by additive adjustments in the rows and columns.

Definition 2.4.5. Given a matrix $A = (X, Y)$, with a set of rows $X = \{x_1, ..., x_n\}$ and a set of columns $Y = \{y_1, ..., y_m\}$, a submatrix $B = (I, J)$ (where $I \subseteq X$, $J \subseteq Y$) is a **coherent addictive bicluster** iff $\forall x, u \in I$,$\forall y, h \in J$, the elements $a_{x,y} - a_{x,h} = a_{u,y} - a_{u,h}$; or a **coherent multiplicative bicluster** iff $\forall x, u \in I$,$\forall y, h \in J$, the elements $a_{x,y}/a_{x,h} = a_{u,y}/a_{u,h}$.

Finally, *order-preserving biclusters* are the most relaxed version of biclusters that we consider. A group of users and items are an order-preserving bicluster if there is a linear order across the rows/columns, regardless of the uniformity of the actual values in the bicluster. This means that order-preserving biclusters group users that have a similar order of preference in the items. This type of biclusters is defined in Def. 2.4.6.

Definition 2.4.6. Given a matrix $A = (X, Y)$, with a set of rows $X = \{x_1, ..., x_n\}$ and a set of columns $Y = \{y_1, ..., y_m\}$, a submatrix $B = (I, J)$ (where $I \subseteq X$, $J \subseteq Y$) is an **order-preserving on rows bicluster** iff there exists a sequence of rows $(x_1, ..., x_{|I|})$, $\forall x_i \in I$, such that the elements $a_{x_i,y} < a_{x_{i+1},y}$; or an **order-preserving on columns bicluster**

iff there exists a sequence of columns $(y_1, ..., y_{|J|})$, $\forall y_i \in J$, such that the elements $a_{x,y_i} < a_{x,y_{i+1}}$.

As pointed out by Madeira and Oliveira in [76], different biclustering algorithms are designed not only to discover specific types of biclusters but also different *bicluster structures*. While some algorithms are designed to find only one bicluster, others assume the existence of several. Madeira and Oliveira enumerate eight different types of bicluster structures. In this work, we do not have any restrictions regarding the bicluster structure, and the overlapping between biclusters should be allowed. Biclusters overlapping, in the context of a U-I matrix, means that a user may belong to more than one group, which makes sense because it is plausible that a user might share some preferences with other users on a subset of items, despite disagreeing with them in another subset of items. Having this in mind, we are interested in algorithms that discover arbitrarily positioned overlapping biclusters [76]. A final aspect to consider when choosing a biclustering algorithm is the tolerance to noise and missing values. In real-world scenarios, it is rare to find "perfect" clusters, the data is usually noisy, and so we might need to tolerate a certain amount of noise in order to find interesting biclusters that can be masked by noise.

Chapter 3

Related Work

This chapter describes the dominant streams of research related to our project. In Section 3.1, we discuss the research works that explored the usage of biclustering in CF approaches. Then, in Section 3.2, we provide an overview of how some neighborhood-based CF algorithms have been adapted to exploit the time information in Collaborative Filtering.

3.1 Biclustering Collaborative Filtering

In this section, we present the most notorious contributions in the field of Collaborative Filtering incorporating the biclustering technique. We separate the approaches according to their recommendation task, describing the main ideas, and identifying some of its advantages and limitations.

3.1.1 Rating Prediction Approaches

George and Merugu first introduced biclustering as a tool to improve Collaborative Filtering in 2005 [34]. In this work, they used weight Bregman co-clustering [8] to group users and items. This algorithm focus on a variation of bicluster (partial co-clustering) that assumes the existence of a fixed number of non-overlapping biclusters, firstly introduced by Hartigan, [44]. Their co-clustering CF approach is based on the idea that the input data's missing ratings can be predicted using a suitable low parameter approximation of the input rating matrix, similar to SVD-based CF approaches. Following this concept, they use co-clustering as a tool to find low parameter approximations. The Bregman co-clustering returns co-clusters and summary statistics derived from the co-clustering that are used to construct a matrix approximation for the input data matrix. For this particular application, the summary statistics are the rating averages of the users, items, and co-clusters. According to the authors, it permits a more reasonable rating approximation than just considering the average value of the co-cluster corresponding to the value we want to predict, which is plausible since it takes into account the biases of individual users and items. After

computing the reconstructed approximate matrix, its values are considered for the rating prediction task. The authors designed incremental and parallel versions of the original co-clustering algorithm to build an efficient real-time CF mechanism, capable of updating the biclusters as new users and ratings enter the system. The results of the work show that their approach achieves satisfactory accuracy compared with baseline matrix factorization models but at a lower computational cost.

In 2007, Castro et al. [123] suggested a biclustering CF methodology, using a heuristic immune-inspired biclustering technique, denoted BIC-aiNet, that finds biclusters with coherent values. Their methodology is based on two main stages: the biclusters' generation, and the similarity computation between the users and the resulting biclusters. After their artificial immune-inspired network generates the biclusters, they are used to predict how the active user would rate a specific item. The rating prediction is performed by searching for the biclusters that include the active user and item, and then, calculating the residue of each bicluster through mean-squared residue (MSR) [18]. The mean-squared residue of a bicluster, MSR(b) , is defined as:

$$\text{MSR(b)} = \frac{1}{|I||J|} \sum_{i \in I} \sum_{j \in J} (a_{ij} - a_{Ij} - a_{iJ} - a_{IJ})^2, \tag{3.1}$$

where $|I|$ is the number of rows (users) of the bicluster, $|J|$ is the number of columns (items), a_{ij} is the value in row i and column j, a_{Ij} and a_{iJ} represent the mean value of row i and column j, respectively, and a_{IJ} is the mean value of the entire bicluster. Finally, the bicluster with the smaller residue value is selected, and the average of its movies' ratings is used as the prediction. This methodology was evaluated in both rating prediction and Top-*N* recommendation scenarios, achieving better scores than the methods used for comparison, being the constant version of BCF [104], one of those.

França et al. [21] also used an immune inspired biclustering algorithm, MOM-aiNET [20] to obtain biclusters used to predict missing ratings for recommendation purposes. Their algorithm generates biclusters with a controlled percentage of missing values. Then, they use a quadratic programming approach to predict those values based on trying to minimize the mean-squared residue of the bicluster. This approach is viewed as an improvement of the one developed by Castro et al. [123] (described above), as the latter simply replaces the missing values in a bicluster by the average of the bicluster's values, not reflecting the trends obtained by coherent biclusters. Whereas, in this approach, the missing values are viewed as variables that should be obtained to minimize the residue.

A more recent work in the direction of biclustering CF was published in 2014 by Liang and Leng [70], who proposed a CF algorithm utilizing information-theoretic co-clustering (ITCC) [23] as biclustering algorithm. Their approach (ITCCCF) consists in performing two-way clustering to discover a set of user clusters and a set of item clusters. After that, they compute the preference of a user u for an item cluster $\hat{i}$, ($P_{u,\hat{i}}$), for each pair user-item cluster as:

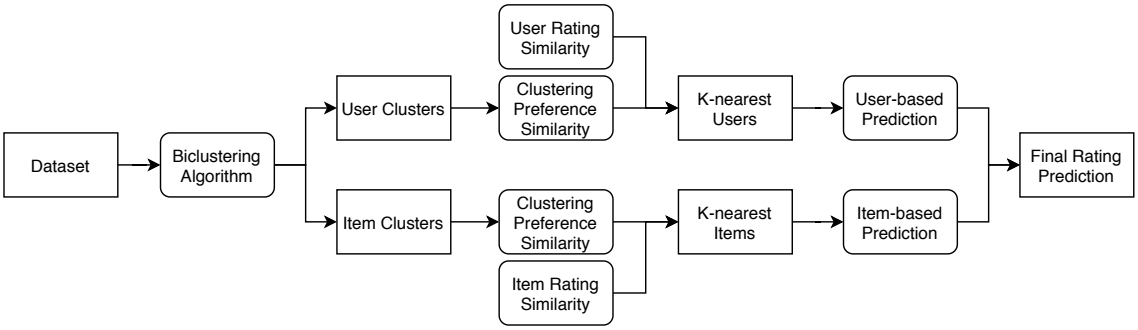


Figure 3.1: Outline of the rating prediction approach (ITCCCF) proposed by Liang and Leng [70].

$$P_{u,\hat{i}} = \frac{|I_u \cap \hat{i}|}{|I_u|}, \tag{3.2}$$

where I_u are the items that the user *u* rated and $\hat{i}$ are the items from the cluster. A matrix of user-cluster preferences is constructed and its values are used to obtain the cosine similarity between each pair of users, $sim_p(u_1, u_2)$. This approach then combines the previous clustering preference similarity with a rating similarity determined through Person's correlation coefficient to find the k most similar users. The k-nearest users are selected and the User-based CF is used to generate the rating prediction. Moreover, the whole process is also repeated for the item clusters and the Item-based CF is used to generate a new rating prediction. The final rating prediction is a linear combination that fuses the ratings from the user-based and item-based approaches. In order to evaluate their approach, using real-world datasets, the authors compared it with 5 state of the art methods, some already mentioned in this work (UBCF [92], IBCF [96], CBCF [97], SF1 [111], BCC [34]), being able to surpass all of them regarding prediction accuracy. Figure 3.1 provides an overview of how this fusion rating technique works.

In 2017, Kant and Mahara [60] also proposed a fusion-based approach called Nearest Biclusters Collaborative Filtering with Fusion (NBCFu), to address rating prediction using biclustering. In this approach, the xMOTIFs algorithm [81] is used to generate the biclusters that are viewed as quality neighbors of users and items. After the biclusters' generation, the authors use the CjacMD measure [103] (Equation 3.3) to obtain the k-nearest biclusters of the users and items. This similarity measure combines Mean Measure of Divergence (MMD) [56], Jaccard Similarity, and Cosine Similarity.

$$sim(u, U_b)_{JacMD} = sim(u, U_b)_{cos} + sim(u, U_b)_{Jaccard} + sim(u, U_b)_{MMD}. \tag{3.3}$$

After that, User-based CF and Item-based CF using cosine similarity are applied to the users/items found in most similar biclusters. Finally, the rating prediction of both approaches is combined through a weighted sum, optimized using gradient descent, to

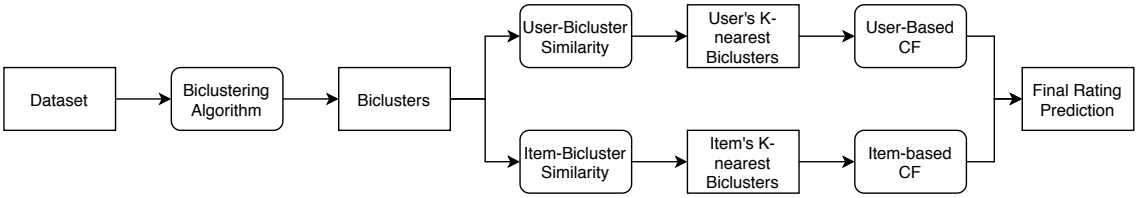


Figure 3.2: Outline of the rating prediction approach (NBCFu) proposed by Kant and Mahara [60].

generate a final and more accurate rating prediction. The authors claim that their approach has better accuracy results than some state of the art methods, including SVD++ [64]. Figure 3.2 shows an overview of the NBCFu.

Elnabarawy et al. [29] examined the viability of using biclustering ARTMAP (BARTMAP) [114] for recommendation purposes. BARTMAP is a biclustering algorithm that utilizes the Adaptive Resonance Theory neural network model [39]. The ratings are predicted through a normalized weighted sum of the ratings that the other users gave to the item in the same bicluster, weighted by their correlation value with that user. The similarity score between an active user u_x and other user u_y belonging to the same bicluster is given by the person correlation coefficient of their ratings in the bicluster. The rating prediction is computed as:

$$r_{u_x,i_t} = \bar{r}_{u_x} + \frac{\sum\limits_{y \in U_b} (r_{u_y,i_t} - \bar{r}_{u_y}) \cdot s(u_x, u_y)}{\sum\limits_{y \in U_b} |s(u_x, u_y)|}, \tag{3.4}$$

where r_{u_x,i_t} is the rating of an user u_x to an item i_t. This formula is an adaptation of the prediction used in User-Based CF in order to only use data from the bicluster, instead of relying on the entire dataset. Using this prediction approach, the users with the highest positive correlation with that user, have the most impact on the prediction. The algorithm's performance was compared against other collaborating filtering techniques, and it performed similarly to the previously mentioned approach, BIC-aiNet [123].

Recently, Singh and Mehotra [101] introduced a new biclustering based CF technique, BBCF. In this work, the authors use biclustering as a preprocessing step to scale CF approaches. Once the biclustering algorithm is executed, the users are compared with the found biclusters using the similarity defined in Equation 3.5, based on the items they have in common.

$$sim(u, b) = \frac{|I_u \cap I_b|}{|I_b|} \tag{3.5}$$

Then, similarities between users and biclusters are weighted with the number of users in the bicluster so that biclusters with more users are privileged, as:

$$\text{WF}(i, b) = sim(u, b) \cdot |U_b|. \tag{3.6}$$

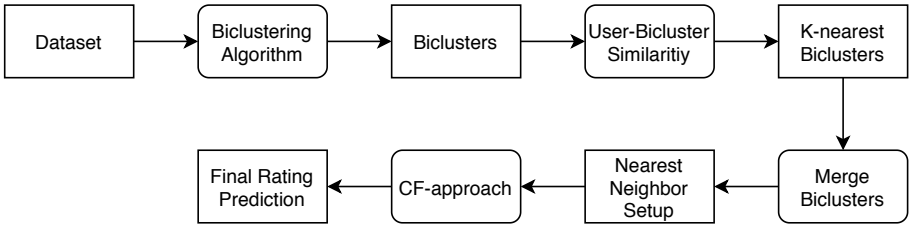


Figure 3.3: Outline of the rating prediction approach proposed (BBCF) by Singh and Mehotra [101].

After that, the K-nearest biclusters of each user are merged, creating a larger bicluster, for each one, referred to as Nearest Neighbor Setup (NSS). The NSS of each user is viewed as a denser subspace of the U-I matrix, that includes users similar to the respective user, in a subset of items. Finally, a classic CF-approach, such as the Item-based, uses the personalized NSS of each user, as a smaller input U-I matrix, instead of the entire dataset, to generate the rating prediction. According to the authors, this approach tackles the scalability and sparsity problems that deteriorate the performance of memory-based CF methods, through the usage of the personalized NSSs instead of the entire U-I matrix. The authors evaluated BBCF using QUBIC and Item-based CF as biclustering and CF approaches, respectively, using both predictive and classification metrics against CF baseline approaches, achieving better scores in most scenarios. However, they also highlight a clear drawback of their methodology. They point out that their approach cannot generate a prediction/recommendation for as many users/items as traditional methods. This handicap occurs because the CF algorithm will be using only a subset of the users/items as database, so it can not consider all the users/items when modelling.

3.1.2 Top-*N* Recommendation Approaches

Symeonidis et al. [105] proposed a neighborhood-based CF algorithm (NBCF) that uses biclustering to improve scalability and accuracy of CF. They also created a similarity measure, defined in Equation 3.5, to identify biclusters that better reflect the users' preferences. Their approach can be combined with biclustering algorithms that find biclusters with either constant or coherent values. The authors used Bimax and xMOTIFs, respectively. However, if we consider biclusters with constant values, we can only discover sets of users and items correlated by the same rating values. Whereas, a coherent pattern allows to find users and items correlated with more complex behaviors, such as users that, for a subset of items, exhibit coherent ratings. After biclusters computation, they measure the similarity of the active user with each bicluster, finding the k-nearest-biclusters to the user. Then, to capture the influence that each item belonging to a bicluster has to the active user, they use *Weighted Frequency (WF)*, as in Equation 3.6. Finally, after having all the *Weighted Frequencies* scores, they sum *WF* values of each item. The final Top-*N* recom-

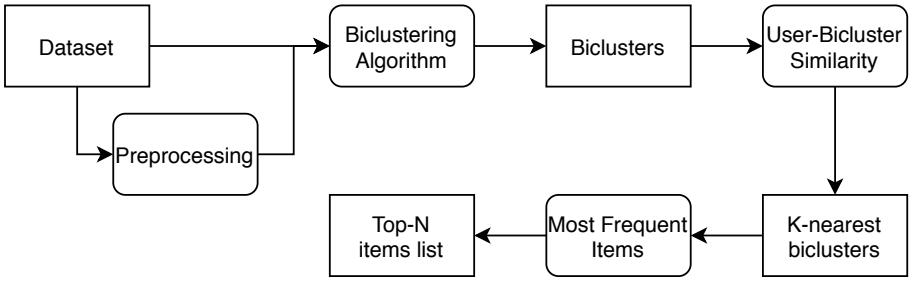


Figure 3.4: Outline of the Top-*N* recommendation approach (NBCF) proposed by Symeonidis et al. [105].

mended items are the N items with the highest sum of WF. Figure 3.4 summarizes the process of this Top-*N* recommendation approach. The authors reported that both proposed approaches (using constant or coherent biclusters) achieved better results in terms of accuracy and efficiency than classic CF algorithms such as User-based CF [14], Item-based CF [96], and clustering-based CF (CBCF) [115]. They also concluded that despite being slightly less efficient, the approach using coherent biclusters discovery could outperform the one using constant biclusters discovery regarding accuracy.

Specially for the Top-*N* recommendation task, Alqadah et al. in 2015 [4] also proposed a biclustering neighborhood-based CF method (BCN). In this work, the authors use properties from the field of Formal Concept Analysis (FCA) [33]. FCA is related to biclustering since both discover sub-matrices with regularities among their elements [58], and FCA algorithms can be adapted to enumerate and order biclusters, which can be used to identify neighborhoods of closely related biclusters. Alqadah et al. took advantage of these FCA properties to build neighborhoods/personalized biclusters for active users. Their approach finds the "smallest" bicluster containing the active user u, which is the bicluster containing the fewest number of users and the greatest number of items. Then, exploring the bicluster neighborhood through FCA properties, it finds similar users to u, and appends the items in the neighborhood to a candidate set of items. Finally, the candidate items are ranked by combining a global and a bicluster neighborhood similarity, and the top n items are returned as recommendations. The global distance between a user and an item, $g(u, i')$, measures the similarity between a user u and and item i' in the entire matrix as:

$$g(u, i') = \frac{\sum_{i \in I} J(i, i')}{|I|}, \tag{3.7}$$

where $J(i, i')$ is the Jaccard index defined over the set of all users who interact with i and the set of those who interact with i', and I is the set of items belonging to biclusters. On the other hand, the local/neighborhood similarity, $l(u, i')$, captures similarities between locally similar users and items, aggregating the bicluster similarity [3] of all biclusters in which i occurs to the smallest bicluster. The authors claim that their approach

is superior to those computing the biclusters offline since, in BCN, they map a user to a bicluster on demand. However, the BCN framework was designed to work on implicit feedback recommendations, and thus it can only be applied to binary data. Nevertheless, to surpass this limitation, it could be an interesting research stream to study the potential of an FCA adaptation capable of dealing with numerical rating data. For instance, Juniarta in a very recent thesis [58] proposed and FCA extension to deal with numerical matrices and discover various types of biclusters which could be very interesting for recommendation purposes.

3.2 Time Dimension in Neighborhood-based Collaborative Filtering

In this section, we review some significant research contributions regarding temporal dynamics in neighborhood-based CF, highlighting how the time information was handled and how it impacted the solutions.

3.2.1 Temporal CF

Ding and Li contribution [24] was one of the first Collaborative Filtering works to study the impact of the temporal information in the recommendation scenario. In their work, the authors presented a new item-based CF algorithm that assigns weights to the items in the recommendation, so that an item that was rated recently has a bigger impact on the prediction phase than an item that was rated longer ago. During the prediction phase, each rating is assigned with a weighted to the time t defined by:

$$f(t) = e^{-\alpha t} \qquad 0 < \alpha \leq 1, \tag{3.8}$$

where t is the time elapsed since a particular moment, and α is the decay parameter that controls how fast old data decays. Higher values of α mean lower importance of historical data compared to the more recent one. The authors studied what would be the optimal values for the parameter α and reported that a fixed value for this parameter does not guarantee good prediction performances. It is claimed that this behaviour is justified by the instability of the users' purchase habits. Furthermore, the interest in some items lasts longer than in other sets of items. In order to get around this issue, their methodology assumes that to each user, similar items have similar decay rates. Thus, they use k-means to find item clusters and compute an α value for each pair user-item cluster. Each α value is obtained by searching for the value that better explains the user's behaviour in each item cluster (see [24] for more details about the optimization problem). After finding the optimal α values, the rating prediction is obtained by:

$$\hat{r}_{u,i} = \frac{\sum_{j \in N} f(t_{u,j}) sim(i,j) r_{u,j}}{\sum_{j \in N} |f(t_{u,j}) sim(i,j)|}, \tag{3.9}$$

where $t_{u,j}$ is the time that the rating from the user u to the item j was produced. The empirical studies in this work suggest that their approach improves the prediction capability of the item-based CF algorithms.

In 2006, the same authors [25] proposed a new CF algorithm, namely recency-based weighting method. Instead of discounting data using a time-weight decay function such in [24], this approach takes into account the data distribution, rather than just considering the time of the records. This recency-based technique intends to tackle a problem known as *concept drifting*. In CF, concept drift means that user purchase preferences that we want to predict are sensitive to time, thus in constant drifting. For example, imagine Figure 3.5 represents the ratings that a user (Bob) gave to comedy movies. From Figure 3.5, we can see that Bob's preferences for comedy movies changes in time. From the timestamp t_0 to t_1, and from t_2 to t_3, we can see that Bob liked comedy since he rated the movies with high scores. However, during the period from t_1 to t_2, the scores were significantly lower. This type of drifts in the users' data is particularly common in CF scenarios, so Ding et al. 's approach addresses this issue by assigning a higher weight to data that follow the same trend as the most recent ratings. In the example from Figure 3.5, the data points from t_2 to t_3 are considered the latest trend, so, their approach assigns more weight not only to the data from t_2 to t_3, but also to the data from t_0 to t_1 (because it has a similar distribution).

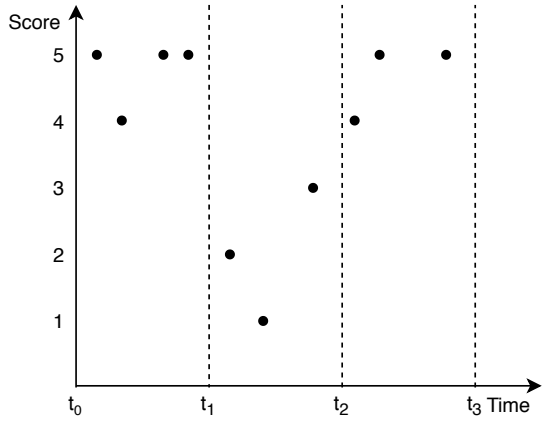


Figure 3.5: Concept Drift in CF (adapted from [25])

Their approach is a adaptation of the item-based CF that assigns different weights to different items, taking into account the deviation of the rating on the item from the most recent rating as:

$$w_{u,i} = (1 - \frac{|r_{u,i} - r_{u,n}|}{M})^{\alpha} \tag{3.10}$$

where $r_{u,i}$ is the rating of the target user on the i-th item, and $r_{u,n}$ represents the most recent rating of the target user on the nearest neighbour items. M is the maximum value in the rating scale, and α a parameter that can be adjusted to tune the performance. So, from this function, we can observe that the more a item's rating deviates from the most recent neighbor rating, the lower the weight of the item. Finally, the prediction phase considers the weight from Equation 3.10:

$$\hat{r}_{u,i} = \frac{\sum_{j \in N} w_{u,j} sim(i,j) r_{u,j}}{\sum_{j \in N} |w_{u,j} sim(i,j)|}, \tag{3.11}$$

where $w_{u,j}$ represents the weight assigned to item j. Besides the introduction of a weight for each item, they also designed and tested a new similarity function that they considered to be appropriate for the item-based CF algorithms. Their experimental results show that their approach can substantially improve the prediction accuracy of the traditional item-based CF.

In a more recent work, Liu et al. [73] introduced decaying time functions in both the similarity computation and in the rating prediction steps of the item-based CF.

3.2.2 Time-aware CF

Yuan et al. [119] in 2013 adapted the traditional User-based CF in order to incorporate temporal information. Their system recommends places for users to visit, referred as points-of-interest (POIs). Their approach is motivated by the fact the humans tend to have periodic behaviour throughout the day. They started by augmenting their user-POI (U-I) data, incorporating time slots based on the hour of the day the user visited the POI. Hence, their input data is a binary user-timeslot-POI data cube, where each element $r_{u,t,i}$ represents the activity of a user u, at a POI i at time slot t, where $r_{u,t,i} = 1$ if the user u has visited/interacted with the POI i at the time slot t, and $r_{u,t,i} = 0$ otherwise.

They extended the traditional cosine similarity between to users (presented in Section 2.1) and modified it as:

$$sim(u,v)_{twarecos} = \frac{\sum_{t \in T} \sum_{i \in I} r_{u,t,i} r_{v,t,i}}{\sqrt{\sum_{t \in T} \sum_{i \in I} r_{u,t,i}^2} \sqrt{\sum_{t \in T} \sum_{i \in I} r_{v,t,i}^2}}, \tag{3.12}$$

which takes into account the time slot of the records when computing the similarity between the users. This means that if two users usually visit the same POIs at the same time, the similarity score between the two will be high. However, the authors found out that the system's performance suffered from the sparsity of the data. To address this weakness, they studied the similarity between time slots and discovered that some time slots were similar (users visiting the same POIs at the two different time slots). So they enhanced their methodology introducing a smoothing component that includes the similarity between time slots. Specifically, computing a new interaction $\tilde{r}_{u,t,i}$ as:

$$\tilde{r}_{u,t,i} = \sum_{t' \in T} \frac{\rho_{t,t'}}{\sum_{t'' \in T} \rho_{t,t''}} r_{u,t',i}, \tag{3.13}$$

where $\rho_{t,t'}$ is the similarity between two time slots. This new interaction $\tilde{r}_{u,t,i}$ replaces the original interaction $r_{u,t,i}$ in the similarity computation, creating a new smoothed similarity function:

$$sim(u,v)_{stwarecos} = \frac{\sum_{t \in T} \sum_{i \in I} \tilde{r}_{u,t,i} \tilde{r}_{v,t,i}}{\sqrt{\sum_{t \in T} \sum_{i \in I} \tilde{r}^2_{u,t,i}} \sqrt{\sum_{t \in T} \sum_{i \in I} \tilde{r}^2_{v,t,i}}}. \tag{3.14}$$

This new similarity function returns high values not only if two users visited the same POIs during the same time slot but also in different but similar time slots. The authors concluded that their time-aware methodology surpassed the classic time-agnostic User-based CF when recommending points-of-interest, as well as the Dings et al.'s temporal approach [24] that we already described above.

Chapter 4

Evaluating the Potentialities of BicPAM

In this chapter, we study the potentialities of BicPAM, a biclustering algorithm proposed by Henriques and Madeira [48], which seems to be a valid option for the biclustering-based CF, since it can efficiently discover constant, coherent, and order-preserving bicluster solutions with varying levels of overlapping, missing values, and noise. We also compare it against two state-of-the-art biclustering approaches that were already adopted by other authors to find biclustering in recommendation scenarios, using our own synthetic benchmark datasets.

4.1 BicPAM - State-of-the-art Pattern-based Biclustering

The BicPAM algorithm is a state of the art biclustering algorithm proposed in 2014 by Henriques and Madeira [48]. BicPAM relies on derived principles from the pattern mining field. Therefore it is classified as a pattern-based biclustering algorithm. Pattern mining-based biclustering approaches usually perform efficient, exhaustive searches and produce flexible bicluster solutions (an arbitrary number, structure, and position of biclusters).

Most of the previously proposed pattern-based approaches assume that the biclusters in the data follow the constant model [83, 85, 98]. However, BicPAM can discover more complex types of patterns in the biclusters such as coherent (addictive/multiplicative) or order-preserving patterns, which can be essential for the discovery of more interesting groups in the recommendation data. Moreover, BicPAM relies on strategies that allow the algorithm to handle noise and missing values in the data.

BicPAM consists of an ordered conjunction of three major phases: *mapping*, *mining*, and *closing*, Figure 4.1 summarizes the main principles of each phase. The first phase, *mapping*, is responsible for itemizing a real-value matrix into an itemset matrix. There are available normalization and discretization options that can be used depending on the properties of the input data. This step is also responsible for offering different methods for handling missing and noisy elements in the data. To deal with missing values, BicPAM provides tree strategies: Removal, Replacement, None (consider it as a special value). As

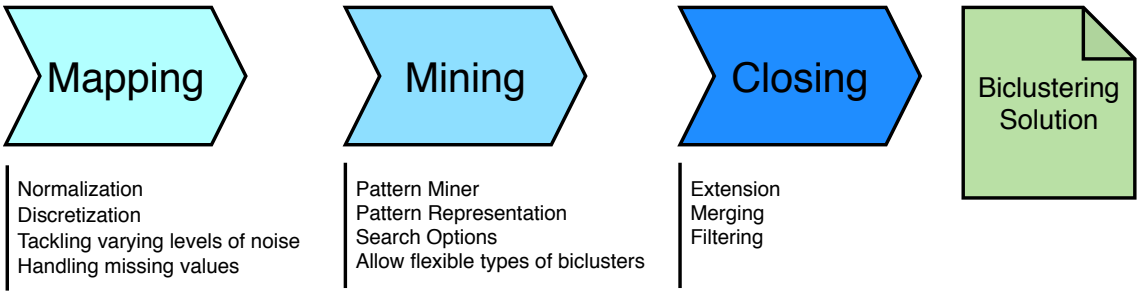


Figure 4.1: Methodology of the BicPAM algorithm.

for the noise in the data, BicPAM offers three methods that can identify noisy elements in the data, but the default strategy consists in considering two or more possible discrete values to an element, based on a threshold.

The *mining* phase is the core of the algorithm, as it is responsible for finding the biclusters. The mining of the biclusters depends on three settings: the pattern-based approach, the pattern representation, and the search strategy. BicPAM uses Frequent Itemset Mining (pattern-based approach) to search for frequent closed (pattern-representation) itemsets in the data, as the default setting. Association rule mining can also be used in the presence of domain knowledge to produce more insightful solutions. Regarding the pattern representation, considering different types of patterns lead to structurally different biclustering solutions, as Figure 4.2 highlights. While retrieving all the frequent itemsets may lead to a biclustering solution with a large number of redundant biclusters, considering only maximal itemsets can lead to incomplete solutions since it discards itemsets with fewer items/columns but higher support. The usage of closed itemsets allows overlapping biclusters if a reduction in the number of columns results in a bicluster with a higher number of rows (support of the itemset). Mining closed itemsets is usually the preferred option since it enables the algorithm to discover all the maximal biclusters in the data (see def. 4.1.4). Lastly, as for the search strategy, the choice of using an apriori-based, pattern-growth, or both combined, mostly depends on the size and density of the input data. The efficiency of the searching algorithms is profoundly affected by those factors.

Figure 4.2 is an illustration of how BicPAM, using a FIM approach, mines itemsets that are subsequently traduced into constant biclusters. In this example, we used the apriori algorithm with a minimum support threshold of 2 to search for the itemsets, however, notice that the initial matrix A needs to be transformed so that the items have their column identification. This step of concatenating the identification of the columns to the item is essential as it allows the algorithm to distinguish between values of different columns. At the end of the process, the obtained (frequent) itemsets are used to derive the resulting biclusters from the corresponding itemsets in the original matrix.

Definition 4.1.1. Given a set $B = \{i_1, ..., i_n\}$ of items, a set $X \subseteq I$ is called and **itemset**. A **transaction database** D is a multiset of itemsets, where each itemset, called a

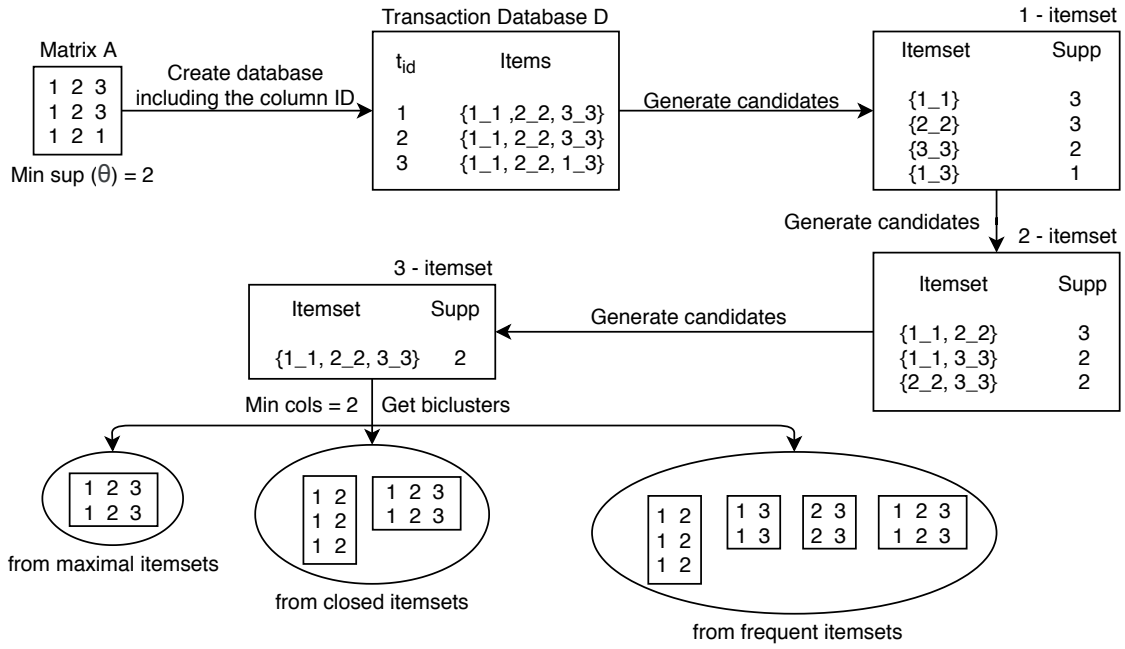


Figure 4.2: Example of mining constants biclusters from a matrix.

transaction, has a unique identifier, called a t_{id} [120].

Definition 4.1.2. Given a transaction database D the **support** of an itemset X in a dataset D, denoted supp(X), is the number of transactions in D where X appears as a subset. The support of an itemset X in D can also be represented as the fraction of transactions in D where X appears as a subset.

Definition 4.1.3. Given a transaction database D and a minimum support threshold θ, **Frequent Itemset Mining (FIM)** is the problem of computing **frequent itemsets**, that is, itemsets having support $\geq \theta$.

- a *closed itemset* is a frequent itemset that has no (frequent) superset with the same support.
- a *maximal itemset* is a frequent itemset for which none of supersets are frequent.

Definition 4.1.4. A **maximal bicluster** is a bicluster that is not included in any other bicluster. This means that it cannot be extended without the need of removing rows and columns.

Finally, the *closing* step is composed of techniques used to post-process the discovered bicluster in order to generate and manipulate the structure and quality of the final biclustering solution. One post-process technique is to extend the biclusters, including rows or columns as long as it still satisfies a criteria. Another option is to merge biclusters. Merging biclusters is based on the idea that if two biclusters overlap/share a significant

area, it could be interesting to produce a larger bicluster that would result from merging those biclusters despite introducing some noise to it. Finally, filtering procedures can also be applied, for instance, to remove duplicated and non-maximal biclusters that could be created through previously mentioned post-processing options.

Regarding effectiveness and efficiency, BicPAM has been submitted to extensive experimental evaluations [47, 48]. In these evaluations, BicPAM was compared against peer pattern-based biclustering approaches, as well as state of the art biclustering algorithms in both synthetic and real-world datasets with different levels of size and noise. BicPAM was able to find optimal solutions, even in noisy environments, outperforming the rest of the algorithms in most of the metrics assessed. However, due to the inherent characteristics of the pattern-based algorithms, BicPAM seems to suffer in terms of performance when confronted with large datasets.

Despite the extensiveness of the evaluations made in the previously mentioned works, to our knowledge, the algorithm was never evaluated in recommendation datasets, known for the extreme sparseness of the data. As a result, in this work, we want to validate if BicPAM is the best biclustering algorithm option for recommendation purposes. One effective way to evaluate biclustering algorithms is by using synthetic data. Synthetic data allows us to generate data with planted hidden biclusters, which can be used as ground truth to test the capabilities of the algorithms. In this chapter, we use synthetic data to compare the capabilities of BicPAM to discover constant columns and order-preserving biclustering solutions against the peers previously used the related-work studies.

4.2 Generating Synthetic Biclustering Benchmark Data - G-Tric

To create synthetic data that resembles real-world data, we used a synthetic biclustering and triclustering data generator called G-Tric[1] [75]. G-Tric is a freshly developed improvement of the BiGen[2] software, proposed by Henriques and Madeira [46], which is, to our knowledge, the most comprehensive generator of synthetic biclustering data. G-Tric offers the user the possibility of creating highly particular datasets, planting not only biclusters but also triclusters (3D biclusters) through the setting of a complete parameterization. Bellow, we enumerate the main parameters of G-Tric and explain the process for choosing the values for each one.

- *Data size ($|X| \times |Y|$).* Regarding the size of the synthetic matrix, we use the same row and column proportion that we find in the popular recommendation Netflix dataset. This dataset has around 27 times more rows (users) than columns (movies),

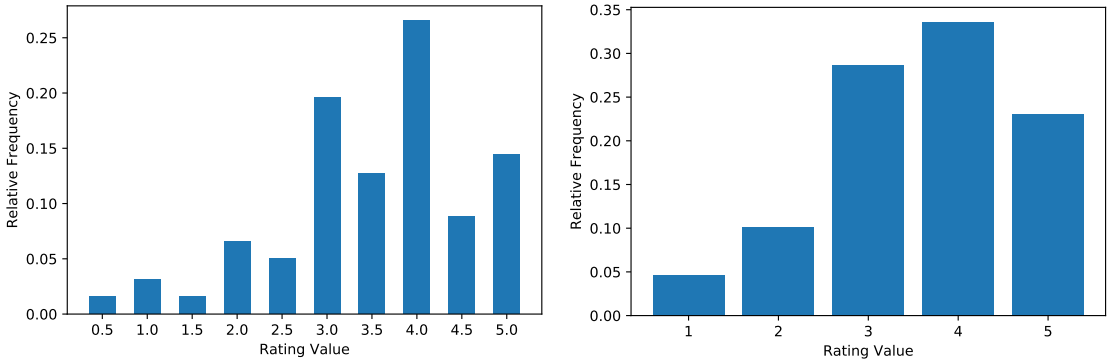
[1] `https://github.com/jplobo1313/G-Tric/`
[2] `http://web.ist.utl.pt/~rmch/software/bigen/`

following the proportion, in our experiment, we considered two matrices with distinct sizes: $1,620 \times 60$ and $8,100 \times 300$.

Our initial idea was also to contemplate a much larger matrix $48,000 \times 1,800$, but that was not possible due to the high computational demands needed to produce such data.

- *Alphabet / Set of values ($\mathcal{L}$).* The values in the data represent ratings from the users from a 1 to 5 rating scale ($\mathcal{L} \in \{1, 2, 3, 4, 5\}$). A float representation with 0.5 increments for each value can also be found in some rating datasets. However, for ease of simplicity and interpretability of the results, we chose to use the integer scale.

- *Distribution of the values.* To determinate realistic probabilities, we examined how many times each rating occurs in our real-world datasets. We found that there are rating values that are more common than others, as Figure 4.3 highlights. Thus, having this in mind, we decided to express the chances of each value of the alphabet to occur as approximate weighted probabilities extrapolated from the real-world datasets. The probabilities chosen were 0.05; 0.10; 0.30; 0.35; 0.20 to the values of 1; 2; 3; 4; 5, respectively.



(a) MovieLens-1M (b) Netflix

Figure 4.3: Distribution of the rating values in the MovieLens-1M and Netflix datasets.

- *Model of the biclusters.* We already identified constant columns and order-preserving biclusters as the most interesting patterns for recommendation purposes. Thus, for each of the data sizes considered, we created one dataset with hidden constant columns biclusters and another with hidden order-preserving biclusters.

- *Number and Size of biclusters.* To find out an appropriate number and size of biclusters to plant in the dataset, we used the BicPAMS (Biclustering based on Pattern Mining Software) [49]. This software makes available pattern-based biclustering

algorithms, including the BicPAM algorithm proposed by the same authors. An exhaustive run on the input data finds all maximal biclusters. Thus, the idea was to use the biclustering solution of BicPAMS to extrapolate the amount biclusters we would plant in the synthetic data, as well as the size of each one.

Due to the problem complexity of the biclustering task, the biclustering algorithms usually do not perform exhaustive searches. However, with BicPAMS, we can set some parameters that inflict the algorithm to perform a near exhaustive search. As suggested by the authors of BicPAMS, to guarantee an adequate exploration of the input dataset, we set the stopping criteria, *minimum number of dissimilar biclusters* to a high value, $1,000,000,000$. We also increased the *number of iterations* of the algorithm to 2, which, after the first iteration, removes highly coherent regions in the dataset that could be preventing the discovery of less-trivial biclusters. There are, however, other parameters, already discussed in Section 4.1, that need to be determined. Table 4.1 compiles the settings established for those parameters and, additionally, it includes brief notes supporting the reasoning behind it. These settings can be viewed as a framework to ensure that BicPAMS performs an exhaustive search on an input matrix.

	Parameter	Value	Note
Covergence	Minimum number of biclusters	1,000,000,000	A high value that does not limit the search, resulting in an exhaustive search.
	Iterations	2	Run two times ensures the discovery of relevant biclusters that could be masked by dense coherent regions.
Mapping	Discretization	None	Rating values are already discrete in the Netflix dataset.
	Normalization	None	Normalization is not required since all the data is on the same scale and lie within an [1, 5] interval.
	Missings handler	Remove	Ignore the missing values of the input data.
Mining	Pattern	Order-preserving	Order-preserving is the most interesting pattern for our problem.
	Orientation	Pattern on rows	The goal is to identify users (rows) with coherent rating pattern, this means, searching for patterns on rows.
	Minimum number of rows bicluster	2	All the maximal biclusters present in the input data, regardless of the number of rows each bicluster has, so parameter set to the minimum value.
	Minimum number of cols bicluster	2	All the maximal biclusters present in the input data, regardless of the number of columns each bicluster has, so parameter set to the minimum value.
Closing	Minimum quality	100%	Consider only non-noisy biclusters.
	Minimum similarity	100%	Consider all the unique biclusters that were mined.

Table 4.1: Framework to ensure exhaustive search from the BicPAM algorithm.

We started by running the BicPAM algorithm on a random $1,620 \times 60$ and scaled five times until we reached the $8,100 \times 300$ subset of the Netflix dataset. We then decided only to consider the biclusters with a $p - value$ less than 1×10^{-3} and analyzed the biclustering solution, including the distribution of the sizes of the biclusters. Tables 4.2 and 4.3 present a summary of the results obtained for those subsets when searching for both constant columns and order-preserving biclusters. Regarding the parameters of G-Tric to handle the size of the planted biclusters, we used the normal distribution option with the average and standard deviation values that we found in our biclustering solutions for the $1,620 \times 60$ and $8,100 \times 300$ matrices.

As we mentioned before, the initial intention was to perform the same analysis on

Number of rows	Number of cols	#biclusters found	#significant biclusters ($p < 0.001$)	Avg. #rows per bicluster ($p < 0.001$)	Avg. #cols per bicluster ($p < 0.001$)
1,620	60	118	8	2.75 ± 1.30	2.88 ± 0.60
3,240	120	914	172	2.27 ± 0.58	3.26 ± 0.79
4,860	180	5,203	1,213	2.41 ± 1.03	3.79 ± 0.86
6,480	240	21,730	5,064	2.50 ± 0.98	4.32 ± 1.06
8,100	300	55,477	17,145	2.74 ± 1.29	4.45 ± 1.14

Table 4.2: Properties of the constant columns biclustering solutions varying the size of the input data.

Number of rows	Number of cols	#biclusters found	#significant biclusters ($p < 0.001$)	Avg. #rows per bicluster ($p < 0.001$)	Avg. #cols per bicluster ($p < 0.001$)
1,620	60	166	44	$4{,}05 \pm 3.86$	2.66 ± 0.64
3,240	120	1,650	950	3.36 ± 3.40	3.11 ± 0.81
4,860	180	10,237	6,009	3.62 ± 5.16	3.58 ± 0.94
6,480	240	47,512	26,657	3.66 ± 5.81	4.05 ± 1.07
8,100	300	135,806	86,921	3.76 ± 5.81	4.24 ± 1.11

Table 4.3: Properties of the order-preserving biclustering solutions varying the size of the input data.

larger datasets such as the $48,000 \times 1,800$, but it would require too many computational resources to perform an exhaustive search on the dataset. As an alternative, we decided to pursue another approach to speculate the properties for the $48,000 \times 1,800$ subset. More details on this attempt can be found in the Figure A.1, and in Tables A.1 and A.2 from the Appendix A, but we resolved to continue the study with only the $1,620 \times 60$ and $8,100 \times 300$ sizes.

- *Overlapping settings.* G-Tric allows the user to specify some features about the overlapping between biclusters, from those we highlight: the percentage of biclusters that overlap in the data; the maximum of biclusters that overlap together in at least one element; and the maximum percentage of biclusters' elements that can be overlapped by another bicluster. For each one of those parameters, we examined the biclustering solutions we previously got from BicPAMS to our datasets and set the values accordingly, as Table 4.4 exhibits.

Data Size	Type of bicluster pattern	% of biclusters overlapping	Maximum number of biclusters overlapping together	Maximum % of bicluster's elements overlapped by other bicluster
1,620×60	Constant columns	62.5%	3	75%
1,620×60	Order-preserving	97.8%	10	75%
8,100×300	Constant columns	99.9%	391	93.3%
8,100×300	Order-preserving	99.9%	2,157	93.3%

Table 4.4: Overlapping statistics from the biclustering solutions of the subsets from Netflix dataset.

- *Missing values.* There are two different parameters to produce the missing values in the data.

    1. Missing values in the background.

       This parameter represents the percentage of missing values in the background of the generated data. The authors of the generator define background as the dataset's elements that do not belong to any of the hidden biclusters. Ideally, we wanted to create synthetic datasets with the same missing value percentage as the Netflix dataset, 98.8%. However, the current version of the generator (G-Tric's Beta version), does not support a parameter to force the number of missing values in the whole dataset. Hence, the overall percentage of missings cannot be controlled as it depends on the area that the planted biclusters occupy. Regardless, we set the percentage of missings in the background to 99% for every generated dataset.

    2. Missing values in biclusters.

       In this study we did not allow the planted biclusters to contain any missing values.

- *Noise.* This field specifies the maximum number of noisy elements in a bicluster. In this task, we used BicPAMS to discover the number and characteristics of the perfect biclusters in the real-world data. Accordingly, for the sake of coherency, we did not allow any deviations from the expected pattern on the planted biclusters in the synthetic data.

4.3 Biclustering Evaluation Methodology

This section presents the evaluation of the biclustering algorithms using the synthetic benchmark data. We start by describing the biclustering algorithms, as well as the datasets and metrics used in the evaluation. Then, we introduce and discuss the results. Finally, we conclude the evaluation, justifying our chosen biclustering algorithm based on the obtained results.

4.3.1 Biclustering Methods

BiMax [9], xMOTIFs [81], and QUBIC [69] are three state-of-the-art biclustering approaches that were already adopted in the literature to find biclusters for recommendation purposes [101, 105, 118]. In this experiment, we use our synthetic data to simulate a recommendation dataset environment and evaluate the xMOTIFs, QUIBIC, and BicPAM to retrieve the hidden biclusters. BiMax was discarded because it only deals with binary data.

xMOTIFs is a nondeterministic greedy biclustering algorithm that searches for what the authors call xMOTIFs - conserved gene expression motifs. In the gene expression context, the algorithm searches for a subset of conserved genes across a subset of samples. In a more general context, xMOTIFs can be seen as biclusters. Therefore, the xMOTIFs algorithm can find biclusters with rows that have the same value over the columns - constant columns pattern. The procedure to search for the largest xMOTIFs can be described as follows. It selects n_s columns at random from all the samples to act as seeds. For each one of the selected columns, it creates n_d sets of s_d samples at random. These sets are candidates for the discriminating set. For each seed-discriminating set pair, the corresponding xMOTIF is computed as in [81]. The xMOTIF is discarded if it has fewer columns than the minimum α-fraction of the dataset columns defined. Finally, it returns the largest xMOTIF [61]. Note that this procedure is run as many times as the number of biclusters, $num\_biclusters$, desired in the output solution. In our experiments, we set the number of seeds $n_s = 10$ and the number of determinants $n_d = 100$. The authors of xMOTIFs, in their original work, use $n_d = 1,000$ as the number of determinants, but since we want to retrieve a high volume of biclusters, using $n_d = 1000$ would be computationally impracticable. The size s_d of the discriminating set was varied from 2 to 10, but the best results were achieved with $s_d = 2$. Since we want biclusters with a minimum of two columns, we chose $\alpha = 0.02$ for the $1,620 \times 60$ and $\alpha = 0.005$ for the $8,100 \times 300$ datasets. Finally, we adjusted the number of biclusters retrieved according to the number of biclusters hidden in each of the input datasets.

QUBIC, on the other hand, is a greedy deterministic algorithm. It represents the data as a bipartite graph and treats the biclustering problem as finding heavy sub-graphs in the data. Similarly to BicPAM and xMOTIFs, this algorithm finds constant columns biclusters. One essential feature of the QUBIC algorithm is that, according to the authors, it can identify all statistically significant biclusters. Moreover, it solves the biclustering task very efficiently [69]. QUBIC offers some flexibility by allowing the user to adjust the value of some parameters. The parameter *o* affects the number of biclusters in the output. Similar to what we did with the xMOTIFs algorithm, we established its value according to the input dataset's hidden biclusters. Regarding the parameter *c* that determines the required consistency level of the bicluster, since we are interested in non-noisy biclusters, we set $c = 1$. Besides, we ensure that the retrieved biclusters are maximal, setting the maximal overlap parameter, $f = 0.99$. Finally, the size of the biclusters is controlled by the parameter *k*, and we set to $k = 2$ so that the biclusters have at least two columns.

We already introduced BicPAM and its parameterization in Section 4.1. The settings used in the experiment were the same used in Table 4.1, except for the *minimum number of biclusters* and *patterns* that were set according to the number and pattern of biclusters we were looking for in each dataset. The *iterations* parameter was also decreased to 1 since we believe the possible benefits of running the algorithm two times do not justify

the downgrade of performance.

To run the xMOTIFs and the QUBIC algorithm, we used the implementations available in a Python library of biclustering algorithms, biclustlib[3] [84]. This library provides an xMOTIFs implementation that follows precisely the pseudo-code described in the original paper. As for the QUBIC, the library wraps an executable of the QUBIC original code [4] implemented in ANSI C by its developers.

4.3.2 Synthetic Data

To generate our synthetic benchmarks datasets, we used the previously described G-Tric generator using the settings detailed in Section 4.2. We produced four datasets that we named alphabetically from "Synthetic-A" to "Synthetic-D". The main characteristics of the generated datasets are displayed in Table 4.5. It is crucial to notice that the current version of the generator (G-Tric's Beta version) cannot fully replicate all the features intended to have in our datasets. For example, regarding the Synthetic-D, the generator could not plant all the 86,921 biclusters. Moreover, since G-Tric parameterization does not allow the user to control the number of missing values in the data, we noticed that the sparseness of our datasets was compromised, especially in Synthetic-D, where the density reaches 63.1%.

Dataset	Alfabet ($\mathcal{L}$)	Data Size	Number of hidden bics	Pattern of the biclusters	Density	Area of biclusters
Synthetic-A	{1, 2, 3, 4, 5}	1,620 × 60	8	CC	0.013	0.07%
Synthetic-B	{1, 2, 3, 4, 5}	1,620 × 60	44	OP	0.016	0.63%
Synthetic-C	{1, 2, 3, 4, 5}	8,100 × 300	17,145	CC	0.091	8.18%
Synthetic-D	{1, 2, 3, 4, 5}	8,100 × 300	66,341	OP	0.631	62.79%

Table 4.5: Characteristics of the generated synthetic datasets.

4.3.3 Performance Metrics

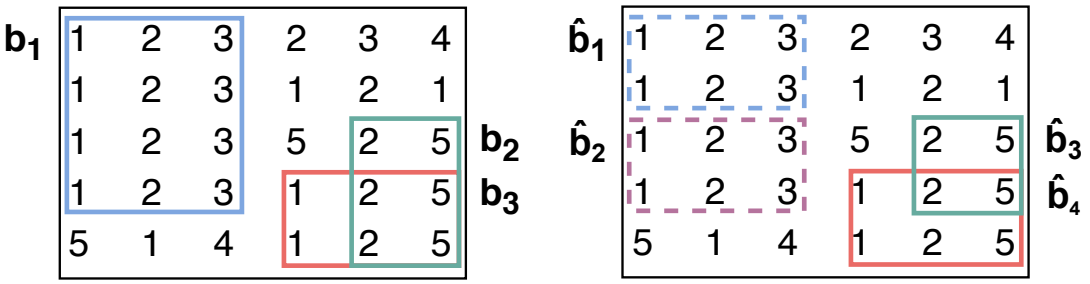
We evaluated the solutions of each algorithm through an external evaluation methodology, using similarity measures to compare the found solutions against a ground truth (the hidden biclusters in each of the synthetic datasets). During the past years, several comparative biclustering studies addressed external evaluation using different similarity measures [13, 30, 84, 89]. Prelić et al. [89] proposed two clustering comparison match scores, *Prelić relevance (S_{prel})* and *Prelić recovery (S_{prec})*, that have been vastly adopted. Another popular measure is the *relative non-intersecting area (S_{rnia})*, proposed by Patrikainen and Meila [86], and its extension *Clustering Error (S_{ce})*. Horta and Campello, in a recent work

[3] `https://github.com/padilha/biclustlib`
[4] `https://github.com/maqin2001/qubic`

[53], reviewed sixteen of the most popular biclustering measures and warned that some of them could easily lead to misleading evaluations. In this work, the authors perform a theoretical comparison of the sixteen similarity measures and conclude highlighting two of them, the *Clustering Error (S_{ce}))* [86] and the *Campello Soft Index (S_{csi})* [17]. In terms of eight properties relevant for evaluating biclustering solutions, these measures, were recognized as being the top similarity measures, and therefore, the ones recommended by the authors for comparing biclustering solutions. In this experiment, we adopt the measures recommended by Horta and Campello to evaluate biclustering solutions. But we also use the S_{rnia} that despite being theoretically inferior to S_{ce}, it still considered as a top measure. Moreover, we complement our evaluation with the measures proposed by Prelić because the S_{prel} and S_{prec} measures evaluate specific properties, allowing its results can be easily interpreted.

Both *Prelić relevance (S_{prel})* [89] and *Prelić recovery (S_{prec})* [89] scores are based on Jaccard index. The S_{prel} is a measure that reflects how well the predicted biclusters represent the reference biclusters in both dimensions (completeness). Whereas, the S_{prec} defines to what extent each of the reference is recovered by the predicted biclustering (precision).

Assuming that $B = \{b_i\}_{i=1}^{k}$ and $\hat{B} = \{b_i\}_{i=1}^{q}$ are, respectively, the reference and the found biclustering solutions in an input matrix $A \in \mathbb{R}^{n \times p}$.



(a) Reference biclustering solution B.

(b) Found biclustering solution $\hat{B}$.

Figure 4.4: Example of two biclustering solutions that we wish to compare using biclustering similarity measures.

Let $S_r(B, \hat{B})$ (Equation 4.1) be the match score for the rows, and $S_c(B, \hat{B})$ an equivalent score for the columns dimension. The overall S_{prel} and S_{prec} are defined by Equation 4.2 and Equation 4.3, respectively.

$$S_r(B, \hat{B}) = \frac{1}{k} \sum_{k}^{i=1} \max_{l \in \mathbb{N}_{1,q}} \left\{ \frac{B_i^r \cap \hat{B}_l^r}{B_i^r \cup \hat{B}_l^r} \right\} \tag{4.1}$$

$$S_{prel}(B, \hat{B}) = \sqrt{S_r(B, \hat{B}) \times S_c(B, \hat{B})} \tag{4.2}$$

$$S_{prec}(B, \hat{B}) = S_{prel}(\hat{B}, B) \tag{4.3}$$

Considering Figure 4.4 as an example. We have $k = 3$, and $q = 4$, which are the number of biclusters in the biclustering solutions. To calculate $S_r(B, \hat{B})$, we need to determine, for each of the biclusters in B, the maximum Jaccard Index with a bicluster in $\hat{B}$. The maximum Jaccard Indexes are $J(b_1, \hat{b_1}) = 0.5$, $J(b_2, \hat{b_3}) = 0.667$, and $J(b_3, \hat{b_4}) = 1$. So, following Equation 4.1, $S_r(B, \hat{B}) = \frac{1}{3} \times (0.5 + 0.667 + 1) \approx 0.72$. Using the same procedure for the columns dimension, we get $S_c(B, \hat{B}) = \frac{1}{3} \times (1 + 1 + 1) \approx 1$. Finally, the overall $S_{prel}(B, \hat{B}) = \sqrt{0.72 \times 1} \approx 0.85$. If we follow the same reasoning for S_{prec}, we obtain $S_{prec}(B, \hat{B}) = \sqrt{0.625 \times 1} \approx 0.79$.

The *Relative non-intersecting Area* (Equation 4.6) [86] and *Clustering Error* (Equation 4.7) [86] are very similar measures. To take the overlapping between biclusters into account, both these measures use a redefined definition of union and intersection sets of biclusters. Let $N_{j1,j2}$ and $\hat{N}_{j1,j2}$ be the number of biclusters belonging to B and $\hat{B}$, respectively, at the i-th row and j-th column entry of the input matrix, A. The size of the redefined union set, is the sum of the maximum number of biclusters in each entry of the input matrix between the reference and found biclustering solutions (Equation 4.4). Whereas the size of the intersection is the sum of the minimum number of biclusters in each entry (Equation 4.5). The *Clustering Error* differs from the *Relative non-intersecting Area* in a sense that it considers the number of elements shared by biclusters. It constructs a matrix M, as in Equation 4.8, with the elements shared by each bicluster $b_i \in B$ and $\hat{b}_i \in \hat{B}$, and aims to find a permutation of biclusters such that the sum of the diagonal elements of M, d_{max}, is maximized. For example, in the case of the two biclustering solutions in Figure 4.4 we have $|U| = 24$ and $|I| = 22$. The maximal permutation of biclusters in Equation 4.8 would be $\{b_1, \hat{b}_1\}, \{b_2, \hat{b}_3\}, \{b_1, \hat{b}_4\}\}$, deriving $d_{max} = 6 + 4 + 6 = 16$. Thus $S_{rnia}(B, \hat{B}) = \frac{22}{24} \approx 0.92$, and $S_{ce}(B, \hat{B}) = \frac{16}{24} \approx 0.67$.

$$|U| = \sum_{j_1, j_2} \max\{N_{j_1,j_2}, \hat{N}_{j_1,j_2}\} \tag{4.4}$$

$$|I| = \sum_{j_1, j_2} \min\{N_{j_1,j_2}, \hat{N}_{j_1,j_2}\} \tag{4.5}$$

$$S_{rnia}(B, \hat{B}) = 1 - \frac{|U| - |I|}{|U|} = \frac{|I|}{|U|} \tag{4.6}$$

$$S_{ce}(B, \hat{B}) = 1 - \frac{|U| - d_{max}}{|U|} = \frac{d_{max}}{|U|} \tag{4.7}$$

	$\hat{b}_1$	$\hat{b}_2$	$\hat{b}_3$	$\hat{b}_4$
b_1	6	6	0	0
b_2	0	0	4	2
b_3	0	0	2	6

(4.8)

The *Campello Soft Index* [53] uses the elements of the input matrix to represent the biclustering solution. The input matrix A is viewed as a set of objects $O = \{\tilde{o}_1, \tilde{o}_2, ..., \tilde{o}_{n,p}\}$,

where $\tilde{o}_k$ represents an element $A_{i,j}$. Therefore, a bicluster can be defined as a set of the objects, $\dot{p}$, that compose it. Horta and Campello use this representation to transform any biclustering solution $\dot{B}$ into an augmented set, given by Equation 4.9. Each of the biclusters in $\dot{B}$ are translated into a cluster $\dot{p}$, and the elements in A that do not belong to any bicluster are singletons in this $\dot{P}$.

$$\dot{P} = \{\dot{p}_1, \dot{p}_2, ..., \dot{p}_k\} \tag{4.9}$$

Using Figure 4.4 as an example, applying the object representation and Equation 4.9, we have $p_1 = \{\tilde{o}_1, \tilde{o}_2, \tilde{o}_3, \tilde{o}_4, \tilde{o}_6, \tilde{o}_7, \tilde{o}_8, \tilde{o}_9, \tilde{o}_{11}, \tilde{o}_{12}, \tilde{o}_{13}, \tilde{o}_{14}\}$, $p_2 = \{\tilde{o}_{23}, \tilde{o}_{24}, \tilde{o}_{25}, \tilde{o}_{28}, \tilde{o}_{29}, \tilde{o}_{30}\}$, $p_3 = \{\tilde{o}_{19}, \tilde{o}_{20}, \tilde{o}_{24}, \tilde{o}_{25}, \tilde{o}_{29}, \tilde{o}_{30}\}$, $p_4 = \{\tilde{o}_5\}$, $p_5 = \{\tilde{o}_{10}\}$, $p_6 = \{\tilde{o}_{15}\}$, $p_7 = \{\tilde{o}_{16}\}$, $p_8 = \{\tilde{o}_{17}\}$, $p_9 = \{\tilde{o}_{18}\}$, $p_{10} = \{\tilde{o}_{21}\}$, $p_{11} = \{\tilde{o}_{22}\}$, $p_{12} = \{\tilde{o}_{26}\}$, $p_{13} = \{\tilde{o}_{27}\}$, and $P = \{p_1, p_2, ..., p_{13}\}$ for the biclustering solution B. Proceeding likewise for the solution $\hat{B}$, we get $\hat{P} = \{\hat{p}_1, \hat{p}_2, ..., \hat{p}_{14}\}$. After transforming both biclustering solutions, the *CSI* measure is calculated comparing the sum of agreements, $a_G^{P,\hat{P}}$, and disagreements,$d_G^{P,\hat{P}}$, between the objects from both solutions regarding biclusters, as in Equation 4.10. More information on how the agreements and disagreements are calculated can be found in the author's supplementary material[5], available online. Taking the solutions in Figure 4.4 as an example, we have $a_G^{P,\hat{P}} = 109$ and $d_G^{P,\hat{P}} = 103$, resulting in $S_{csi} = \frac{109}{212} \approx 0.51$.

$$S_{csi}(P, \hat{P}) = \frac{a_G^{P,\hat{P}}}{a_G^{P,\hat{P}} + d_G^{P,\hat{P}}} \tag{4.10}$$

Besides the external evaluation, we believe that the algorithm's efficiency should also be taken into account since, in a recommendation scenario, some applications may have time requirements in the presence of large volumes of data. Thus, in this experiment, we measure the time the algorithms took to return the biclustering solution. Furthermore, we also measured how much memory was allocated during the execution of the algorithms. Note, however, that the algorithms' time and space efficiency are highly dependent on many factors, including its implementations. So, this type of experimental measurement can only be viewed as a superficial evaluation of the implementations we used, instead of a precise evaluation of the algorithms.

4.3.4 Results and Discussion

Below, we introduce the results of our analysis to assess the three biclustering algorithms' ability to recover hidden biclusters. We start by presenting the results of each algorithm for each one of the synthetic datasets considering the performance metrics described in Section 4.3.3. Then, we discuss the results and exhibit a summary of the characteristics of the produced solutions. The experiments were performed using the biclustlib[6] Python

[5] `https://horta.github.io/biclustering/paper/appendix.pdf`
[6] `https://github.com/padilha/biclustlib`

library on a machine with Intel Xeon 2.40GHz having thirty-two cores and 32GBs of RAM.

Tables 4.6, 4.7, and 4.8 show the performance results of each algorithm. Due to computational resources constraints, we were not able to calculate the S_{ce} and the S_{csi} scores for some of the datasets.

QUBIC was able to produce interesting solutions, especially for the Synthetic-A and Synthetic-D datasets. The solution for the Synthetic-A dataset had the best scores for the S_{prel}, S_{rnia}, S_{ce} similarities, being surpassed only by BicPAM in the S_{prec} and S_{csi}. QUBIC also achieved the best S_{prel}, and the best S_{prel}, S_{rnia}, S_{csi} for the Synthetic-C and Synthetic-D, respectively. In terms of time and memory efficiency, QUBIC was not the best in neither, but it was decent overall.

Dataset	Time (s)	Max Memory (MB)	S_{prel}	S_{prec}	S_{rnia}	S_{ce}	S_{csi}
Synthetic-A	1.561	531.570	**0.9798**	0.6124	**0.6892**	**0.6892**	0.0005
Synthetic-B	1.587	537.152	0.5344	0.4108	0.2642	0.1812	0.0572
Synthetic-C	35.479	833.086	**0.3575**	0.1021	0.0186	0.0042	0.0057
Synthetic-D	**477.380**	3,223.422	0.2245	**0.1539**	**0.1555**	-	**0.1017**

Table 4.6: Similarity scores and efficiency of the QUBIC algorithm.

The xMOTIFs, under the settings described in Section 4.3.1, was not able to generate satisfactory solutions, compared with QUBIC and BicPAM. The peers outperformed it in all of the scores except for S_{prel} in Synthetic-D's solution. xMOTIFs, despite being the worst in computation speed, it had undoubtedly the best results regarding memory used for all solutions.

Dataset	Time (s)	Max Memory (MB)	S_{prel}	S_{prec}	S_{rnia}	S_{ce}	S_{csi}
Synthetic-A	198.162	**448.711**	0.3227	0.2141	0.0588	0.0588	< 0.0001
Synthetic-B	1,241.306	**448.719**	0.4390	0.3513	0.1676	0.1332	< 0.0001
Synthetic-C	87,453.563	**454.008**	0.3439	0.1753	0.0570	0.0083	< 0.0001
Synthetic-D	1,037.261	**449.664**	**0.2453**	0.0492	0.0088	-	< 0.0001

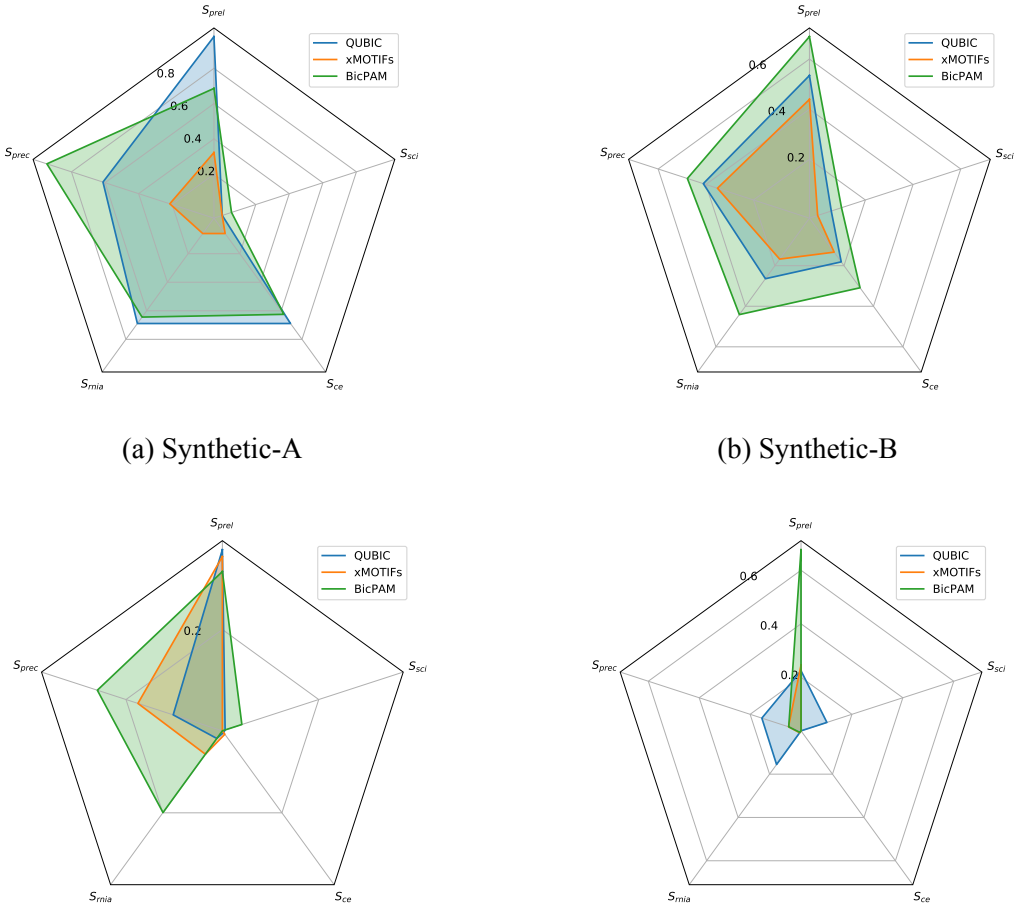
Table 4.7: Similarity scores and efficiency of the xMOTIFs algorithm.

On the other hand, BicPAM obtained the best similarity results on most of the solutions, particularly in the Synthetic-B and Synthetic-C datasets. The results for the Synthetic-A, were only slightly surpassed by the QUBIC's results in some similarities. In terms of time and memory efficiency, BicPAM was the fastest to create a solution for Synthetic-A, B, and C.

Dataset	Time (s)	Max Memory (MB)	S_{prel}	S_{prec}	S_{rnia}	S_{ce}	S_{csi}
Synthetic-A	**0.236**	512.000	0.6862	**0.9464**	0.6442	0.6250	**0.0557**
Synthetic-B	**0.394**	512.000	**0.6896**	**0.4774**	**0.4411**	**0.3089**	**0.1000**
Synthetic-C	**32.633**	1,556.000	0.3143	**0.2595**	**0.1994**	-	**0.0405**
Synthetic-D	10,899.418	29,012.000	0.0677	0.0483	0.0072	-	-

Table 4.8: Similarity scores and efficiency of the BicPAM algorithm.

A more evident visualization can be found in Figure 4.5, where we can easily compare the performances from each algorithm in each of the datasets through radar charts.



(a) Synthetic-A

(b) Synthetic-B

(c) Synthetic-C

(d) Synthetic-D

Figure 4.5: Performance of the biclustering algorithms in each synthetic dataset.

Regarding the similarity metrics, from a generalist point of view, BicPAM was the best among the biclustering techniques evaluated. This superiority was mainly reflected in Synthetic-B and Synthetic-D, where none of the peers was able to match the BicPAM's solutions. The poor performances in those datasets were already expected since the hidden biclusters in both of these datasets followed an *order-preserving* pattern that neither QUBIC nor xMOTIFs can identify. In Synthetic-A and Synthetic-D, QUBIC's solutions

contested the ones from BicPAM, primarily if we focus on the S_{prel}, S_{rnia} and S_{ce}. These results indicate that QUBIC may produce solutions with more correct/precise biclusters (higher S_{prec} value), but the ones from BicPAM are more complete (higher S_{prel} value). We believe that a possible explanation for the completeness of the BicPAM solutions is the exhaustive nature of the algorithm. When analyzing the S_{csi} values, which evaluates many aspects of the solutions, BicPAM was superior. The fact that this measure takes into account many features of the solution, evaluating it in a generalist way, makes it challenging to identify the exact properties that made the BicPAM's solutions better than the remaining ones. Hence, a possible future improvement to the biclustering field could be the creation and adoption of new similarities that focus on the measurement of more specific solution's features.

As for the efficiency of biclustering algorithms, there is usually a trade-off between the quality of the solution and the time it takes to be produced, caused by the complexity of the biclustering task. When it comes to pure speed to produce the results, the size of the dataset seems to influence all of the algorithms. As they took more time in Synthetic-C and Synthetic-D. If we look at the number of biclusters returned by each algorithm in Tables 4.9, 4.10, and 4.11, it seems there is a direct relation between the number of biclusters and the time the algorithm needs to compute. BicPAM was the fastest technique for three of the four synthetic datasets, but QUBIC took considerably less time to produce the Synthetic-D's solution.

Dataset	#biclusters found	Avg. #rows per bicluster	Avg. #cols per bicluster
Synthetic-A	5	3.40 ± 1.02	3.00 ± 0.00
Synthetic-B	30	2.07 ± 0.25	3.23 ± 0.50
Synthetic-C	147	2.00 ± 0.00	16.92 ± 2.18
Synthetic-D	6,830	3.95 ± 0.25	17.33 ± 2.51

Table 4.9: General information of the QUBIC's solutions.

Dataset	#biclusters found	Avg. #rows per bicluster	Avg. #cols per bicluster
Synthetic-A	4	2.00 ± 0.00	2.00 ± 0.00
Synthetic-B	23	2.87 ± 0.90	2.09 ± 0.28
Synthetic-C	810	9.29 ± 12.15	2.06 ± 0.30
Synthetic-D	40	202.5 ± 408.11	3.08 ± 2.63

Table 4.10: General information of the xMOTIFs' solutions.

Dataset	#biclusters found	Avg. #rows per bicluster	Avg. #cols per bicluster
Synthetic-A	14	3.50 ± 1,00	2.62 ± 0.48
Synthetic-B	47	7.43 ± 1.62	2.02 ± 0.14
Synthetic-C	26,347	19.60 ± 9.84	2.01 ± 0.09
Synthetic-D	57,336	2,015.14 ± 798.39	2.43 ± 0.50

Table 4.11: General information of the BicPAM's solutions.

Another aspect to note is that even though we configured all algorithms to return the same number of biclusters, neither QUBIC nor xMOTIFs retrieved as many biclusters, in any of the solutions, as BicPAM. This behaviour was already anticipated for the Synthetic-B and Synthetic-D, since BicPAM is the only algorithm able to identify *order-preserving* biclusters. However, for Synthetic-A and Synthetic-C, we believe that the techniques' intrinsic features are the main cause of these results. For instance, xMOTIFs' non-deterministic nature led its solutions to include many empty or non-valid biclusters, with less than 2 rows or columns.

Focusing on the average size of the found biclusters, it is clear that QUBIC tends to discover bicluster with more columns than rows. On the other hand, xMOTIFs and BicPAM tend to discover biclusters with more rows than columns.

Concerning memory consumption, the performance of QUBIC and BicPAM degraded as the number of biclusters in the datasets increased, whereas xMOTIFs was not affected by the characteristics of the input dataset, which is in agreement with other studies [30, 84].

4.4 Final Remarks

This chapter evaluated the potentialities of the biclustering algorithm BicPAM to discover biclusters in classic U-I interaction matrices of the Collaborative Filtering scenario. Using synthetic data resembling real-world U-I matrix data, we evaluated BicPAM and two other biclustering algorithms (QUBIC and xMOTIFs).

From the results discussed in the Section 4.3, we conclude that QUBIC and BicPAM are both able to produce biclustering solutions with better quality than xMOTIFs. Regarding the quality of the bicluster solutions, it is very difficult to determine which biclustering algorithm is the best for the evaluated task. QUBIC was able to produce more correct biclustering solutions, whereas BicPAM created more complete solutions. However, from a generalist perspective, considering all the evaluation metrics, BicPAM seems to create superior biclustering solutions.

As for the characteristics of the biclusters found, there is a clear difference between QUBIC and BicPAM. QUBIC has a clear tendency to discover biclusters with more columns than rows, while BicPAM has the opposite tendency.

Despite our evaluation showing that both QUBIC and BicPAM could be viable options to include in a biclustering-based CF approach, we decided to continue this work adopting QUBIC as our default biclustering algorithm. The reasoning behind this decision is not based on BicPAM's ability to recover the hidden biclusters, but rather the nature of those biclusters. For the specific approach that we propose in Chapter 5, the biclusters should include as many items as possible, instead of including many users, as it is the natural nature of BicPAM's discovered biclusters. Further details about this decision can be found in Chapter 5.

Chapter 5

USBCF - A User-Specific Bicluster-based Approach for Collaborative Filtering

This chapter proposes a novel Collaborative Filtering approach, which uses biclustering to create user-specific U-I matrices (USBCF). We start by motivating our approach and explaining the main idea behind it. Then, we focus individually on the approach's main components, highlighting the design and implementation decisions. Finally, we conclude by reviewing the proposed approach, discussing its strengths and limitations, and introducing research avenues to further improve the USBCF methodology.

5.1 Motivation and Main Concept

Most Collaborative Filtering systems, especially memory-based such as user-based and item-based, have issues when dealing with the nature of today's feedback data. Those systems operate in environments with a large number of items, which leads to huge sparsity in the feedback data, affecting the systems' scalability and accuracy. Clustering techniques, including biclustering, can be an excellent option to tackle both of these issues. USBCF is a new Collaborative Filtering approach that uses the biclustering technique to reduce the entire U-I matrix into much smaller, user-specific U-I matrices. The concept of creating user-specific U-I matrices using biclustering was already adopted in another approach (BBCF) [101]. However, the BBCF method has a minimal coverage capability for producing predictions/recommendations, which means it can produce noticeable recommendations but only for a small subset of users and items. USBCF tries to overcome this significant limitation while maintaining reliable performance results.

Our methodology consists of three main steps. We start by discovering biclusters in the data. Then, using the biclusters, we create a smaller and denser dataset personalized for each user. Once the personalized dataset is created, a CF algorithm is used to originate a unique CF model for each system user.

5.2 USBCF Overview

In this section, we detail each of the three main steps composing the USBCF approach. Besides discussing our solution's implementation technicalities, we also mention other alternatives that could be viable for implementation. Figure 5.1 presents and overview of the USBCF methodology when applied to a simple U-I matrix to create a personalized CF model for the user u_6.

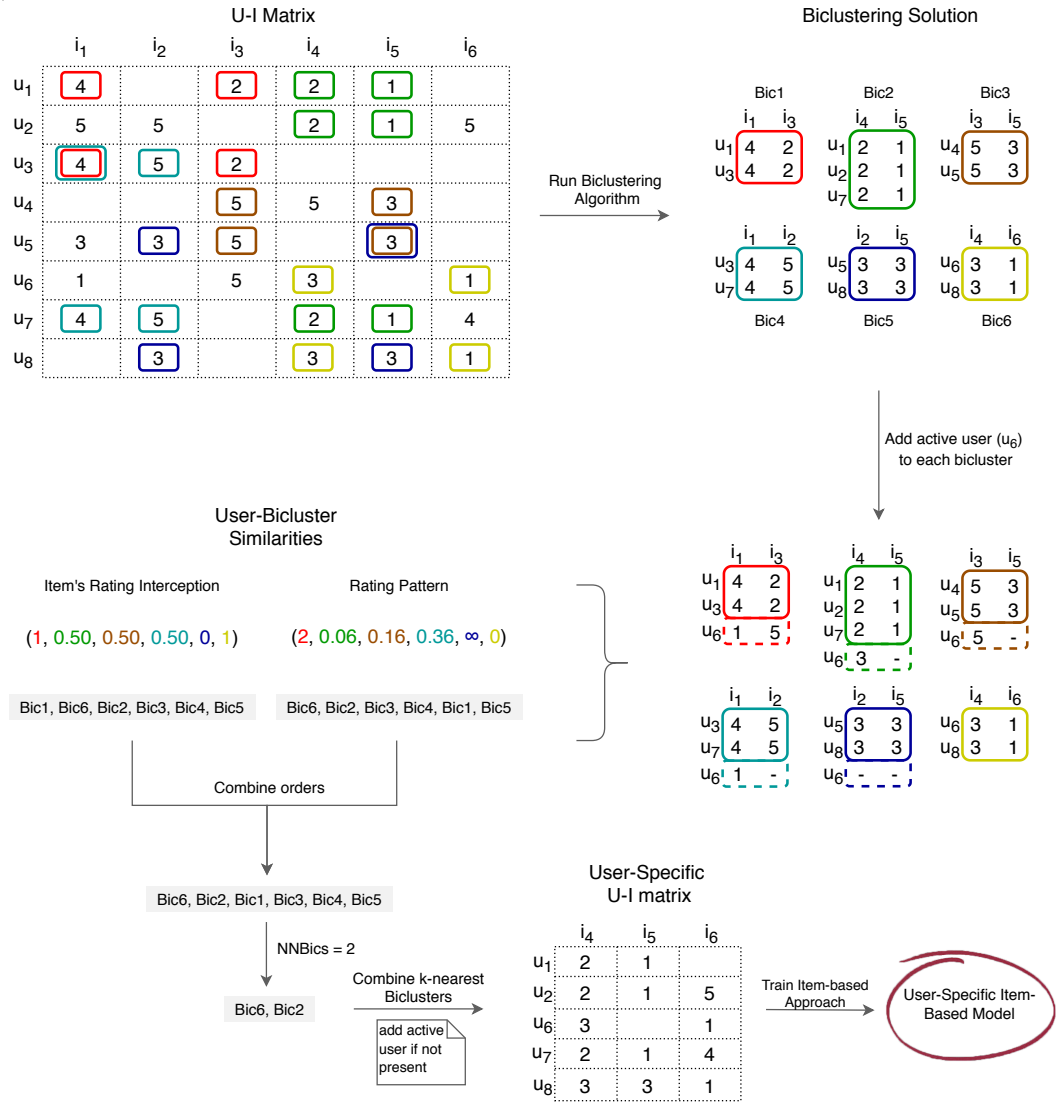


Figure 5.1: USBCF approach applied on a small U-I matrix.

5.2.1 Discover Biclusters in U-I Matrix

Biclustering Algorithm

The first step of the approach consists of running a biclustering algorithm to the U-I dataset to discover biclusters. The goal is to obtain a biclustering solution that highlights the

maximum amount of correlations in the dataset. It is also essential that it contains most of the original dataset elements because we need information from most users and items in the system. To decide which biclustering algorithm we would choose to include in our approach, we conducted the study in Chapter 4, in which we concluded that both QUBIC and BicPAM are viable options. During the development of the approach, we decided to use QUBIC due to its nature of creating biclusters with more columns than rows, contrarily to BicPAM that tends to generate biclusters with more rows than columns. This natural behavior of QUBIC is particularly essential because, during the creation of a user-specific dataset, we need the dataset to contain most of the items present in the system. A Collaborative Filtering model trained in only a subset of the items would necessarily lead to a lack of coverage in the predictions since it would not even recognize that some items existed in the system.

QUBIC Settings

QUBIC algorithm is a greedy deterministic algorithm that finds constant-values on columns biclusters. In our approach, we considered only non-noisy biclusters, setting the parameter $c = 1$ and controlling the maximal overlap with $f = 0.99$, making sure the biclusters are maximal. However, it could be interesting to understand how different algorithm setups can impact the approach's performance. A detailed description of the algorithm and its parameterization can be found in Section 4.3.1 and in QUBIC's original work [69].

5.2.2 Create User-Specific Bicluster-based Matrix

The goal of this second step is to create, for each user, a new data-region with meaningful and personalized information. To do so, we take advantage of the biclusters found by the biclustering algorithm in the previous module. After the biclusters are generated, we can expect that some of those biclusters represent a user's preferences better than others. From this perspective, we try to find a "neighborhood" of biclusters, in other words, a subset of biclusters that individually are "similar" to a specific user. Then, we aggregate the neighborhood biclusters and create a user-specific dataset. Singh and Mehrotra also used the concept of similarity between a bicluster and a user in their methodology. However, their similarity is solely based on the rated items' interception. We propose a different user-bicluster similarity that, besides the rated items' interception, also considers the value of those ratings. Bellow, we explain the details of our new similarity.

User-Bicluster "Similarity"

To find the k-nearest biclusters to the active user, USBCF does not use a similarity in its "true" concept. Instead, we obtained the k-nearest biclusters by combining two distinct

measures. We refer to these measures as **rated items' interception** ($measure_{intercept}$) and **rating pattern** ($measure_{pattern}$).

In the previous related-work approaches [101, 105], some authors highlighted how a viable similarity between a user and a bicluster could be obtained by the interception between the items the user rated and the items of the bicluster, as indicated in Section 3.1. Similarly to these previous works, we also consider a version of this type of similarity, but, for our approach, we do not weight the number of users of the bicluster, as we do not need to favor the biclusters that have more users. Our rated items' measure is given by:

$$measure_{intercept}(u, b) = \frac{|I_u \cap I_b|}{|I_b|}. \tag{5.1}$$

Its values range between [0,1], with 1 meaning that the user and the bicluster are very similar. Nevertheless, we argue that a similarity entirely based on rated items' can be enhanced since it completely disregards those ratings' values. For instance, in Figure 5.2, we have an example of bicluster and an active user that rated all the items in the bicluster. If we stick to the measure in Equation 5.1, we will obtain a similarity of value 1, which indicates a maximum similarity. Notice, however, that despite the active user rated all bicluster items, the values of those ratings follow a completely different rating pattern. Thus, we propose an augmentation of the procedure previously adopted by other authors.

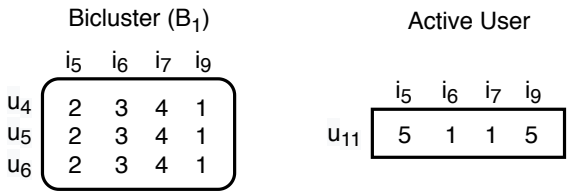


Figure 5.2: Example of a bicluster and an active user with distinct rating pattern.

We believe that if a user is "similar" to a bicluster, then the user should belong to that bicluster. This comes directly from the definition of bicluster since, for a group of users to be in the same bicluster, there has to be a correlation in their rating pattern. Exploring this idea, we decided to create a new "similarity", which measures how a bicluster's quality is affected if we include the active user in it. We calculate this new measure as:

$$measure_{pattern}(u, b) = residue_{b+u} - residue_b, \tag{5.2}$$

where $residue_b$ is the amount of bicluster's residue, and $residue_{b+u}$ is the residue of the bicluster with the row of the active user in it. By subtracting the residue of the final bicluster (with the user) for the initial bicluster (without the user), we measure the amount of residue that the user is responsible for producing. The range of values the $measure_{pattern}$ takes depends on the quality measure adopted. However, a higher value means a more significant impact from the user on the quality, and thus a lower "similarity"

between user and bicluster. Note that in some cases, the active user already belongs to to the bicluster. In those situations the value of the rating pattern measure is zero, since the bicluster is the same before and after introducing the user.

To evaluate the quality of the biclusters, we considered the *Mean Squares Residue* (MSR). The MSR, popularized by Cheng and Church [18], is a robust measure to evaluate the coherence of bicluster's rows and columns. MSR is capable of detecting shifting tendencies within the bicluster [88]. Besides, biclusters with constant-values on the columns obtain maximum quality in this measure.

Nevertheless, MSR is not defined for biclusters with missing values. Even if we do not allow missing values when discovering the biclusters, we may introduce missing values when we include the active user since he may not have rated all of the items. We considered four different options to deal with the missing values in the biclusters:

1. Replace the missing values with a constant value. It is a simple solution, but we would be introducing residue in the biclusters. Since the rated items' interception already "penalizes" biclusters with items the user did not rate, introducing residue with the missing values, we would be "penalizing" the same biclusters twice.

2. Replace the missing values with mean, mode, or median of the bicluster's row, column, or elements. Simply replacing missing values with an average may not reflect the real trends of the bicluster.

3. Adapt MSR to ignore missing values when computing the residue. We can make sure only the remaining elements contribute to the residue estimation by ignoring the missing values, reducing the missing values' impact.

4. Implement a new quality measure well defined for biclusters with missing values. The QUBIC only produces biclusters with constant values on columns. Thus, we can create a quality measure specific to this type of pattern.

We decided to pursue with only the third and fourth option, since the first and second option would penalize biclusters with missing values. We adapted the MSR measure so that it does not take into account the missing entries when computing the residue. We refer to this new MSR as MSR-missing-adaptation. Algorithm 2 shows the implementation of this new adapted measure.

Algorithm 1: Determine MSR adapted to biclusters with missing values.

Data: A bicluster in the form of a matrix.
Result: The MSR-missing-adaptation of the matrix.

```
/* means without missing values */
rows_means ← rows_nan_means(matrix)
cols_means ← columns_nan_means(matrix)
matrix_mean ← matrix_nan_mean(matrix)
/* count non missing values */
count ← 0
/* residue accumulator */
residue ← 0

/* iterate elements of the bicluster */
for row in matrix do
    for col in row do
        elem ← matrix[row,col]
        /* only non missing elements contribute */
        if elem not nan then
            elem_res = elem - rows_means[row] - cols_means[col] +
              matrix_mean
            residue ← residue + pow((elem - elem_res), 2)
            count ← count + 1
    end
end

return 1/count * residue
```

We also created a new quality measure, the *Column-Mean-based Residue* (CMR). The proposed biclustering solution only contains biclusters with constant-values on columns, so a simple residue based on the difference between the elements and the means of the columns can be used if it handles the missing values in biclusters. The CMR quality measure is very similar to the MSR-missing-adaptation. However, since it is was implemented specifically for a unique type of biclusters, we believe it can produce more accurate results specifically for our approach. The pseudocode for CMR is available in Algorithm 2.

Algorithm 2: Determine Column-Mean-based Residue of a bicluster.

Data: A bicluster in the form of a matrix.
Result: The CMR of the matrix.

```
/* columns' means without missing values */
cols_means ← columns_nan_means(matrix)
/* count non missing values */
count ← 0
/* residue accumulator */
residue ← 0

/* iterate elements of the bicluster */
for row in matrix do
    for col in row do
        elem ← matrix[row,col]
        /* only non missing elements contribute */
        if elem not nan then
            residue ← residue + pow((elem - cols_means[col]), 2)
            count ← count + 1
    end
end

return 1/count * residue
```

There is one particular situation when computing the value of rating pattern measure. When the rated items' interception is zero, meaning the active user did not rate any bicluster items, it does not make sense to compute the measure since we would only be creating missing values in the bicluster. In those cases, we consider the value of the rating pattern to be infinite.

Combine User-Bicluster "Similarities"

After the similarities computation, each user has two similarity values for each bicluster (rated items' interception, and rating pattern). Combining both values into one, for instance, multiplying both values, does not seem a reasonable solution. Unlike the rated items' interception, the range of the rating pattern measure is not bounded between zero and one, which means the combination of both values could lead to unpredictable and undesirable results.

Instead of combining these two values, we use them to create two ranked lists of biclusters for each user. This means that each user has two lists of biclusters, each one ordered by how "similar" the biclusters are to the user according to the respective measure. After that, we combine the orders of both ranked lists and create a single list with the final order of biclusters.

The process to merge both bicluster ranking lists is based on the mean position the biclusters on both lists. For instance, recall the User-Bicluster Similatities module in Figure 5.1, in which the ranked lists are: $l_1 = [Bic1, Bic6, Bic2, Bic3, Bic4, Bic5]$ and $l_2 = [Bic6, Bic2, Bic3, Bic4, Bic1, Bic5]$. The mean positions of each biclusters are $Bic1 = \frac{1+5}{2} = 3, Bic2 = \frac{3+2}{2} = 2.5, Bic3 = \frac{4+3}{2} = 3.5, Bic4 = 4, Bic5 = 6, Bic6 = 1.5$. Sorting the mean positions of each bicluster, we obtain the final ranking: $l_{final} = [Bic6, Bic2, Bic1, Bic3, Bic4, Bic5]$. Finally, we select *k* first biclusters of the final list, where *k* is the number of nearest biclusters we want to consider as the neighborhood of the user (NNBics). Algorithm 3 summarizes the whole process.

Algorithm 3: Find the active user's k-nearest-bics.

Data: The active user, a list of biclusters, and a value k.
Result: The k-biclusters most "similar" to the active user.

```
result_intercept ← list()
result_pattern ← list()
for bic in biclustering_solution do
    intercept ← obtain_sim_intercept(user, bic)
    pattern ← obtain_sim_pattern(user, bic)
    result_intercept.append(intercept)
    result_pattern.append(pattern)
end
ordered_bics_sim_intercept ← reverse(sort(result_intercept))
ordered_bics_sim_pattern ← sort(result_pattern)
nearest_bics ← combine(ordered_bics_intercept, ordered_bics_pattern)
return filter(nearest_bics, k)
```

Aggregating the Neighborhood

After selecting the k most similar biclusters to each user, we aggregate the biclusters in each neighborhood in order to create a personalized dataset for each user. The procedure to obtain the personalized dataset consists in merging all the users and items of the bicluster and obtaining the submatrix of the original U-I that corresponds to those rows and columns. For example, considering $Bic2 = \{\{u_1, u_2, u_7\}, \{i_4, i_5\}\}$ and $Bic6 = \{\{u_6, u_8\}, \{i_4, i_6\}\}$ from Figure 5.1, the union set of the users of both biclusters is $U = \{u_1, u_2, u_6, u_7, u_8\}$ and the union set of the items is $I = \{i_4, i_5, i_6\}$. Figure 5.3 illustrates the process.

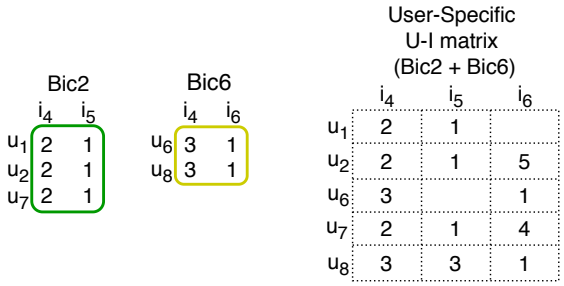


Figure 5.3: Example of aggregating two biclusters to create a new dataset.

Additionally, we add the active user when he is not in the set of users of his own user-specific dataset. This step is crucial because if the active user is not in the dataset, no Collaborative Filtering approach will be able to generate recommendations for him since he would not be in the training data, covering none of the predictions or recommendations.

Adaptive Bicluster Neighborhood Dimension

A critical aspect of the neighborhood approaches is the choice of the hyperparameter k, the number of nearest neighbors. This hyperparameter usually has an enormous impact on the performance of the algorithms, so we created a version of our approach that, for each user, decides how many biclusters should be included in his neighborhood. The choice of the number of nearest bicluster neighbors impacts the dimensions of the resulting personalized matrix directly. Thus, we believe there are two imperative guidelines to follow when deciding how many biclusters should be included.

Firstly, the biclusters should contain data regarding most system items. For our approach, it is decisive that the personalized matrices contain at least most of the system items. Otherwise, the CF models would never be able to output recommendations of the items they did not know existed.

Secondly, the number of biclusters should be controlled so that the dimensions of the personalized matrix does not grow tremendously, damaging the scalability of the approach.

To maximize the coverage of the approach, ideally, each personalized matrix should have a small fraction of the system users, but the entire system items. However, to create the personalized matrix, we use data from the biclusters, which means that USBCF is limited to the items the biclustering algorithm included in the biclustering solution.

We propose a user-adaptive bicluster neighborhood, which instead of filtering the first k biclusters from the ordered list, it selects as many biclusters it needs until it ensures the personalized dataset covers a certain portion of the items included in the biclustering solution.

Despite not being a parameter-free approach (we still have to establish the portion of items we want to guarantee), this type of parameterization is much more intuitive since it does not depend on the size of the biclusters found by the biclustering algorithm.

5.2.3 Learning User-Specific Recommendation Models

Once the personalized datasets are created, a Collaborative Filtering algorithm is trained with them, creating a unique recommendation model for each individual. Despite any Collaborative Filtering approach could be combined with the USBCF approach, we believe the User-based CF and the Item-based CF are suitable choices. These approaches are the basis of CF, recognized for producing rating predictions with exceptional quality. When allied with USBCF, the USBCF approach can create smaller and denser U-I matrices which allow the models to scale and perform the recommendation tasks effectively.

5.3 Final Remarks

This chapter proposed USBCF, a user-specific Collaborative Filtering approach that uses biclustering as a step to scale and improve traditional CF methods. The biclustering task is used to group users with similar rating patterns in a subset of items. Then, a personalized U-I matrix is created for each user, resulting from the aggregation of some biclusters. Finally, a classic CF approach is trained for each of the personalized U-I matrices.

The USBCF was designed to enhance traditional Collaborative Filtering approaches and overcome the major limitation of the state-of-the-art biclustering-based CF (BBCF), the coverage capability. The proposed approach offers **scalability** in large-scale datasets by dividing it into smaller personalized U-I matrices. Both the training and recommendation phases can be performed in an embarrassingly parallel manner since each user has its own model. Besides scalability, biclustering can also improve the traditional U-I approaches in high **data sparsity** environments since it creates denser matrices to train the approaches. Moreover, biclustering offers a solution that can be used to promote **comprehensibility** and **explainability** of the recommendation.

For upcoming work on the USBCF, we intend to follow these major directions:

- **Memory-dependency.** The biclustering algorithm adopted, QUBIC, needs to have the entire U-I matrix in memory to perform the biclustering task, which can be impractical when dealing with large-scale recommendation datasets. We believe a solution to this problem is adopting a biclustering algorithm that takes advantage of matrix sparsity, such as BicNET [50].

- **Adapt to new biclustering solutions.** We want to understand how different biclustering solutions can enhance the performance, for instance, discovering order-preserving biclusters instead of the constant-values on the columns like the ones discovered by QUBIC.

- **Upgrade the user-bicluster "Similarity".** We intend to develop a new user-bicluster measure that, in a single step, privileges biclusters with items the active user rated, as well as considers the values of those ratings.

- **Maximize coverage.** Even though we significantly improved the BBCF regarding coverage, some systems items are still discarded during the biclustering phase. By including those items in some biclusters, we can maximize the coverage of the models.

- **Enhance scalability.** Even though our method improves the scalability of CF models by dramatically reducing the number of users in each matrix, it still needs to cover most system items. This means that using an Item-based approach, the model's training phase still makes n^2 comparisons between item vectors.

Chapter 6

USBCF - A Case Study on Movie Recommendation

In this chapter, we discuss the issues related to the evaluation of our approach. In order to develop and evaluate our proposed methodology of Collaborative Filtering, we conducted a set of experiments on the MovieLens-100k dataset, provided by the GroupLens research lab[1]. In this case study, we compare the proposed approach's performance against baseline and state-of-the-art methods for both rating prediction and Top-*N* recommendation tasks. In particular, we intend to answer the following research questions:

- **RQ1.** How does each module/feature of our approach affect the performance of the model?
- **RQ2.** How does the number of biclusters chosen for each user influence the performance of our approach?
- **RQ3.** How does USBCF compare against baseline CF approaches in terms of quality of predictions, quality of recommendations, ranking relevance and coverage?
- **RQ4.** Can USBCF surpass the main limitations presented by state of the art biclustering based CF (BBCF) [101]?
- **RQ5.** What are the main limitations of the proposed approach?

6.1 Evaluation Methodology

We first introduce the details of the experimental platform, including the used software. Then, we describe the dataset and evaluation metrics. Lastly, we specify the approaches that served as a benchmark to compare with our model.

[1] `https://grouplens.org/datasets/movielens/20m/`

6.1.1 Experimental Platform and Software

The models were trained on a machine with Intel Xeon 2.40GHz having thirty-two cores and 32GBs of RAM. For generating the predictions and recommendations, Google Colaboratory was used with 2-core Xeon 2.20GHz and 13GBs of RAM. Regarding the implementation of the approaches, the USBCF and the BBCF were implemented in Python. As for the remaining approaches, it was used their available implementation in LensKit [26] (version 0.10.1), a Python-based toolkit for recommender systems.

6.1.2 Benchmark Dataset

All the experiments were run on the MovieLens-100k dataset because it is one of the most popular publicly available benchmark datasets to serve as domain testing for new approaches. This version of the MovieLens' datasets contains 100,000 ratings (1-5) from 943 users on 1682 movies. The data is in the form of triplets <*user, item, rating*>. Each user has a minimum of 20 ratings made and a maximum of 737, but the average is 106 ratings per user with a standard deviation of 100 ratings. The sparsity of the MovieLens-100K is 93.6%, in other words, approximately 6.4% of the entries are filled.

6.1.3 Evaluation Metrics

To measure the rating prediction accuracy of the recommendation algorithms, we used two standard metrics, namely *Mean Absolute Error (MAE)* and *Root Mean Squared Error (RMSE)*, already defined in Chapter 2. As the prediction accuracy of a recommendation system in many cases, especially in collaborative filtering systems, grows with the amount of data, some algorithms may provide recommendations with high quality, but only for a small portion of the items. In this work, we also evaluate the overall *coverage* capability of the models, which is defined as the percentage of user-item pairs for which a prediction can be made [95].

To evaluate the approach in the context of Top-N recommendation, we opted for using the classic *precision/recall* framework. As these metrics are often conflicting in nature (a more extensive recommendation list tends to increase recall but decrease precision), we also include the standard F_1 metric in our evaluation. We compute the precision, recall, and F_1 score for each individual user and calculate the average values to obtain the precision, recall, and F_1 score of the model in the test-set. We also include the *nDCG* score in our evaluation to evaluate the ranking relevance of the generated recommendations.

The results reported in this evaluation are the values obtained over runs of five-fold cross-validation with 80% of the ratings as training data and the remaining 20% for testing in each run. Besides the traditional cross-validation, in Appendix B we also report the results for five-fold cross-validation where the users are the basis of splitting, instead of the rows. When we use the user-based splitting, we make sure that each test set has a

predefined fraction of each user's ratings, allowing us to control the experimental conditions on a user-by-user basis. For instance, avoiding that, by chance, all the ratings from a user are put in the test set (and no data from some users in the training set). In these experiments, we decided to use 20% of each users' data for testing, this means that if a user u has 20 ratings in the entire dataset, 4 ratings are randomly sampled to be in the test set, and the remaining are put in the training set. An illustration of this user-based split cross-validation is shown in Figure 6.1.

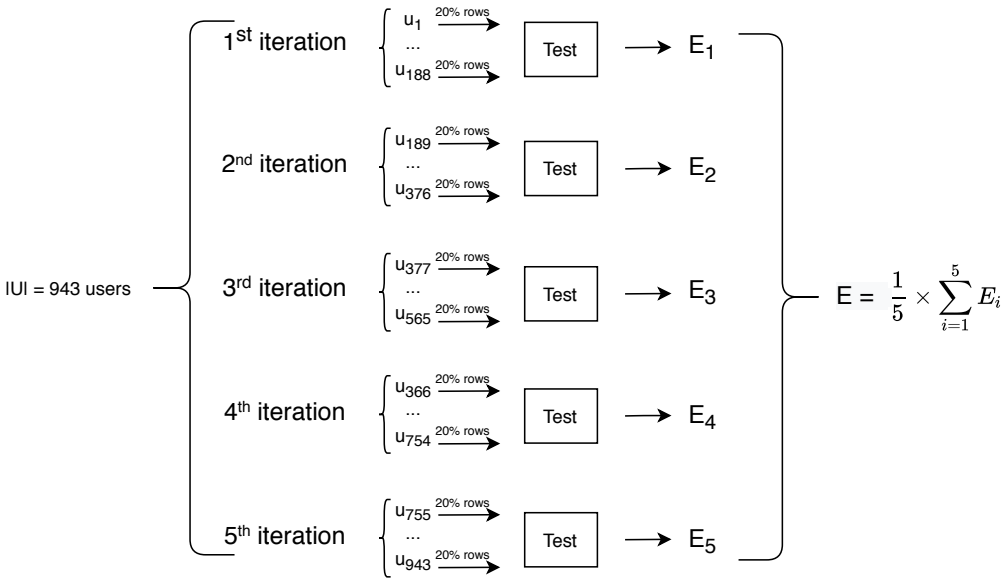


Figure 6.1: User-based split cross-validation.

6.1.4 Approaches for Comparative Analysis

To compare the performance of the USBCF approach, we also evaluated some popular rating prediction methods:

- **Baseline Rating Predictor (Bias) [32].** A user-item bias rating prediction algorithm, denoted by : $r_{u,i} = \mu + b_i + b_u$, where μ is the global mean rating, b_i is item bias, and b_u is the user bias.

- **User-based CF (USCF) [92].** User-based CF using normalized cosine-similarity as similarity function and weighted-average as aggregation function.

- **Item-based CF (IBCF) [96].** Item-based CF using normalized cosine-similarity as similarity function and weighted-average as aggregation function.

- **Singular Value Decomposition (SVD-ALS) [121].** Classic Matrix Factorization algorithm that uses Alternating-Least-Squares with Weighted Regularization to create a low-rank approximation of the U-I matrix.

- **Singular Value Decomposition (SVD-SGD) [26].** Standard Matrix Factorization algorithm like SVD-ALS, but the model's parameters are learned with TensorFlow's gradient descent instead of the alternating-least-squares algorithm.

- **SVD Model by Simon Funk (FunkSVD) [32].** The famous SVD-inspired approach popularized by Simon Funk during the Netflix Prize contest. It uses regularized stochastic gradient descent to train the user-feature and the item-feature matrix.

- **Biclustering-based CF (BBCF) [101].** Biclustering based CF proposed by Singh and Mehrotra in 2018.

We also include the following Top-N-only approaches:

- **Random Item Recommendation (Random) [26].** Model that recommends items randomly.

- **Most Popular (Popular) [26].** Model that recommends the most frequently-consumed items a user has not yet interacted with.

- **Implicit Matrix Factorization (ImplicitMF) [54].** Implicit matrix factorization trained with alternating least squares. This approach binarizes the I-U matrix, considering all the ratings as positive-interactions and the missing values as zero.

6.2 Results and Discussion

This section presents the experiments conducted to evaluate the proposed approach, as well as its results and discussion.

6.2.1 Baseline Approaches - Models and Parameterization

Correctly setting up methods by tuning its hyperparameters is central for a fair evaluation. Rendle et al. [91] highlighted this by carefully setting up baseline methods and showing that they outperformed numerous recent publications of approaches reported as being superior to the baselines on the MovieLens-10M dataset. Thus, we start our evaluation by testing how the baseline approaches' parameterization setup affects their performance in the MovieLens-100k dataset. Then, we select a tuned model from each approach, compare it, and use it to serve as a benchmark to evaluate the USBCF model.

Basic Models

Regarding the basic non-Collaborative Filtering methods, there are no major hyper-parameters to optimize. In Table 6.1 we present the results we obtained with these models basic models.

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
Bias	0.948 ± 0.004	0.748 ± 0.003	100.00	0.012 ± 0.004	0.002 ± 0.001	0.016 ± 0.001	0.002 ± 0.001
Random	-	-	-	0.014 ± 0.001	0.002 ± 0.000	0.017 ± 0.001	0.004 ± 0.000
Popular	-	-	-	0.155 ± 0.002	0.034 ± 0.001	0.059 ± 0.001	0.061 ± 0.001

Table 6.1: Results of the basic models.

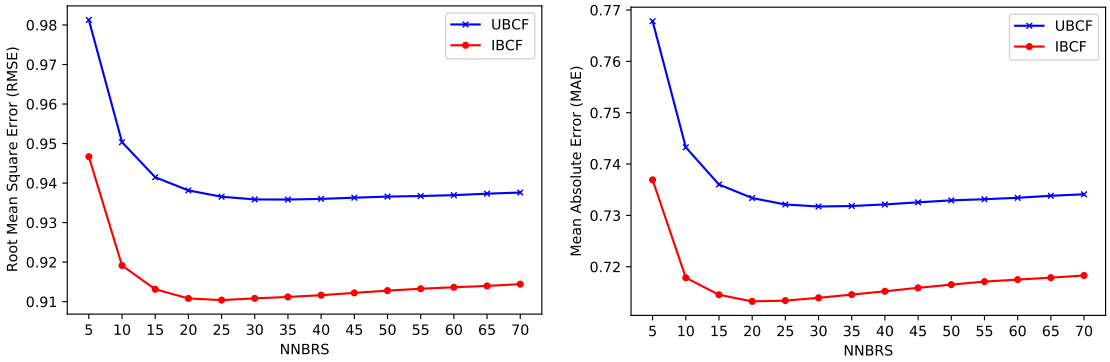
UBCF and IBCF

The maximum number of users/items in the neighborhood (NNBRS) for both UBCF and IBCF is known for significantly impacting the quality of the produced model [96]. So, we studied the sensitivity of both approaches to this parameter. Table 6.2 summarizes the main results of this experiment.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
UBCF	nnbrs=10	0.950 ± 0.005	0.743 ± 0.004	**99.72**	**0.016 ± 0.002**	**0.002 ± 0.000**	**0.017 ± 0.000**	**0.004 ± 0.001**
	nnbrs=20	0.938 ± 0.006	0.733 ± 0.005	99.72	0.009 ± 0.002	0.001 ± 0.000	0.015 ± 0.001	0.002 ± 0.000
	nnbrs=30	**0.936 ± 0.005**	**0.732 ± 0.005**	99.72	0.007 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.002 ± 0.000
	nnbrs=40	0.936 ± 0.006	0.732 ± 0.005	99.72	0.006 ± 0.001	0.001 ± 0.000	0.015 ± 0.001	0.001 ± 0.000
	nnbrs=50	0.937 ± 0.005	0.733 ± 0.005	99.72	0.005 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=60	0.937 ± 0.005	0.733 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=70	0.938 ± 0.005	0.734 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
IBCF	nnbrs=10	0.919 ± 0.003	0.718 ± 0.002	**99.51**	**0.042 ± 0.003**	**0.007 ± 0.001**	**0.026 ± 0.001**	**0.013 ± 0.001**
	nnbrs=20	**0.911 ± 0.003**	**0.713 ± 0.003**	99.51	0.042 ± 0.003	0.007 ± 0.001	0.026 ± 0.001	0.012 ± 0.001
	nnbrs=30	0.911 ± 0.002	0.714 ± 0.002	99.51	0.040 ± 0.004	0.006 ± 0.001	0.025 ± 0.002	0.011 ± 0.002
	nnbrs=40	0.912 ± 0.003	0.715 ± 0.002	99.51	0.040 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.011 ± 0.001
	nnbrs=50	0.913 ± 0.003	0.717 ± 0.003	99.51	0.039 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.010 ± 0.001
	nnbrs=60	0.914 ± 0.003	0.718 ± 0.003	99.51	0.038 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.010 ± 0.001
	nnbrs=70	0.914 ± 0.003	0.718 ± 0.003	99.51	0.037 ± 0.004	0.006 ± 0.001	0.024 ± 0.001	0.010 ± 0.001

Table 6.2: Results of the User-based CF and Item-based CF varying the maximum number of neighbors' parameter

As we can observe in Figure 6.2, regarding prediction accuracy, there are significant improvements in both approaches when we grow the neighborhood size, especially between 5 to 20/25 for the number of neighbors' parameter, which is similar to the results obtained by Sarwar et al. in the same dataset [96]. Concerning the quality of Top-*N* recommendations, a smaller neighborhood tends to generate better recommendations. So, for the rest of the evaluation, we set the maximum number of neighbors to 20 in both the UBCF and the IBCF approaches.



(a) RMSE of different NNBRS setups.

(b) MAE of different NNBRS setups.

Figure 6.2: Sensitivity of UBCF and IBCF to the size of the neighborhood.

Matrix Factorization Methods

For the Matrix Factorization-based approaches, we performed a grid search over the two main hyperparameters: the number of features and the regularization factor of their respective optimization algorithm. For the number of features we set up a grid search over seven values $\{5, 10, 20, 50, 100, 200, 500\}$. Whereas for the regularization factor the used grid was $\{0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.5\}$. In Table 6.3 we present the setup with best performance for each method. The complete results of this grid-search can be found in Table B.2.

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
FunkSVD (f=10, reg=0.2)	0.939 ± 0.003	0.742 ± 0.003	99.81	0.068 ± 0.002	0.011 ± 0.001	0.034 ± 0.001	0.021 ± 0.001
SVD-ALS (f=100, reg=0.1)	0.915 ± 0.003	0.719 ± 0.003	99.81	0.071 ± 0.002	0.012 ± 0.000	0.035 ± 0.001	0.026 ± 0.001
SVD-SGD (f=100, reg=0.5)	0.965 ± 0.006	0.759 ± 0.004	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
ImplicitMF (f=500, reg=0.01)	-	-	-	0.252 ± 0.002	0.064 ± 0.002	0.095 ± 0.002	0.112 ± 0.003

Table 6.3: Results of the Matrix Factorization models with best performance.

We can not guarantee that those models' setups are optimal as a more exhaustive grid search could deal to better performances. Moreover, we did not contemplate all the hyperparameters of each method, for instance, the learning-rate of the optimization algorithm could also further improve the performance of the models.

6.2.2 BBCF

Singh and Mehrotra in their BBCF work [101] used an Item-based approach as the recommendation algorithm. To evaluate BBCF, the authors fixed the neighborhood of the IBCF

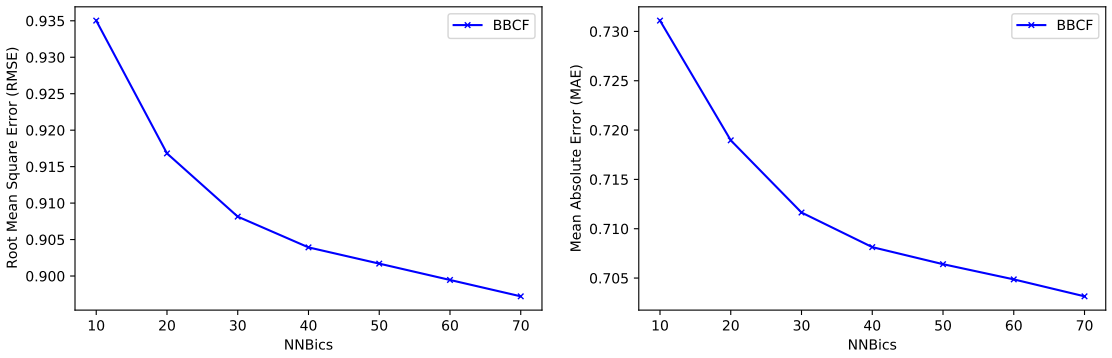
to 10 and used adjusted-cosine as the similarity measure. They also studied how the allowed overlapping percentage between biclusters affected the performance of BBCF and concluded that a higher degree of overlap tends to lead to better results.

In our experiments, instead of the IBCF with the adjusted-cosine similarity, we used IBCF with normalized cosine similarity and a neighborhood size of 20 because those were the settings we set in the comparative IBCF baseline method. Regarding overlapping percentage between biclusters, Singh and Mehrotra concluded that higher values of overlapping significantly improve the performance of the approach, so we set it to 99%, so that it considers maximal biclusters. Similarly to Singh and Mehrotra, we also investigate the impact that the number of nearest biclusters (NNBics) has on the quality of predictions and recommendations. Table 6.4 summarizes the results.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF	NNBics=10	0.935 ± 0.006	0.731 ± 0.007	27.85	**0.248 ± 0.010**	**0.020 ± 0.001**	**0.038 ± 0.001**	**0.053 ± 0.004**
	NNBics=20	0.917 ± 0.005	0.719 ± 0.005	35.30	0.232 ± 0.013	0.018 ± 0.001	0.035 ± 0.002	0.049 ± 0.003
	NNBics=30	0.908 ± 0.005	0.712 ± 0.004	38.95	0.218 ± 0.012	0.017 ± 0.001	0.033 ± 0.002	0.047 ± 0.002
	NNBics=40	0.904 ± 0.005	0.708 ± 0.003	41.61	0.211 ± 0.011	0.017 ± 0.001	0.032 ± 0.001	0.045 ± 0.003
	NNBics=50	0.902 ± 0.006	0.706 ± 0.004	43.55	0.203 ± 0.010	0.016 ± 0.001	0.031 ± 0.001	0.044 ± 0.002
	NNBics=60	0.899 ± 0.006	0.705 ± 0.004	44.96	0.196 ± 0.011	0.015 ± 0.001	0.030 ± 0.001	0.043 ± 0.002
	NNBics=70	**0.897 ± 0.005**	**0.703 ± 0.003**	**46.16**	0.188 ± 0.011	0.015 ± 0.001	0.029 ± 0.001	0.040 ± 0.002

Table 6.4: Sensitivity of BBCF to the number of biclusters in the neighborhood.

Our experiments' results are very similar to the ones Singh and Mehrotra obtained in their work, as prediction accuracy tends to be better when the number of nearest biclusters is between 50 and 100. In Figure 6.3 we can see that in our experiments both, RMSE and MAE are minimal for the largest biclusters' neighborhoods. Due to experimental constraints, we were not able to test the approach with even larger neighborhoods. However, according to the BBCF's original work, that it would not result in further accuracy gains.



(a) RMSE of different NNBics setups.

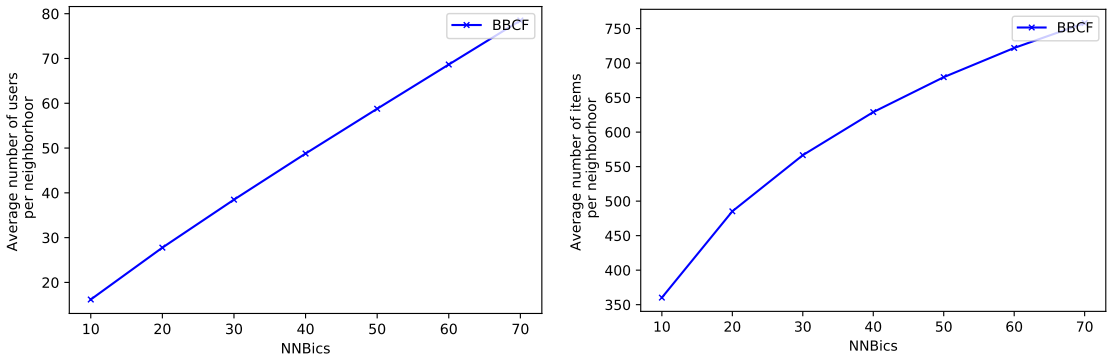
(b) MAE of different NNBics setups.

Figure 6.3: Sensitivity of BBCF to the size of the biclusters' neighborhood.

On the other hand, the recommendation quality peaks with lower numbers of nearest biclusters. Coverage of the predictions is clearly the major limitation of the BBCF ap-

proach. It tends to improve as the number of biclusters per neighborhood increases, but even for NNBics=70 it covers only 46% of the prediction pairs which is considered poor.

The purpose of this approach is to provide quality predictions in an acceptable time, tackling the scalability issues of the traditional memory-based approaches. So, we also examine the average dimensions of the personalized dataset produced for each user. In Figure 6.4, we can see the dimensions of the personalized datasets increases as we use larger values for the NNBics parameter. These results show that, as expected, larger values of NNBics lead to more complex personalized models, impacting the scalability of the approach.



(a) Average number of users per NNBics setup. (b) Average number of items per NNBics setup.

Figure 6.4: Effect of the number biclusters in the neighborhood on the average size of the personalized dataset.

For the remaining of the experiments, we set the number of nearest biclusters to 50. From our experiments, it appears to be a value that balances the performances of the algorithm and maintains a reasonable average size of the personalized datasets.

6.2.3 USBCF

To assess the impact of each USBCF feature in the performance of the proposed approach, we conducted some experiences. We start with the BBCF (NNBics=50) as a baseline, and then we individually add the new features that assemble the USBCF, evaluating its performance.

BBCF-NoWeight

We start by investigating if, as we stated in Section 5.2.2, our adaptation to the previously adopted rated items' interception does not damage the performance of the model. Here we evaluate the BBCF approach with the slight modification of not favoring the biclusters with more users. We refer to this model as "BBCF-NoWeight".

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF-NoWeight (NNBics=50)	0.904 ± 0.007	0.708 ± 0.004	45.30	0.184 ± 0.007	0.018 ± 0.008	0.034 ± 0.001	0.044 ± 0.003

Table 6.5: Results of the BBCF-NoWeight model.

As Table 6.5 shows, the simplification to the rated items' interception similarity had minimal impact on the model's performance. It even improved the coverage of BBCF (NNBics=50) by almost 2%, maintaining the prediction and recommendation tasks' overall quality. This result supports our decision of adapting the user-bicluster similarity measure.

BBCF-CombineSimilaritity-MSR

Another important feature of our approach is the new user-bicluster similarity which combined the rated items' interception with the rating pattern measure. In this approach we include our user-bicluster similarity in the BBCF (NNBics=50) baseline to understand if it improves the original BBCF approach. We refer to this new model as "BBCF-CombinedSimilaritity-MSR".

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF-CombineSimilaritity-MSR (NNBics=50)	0.855 ± 0.256	0.694 ± 0.238	0.30	0.287 ± 0.025	0.011 ± 0.002	0.020 ± 0.003	0.039 ± 0.010

Table 6.6: Results of the BBCF-CombinedSimilarity-MSR model.

From Table 6.6 we can see an improvement regarding the average quality of the predictions followed by a significant increase in the standard deviation. This result is mainly caused by the immense lack of coverage from the model, being able to rate only 3% of the user-item pairs.

Concerning the lack of coverage of this model, we believe it can be explained by the characteristics of the new similarity measure. The similarity measure adopted by the BBCF, despite not considering the biclusters' rating pattern, ensures the chosen biclusters include items that the user had previously rated. When we combine it with the rating pattern measure, we reduce its influence, and the model creates personalized sparser matrices, which leads to a lack of coverage from the resulting model.

USBCF-MSR and USBCF-CMR

To measure the rating pattern "similarity" between users and biclusters, we tested the proposed USBCF approach with two different quality measures as described in Section 5.2.2. The main difference to the "BBCF-CombineSimilarity-MSR" model tested previously, is the step responsible for adding the user to his user-specific dataset. This experiment,

demonstrates the importance of adding the user to his dataset regarding the models' coverage capability and understand if the small differences between the two quality measures can impact the performance of the models. We refer as "USBCF-MSR" to the model using the Mean-Square-Residue adaptation, and as "USBCF-CMR" to the model using the proposed Mean-Column-based-Residue. Table 6.7 presents the results obtained for models with using a bicluster neighborhood of 50 biclusters.

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
USBCF-MSR (NNBics=50)	0.927 ± 0.004	0.725 ± 0.003	84.55	0.073 ± 0.001	0.012 ± 0.001	0.035 ± 0.001	0.024 ± 0.001
USBCF-CMR (NNBics=50)	0.927 ± 0.005	0.725 ± 0.003	**85.71**	0.073 ± 0.001	0.012 ± 0.000	0.035 ± 0.001	0.024 ± 0.000

Table 6.7: Results of the USBCF-MSR and the USBCF-CMR models.

Both USBCF-MSR and USBCF-CMR models obtained very similar prediction and recommendation quality. USBCF-CMR covered slightly more test predictions. When compared with the ones from the baseline methods, these results were only surpassed by the IBCF and the SVD-ALS regarding the predictions' accuracy. In terms of recommendations, the USBCF models had similar performance to the SVD-ALS, being only surpassed by the Top-N-only baselines (Popular and ImplicitMF).

Compared to the BBCF (NNBic=50), the USBCF models could not maintain the impressive MAE and RMSE results of the BBCF. Nevertheless, while the BBCF approach was only able to cover 43.55% of the test predictions, the USBCF covered 85.71%, which is an immense improvement. We believe the upper hand of the BBCF regarding accuracy is directly linked to the model's lack of coverage. The lack of coverage of the BBCF approach derives from the fact user-specific neighborhood not containing the active user neither some items, which means the bicluster algorithm did not cluster the active user nor similar users with those items. A possible reason for this to happen is the fact that there are no strong correlations connecting the active user and the items, making it harder for any method to predict those pairs. If the model could predict the remaining user-item test pairs, it would necessarily produce less accurate outputs.

These results show that the proposed methodology can successfully improve the coverage of the state-of-the-art biclustering-based Collaborative Filtering while continuing to produce good quality predictions.

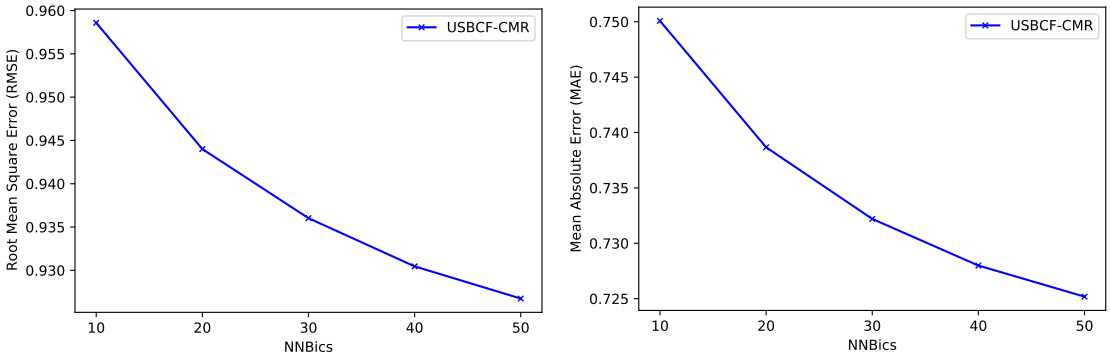
USBCF-CMR NNBics Sensitivity

To further evaluate the proposed approach, we tested the impact the number of neighborhood biclusters parameter has on the USBCF-CRM performance. Table 6.8 presents the obtained results.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
USBCF-CMR	NNBics=10	0.959 ± 0.009	0.750 ± 0.008	61.44	**0.090 ± 0.003**	**0.015 ± 0.000**	**0.040 ± 0.001**	**0.029 ± 0.001**
	NNBics=20	0.944 ± 0.003	0.739 ± 0.003	72.91	0.083 ± 0.002	0.014 ± 0.000	0.037 ± 0.000	0.027 ± 0.001
	NNBics=30	0.936 ± 0.004	0.732 ± 0.003	78.74	0.080 ± 0.001	0.013 ± 0.000	0.037 ± 0.001	0.026 ± 0.001
	NNBics=40	0.930 ± 0.004	0.728 ± 0.003	82.38	0.076 ± 0.001	0.012 ± 0.000	0.036 ± 0.000	0.025 ± 0.001
	NNBics=50	**0.927 ± 0.005**	**0.725 ± 0.003**	**85.71**	0.073 ± 0.001	0.012 ± 0.000	0.035 ± 0.001	0.024 ± 0.000

Table 6.8: Sensitivity of the USBCF-MCR to the number of biclusters in the neighborhood

The results from Table 6.8 show very similar behaviour to the ones obtained when we varied the value of NNBics parameter in the BBCF approach. The rating prediction quality improves as the number of NNBics increases, as Figure 6.5 shows. On the other hand, the quality decreases, which is the standard behaviour for all the rating prediction approaches included in this evaluation. Focusing on the coverage capability, when compared to the results obtained by BBCF in Table 6.4 these results strengthen the argument that our proposed methodology can overcome the major limitation of the BBCF approach, covering a munch larger portion of the user-item test pairs.

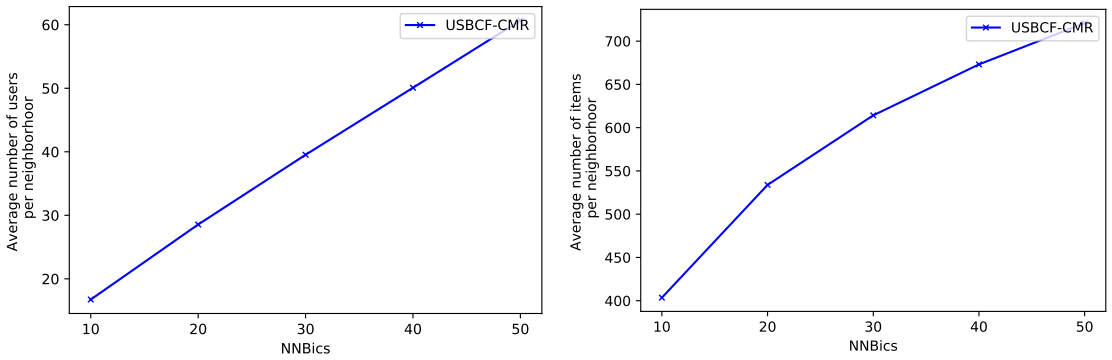


(a) RMSE of different NNBics setups.

(b) MAE of different NNBics setups.

Figure 6.5: Sensitivity of USBCF-CMR to the size of the biclusters' neighborhood.

Similarly to the BBCF experiment, we also examined the average dimensions of the bicluster-based personalized matrices. In Figure 6.6 a) and b), we see that as the value of NNBics increases, the average number of users per bicluster grows linearly, whereas the growth of number of items seems to dissolve for larger values of NNBics.



(a) Average number of users per NNBics setup. (b) Average number of items per NNBics setup.

Figure 6.6: Effect of the number biclusters in the neighborhood on the average size of the personalized dataset.

The results of this experiment show the importance of the hyperparameter that controls the number of biclusters that form a neighborhood. Increasing this hyperparameter seems to lead to better prediction quality and coverage, which is expected since if we used all of the discovered biclusters, we would be creating a matrix similar to the original U-I matrix, instead of smaller and personalized matrices. There is an evident trade-off between scalability (with smaller neighborhoods), and performance (with larger neighborhoods).

USBCF-CMR-Adaptive

As described in Section 5.2.2, we created a version of USBCF which tries to overcome the dependency of the NNBics hyper-parameter. This version of the proposed approach instead of the NNBics parameter, it uses a parameter that controls the minimum percentage of the items, discovered by the biclustering algorithm, to include in each neighborhood. We refer to this parameter as "ItemsPortion". In this experiment, we evaluate the adaptive bicluster neighborhood version of USBCF using ItemsPortion = 0.7, which adds biclusters to the neighborhoods until the neighborhood has data about 70% of the possible items. In Table 6.9, we summary the results obtained for this version of the approach.

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
USBCF-CMR-Adaptive (ItemsPortion=0.7)	0.923 ± 0.006	0.723 ± 0.004	88.01	0.069 ± 0.001	0.012 ± 0.000	0.034 ± 0.001	0.023 ± 0.001

Table 6.9: Results of the Adaptive Bicluster Neighborhood Dimension version of the USBCF-CMR.

The results show that the USBCF-CMR-Adaptive (ItemsPortion=0.7) surpassed the USBCF-CRM (NNBics=0.5) regarding prediction quality, including coverage capability. We analyzed the average dimensions of the personalized matrices produced by USBCF-

CMR-Adaptive (ItemsPortion=0.7) and it included, on average, 71 users and 770 items, which is around 7.1% of the system users and 45% of the system items.

6.3 Final Remarks

In this case study, we performed a set of experiments on the MovieLens-100k benchmark dataset with the USBCF approach. In particular, we studied the proposed approach, comparing it against baseline CF methods and the state-of-the-art biclustering-based CF.

In Collaborative Filtering literature, it is challenging to standardize performance values for the CF methods. It is prevalent for different authors to obtain different results for the same approaches in the same dataset since each work opts for a particular evaluation methodology, using different parameterizations. However, for the MovieLens-100k, to our knowledge, the best accuracy values are around 0.91/0.92 of RMSE, and 0.71/0.72 of MAE [55, 90]. Our evaluation of the USBCF approach confirms that our method can obtain state-of-the-art accuracy results. When using a bicluster neighborhood size of 50, the USBCF model obtained an RMSE of 0.927, only slightly outperformed by Item-Based CF and SVD-ALS. We believe that our approach can produce even lower prediction errors, as we did not fully explore it regarding its different setups. The biclustering algorithm plays a significant role in the BBCF approach, as its solution is based on the entire method. It could be interesting to evaluate the approach with QUBIC under different parameterization settings, for instance, reducing the biclusters' quality by allowing missing values. Alternatively, we could try USBCF using a different biclustering algorithm, finding types of biclusters.

Nevertheless, the primary goal of the USBCF approach is not to obtain the best prediction errors but rather enhance the traditional method by improving its scalability. Clustering-based Collaborative Filtering methods are known for producing worse prediction quality than the baseline-neighborhood methods. However, while dividing the data into clusters may reduce the recommendations' accuracy, it may be a worthy trade-off between accuracy and throughput in a real application.

The results of our experiments show that despite the fact that the state-of-the-art biclustering-based CF (BBCF) outperformed USBCF regarding prediction accuracy, the proposed approach was able to successfully overcome BBCF's major limitation, improving its coverage capability by 43%.

To further validate our approach, we intend to further evaluate USBCF in a larger dataset with different characteristics (MovieLens-1M), with even more exhaustive experiments, including throughput, measuring the time taken by model's training and prediction phases. A more in-depth evaluation is needed to understand if USBCF can effectively scale traditional memory-based approaches, supporting its applicability in real-world recommender systems.

Chapter 7

Conclusions and Future work

Biclustering has gained increasing attention in the literature as an effective advanced clustering technique with applicability not only in gene expression data but also in different contexts, such as text-mining and marketing analysis [76].

In the Collaborative Filtering (CF) domain, clustering-based approaches are popular as a dimensionality reduction mechanism to work with large-scale datasets. Despite the fact that few works explored biclustering in a CF scenario, biclustering has shown potentialities to further improve the clustering-based CF literature by taking advantage of the existent duality between users and items, instead of considering only the similarities between users or items, individually, like most CF algorithms.

In this context, this work provides a review of the traditional Collaborative Filtering recommendation scenario, presenting CF's standard settings, tasks, and challenges. Moreover, it describes state-of-the-art approaches and discusses the standard evaluation methodology.

Then, we survey the biclustering domain, define the main concepts, and interpret how different authors included this clustering task in the Collaborative Filtering domain. We also reviewed some works incorporating the time-dimension in classic neighborhood-based CF algorithms, as we believe it can also be easily incorporated in bicluster-based CF approaches.

The main contribution of this work is a user-specific CF approach that uses biclustering as a sub-step to scale and improve traditional CF methods. The process to develop the USBCF approach included three main stages.

The first stage evaluated the potentialities of BicPAM, a state-of-the-art pattern-based biclustering algorithm. We tested its performance against two other biclustering algorithms, which had already been used in the discovery of biclusters in U-I interaction matrices (xMOTIFs and QUBIC).

In the second stage, and using as reference the state-of-the-art biclustering-based CF approach (BBCF), we proposed a novel bicluster-based method, which overcomes the coverage limitation presented by BBCF. During the process of developing USBCF, we created

a new methodology to measure the "similarity" between a user and a bicluster. Furthermore, we adapted the Mean-Square-Residue (MSR) measure, which evaluates the residue within a bicluster, so that it does not consider missing values when computing the residue. We also created our own bicluster quality measure, Mean-Column-Residue (MCR), which is a novel measure specific for biclusters with constant-values on the columns, and adapted it so that missing values do not contribute to the biclusters' residue.

Finally, we validated the approach using the MovieLens-100k benchmark dataset. We demonstrate that USBCF is an improvement when compared to the previously proposed bicluster-based CF methods and can obtain similar accuracy scores to the state-of-the-art CF methods while improving scalability.

We believe this work displays the potentialities of the biclustering technique for recommendation purposes. Despise the promising results of these primary tests, we want to further improve and validate the USBCF approach in the future. Bellow, we highlight potentially relevant research avenues for future work:

- **Incorporate a time-exploiter technique in USBCF.** There is strong scientific evidence showing that including time-information in the CF algorithms can significantly improve the methods' accuracy [66, 110]. To our knowledge, there is still no work in the literature in the direction of studying a bicluster-based CF technique exploiting the temporal dynamics of the data. In Section 2.3 we already identified some techniques which could easily be included in the USBCF approach.

- **Explore different QUBIC's parameterization setups.** The biclustering algorithm plays a significant role in the USBCF approach. In Chapter 5 we proposed the usage of QUBIC with a standard parameterization. However, we could, for instance, allow a controlled percentage of noisy elements to be included in the biclusters. By investigating how its parameterization affects the CF approach's quality, it may be possible to improve the USBCF methodology.

- **Different biclustering algorithms.** In Chapter 5 we used QUBIC as our biclustering algorithm for the USBCF approach. However, any algorithm could be used instead, with minor modifications to the overall methodology. During this project's development, new biclustering algorithms were proposed, from which we highlight QUBIC2 [112] and RecBic [74]. Moreover, the usage of BicPAM [48] to explore the same methodology on order-preserving biclusters can also be an interesting experiment.

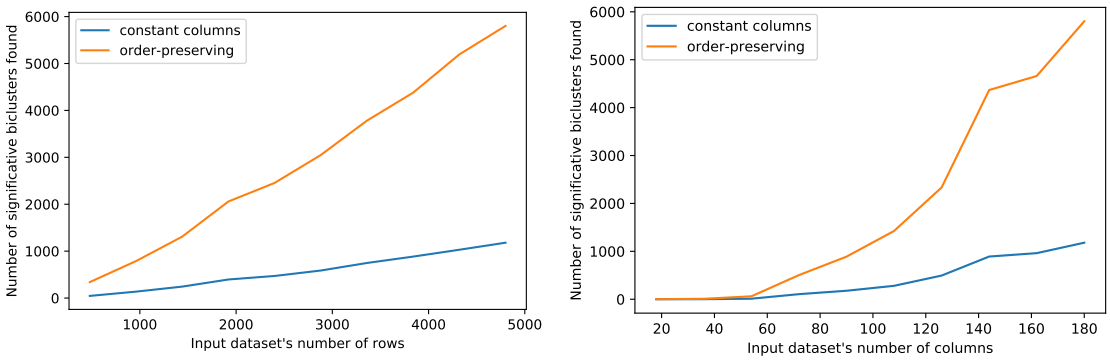
- **Incremental version of USBCF.** The training phase of most CF approaches, including USBCF, is usually computationally expensive, which prohibits frequent re-training of the model. This type of CF approaches can only be deployed in static

settings where the known preferences do not vary with time. An incremental version of USBCF by automatically adding new users and items to biclusters would improve the applicability of the approach to more real-world applications.

- **Exhaustive evaluations.** Evaluate the robustness of the approach in larger datasets, with an especial focus on studying the scalability-accuracy trade-off, to validate the applicability of the proposed approach.

Appendix A

Biclustering Algorithms Study



(a) Incrementing rows

(b) Incrementing columns

Figure A.1: Evolution of the biclusters found according to the number of rows and columns of the input data.

Number of rows	Number of cols	#biclusters found	#significant biclusters ($p < 0.001$)	Avg. #rows per bicluster ($p < 0.001$)	Avg. #cols per bicluster ($p < 0.001$)
480	180	280	46	2.11 ± 0.31	3.76 ± 0.79
960		692	137	2.18 ± 0.44	3.62 ± 0.84
1440		1212	244	2.27 ± 0.62	3.60 ± 0.91
1920		1729	396	2.30 ± 0.62	3.66 ± 0.91
2400		2097	471	2.31 ± 0.64	3.62 ± 0.90
2880		2625	587	2.34 ± 0.78	3.67 ± 0.89
3360		3239	748	2.37 ± 0.85	3.68 ± 0.90
3840		3830	884	2.41 ± 0.98	3.72 ± 0.89
4320		4378	1030	2.44 ± 1.07	3.76 ± 0.90
4800		5075	1180	2.42 ± 1.05	3.78 ± 0.87
4800	18	18	0	-	-
	36	72	1	2.00 ± 0.00	3.00 ± 0.00
	54	230	11	2.36 ± 0.64	2.91 ± 0.79
	72	809	106	2.39 ± 0.90	3.30 ± 0.66
	90	1067	177	2.34 ± 0.79	3.34 ± 0.71
	108	1428	280	2.40 ± 1.09	3.37 ± 0.74
	126	2204	494	2.41 ± 1.05	3.52 ± 0.77
	144	4223	891	2.40 ± 0.97	3.78 ± 0.83
	162	4433	962	2.41 ± 0.98	3.78 ± 0.85

Table A.1: Properties of the constant columns biclustering solutions varying the size of the input data

Number of rows	Number of cols	#biclusters found	#significant biclusters ($p < 0.001$)	Avg. #rows per bicluster ($p < 0.001$)	Avg. #cols per bicluster ($p < 0.001$)
480	180	759	340	2.52 ± 1.11	3.47 ± 1.11
960		1678	788	2.75 ± 1.75	3.41 ± 1.00
1440		2668	1309	3.01 ± 2.87	3.44 ± 0.96
1920		3825	2058	3.11 ± 3.22	3.49 ± 0.97
2400		4502	2454	3.22 ± 3.60	3.47 ± 0.96
2880		5445	3049	3.31 ± 3.88	3.49 ± 0.94
3360		6622	3782	3.38 ± 4.18	3.50 ± 0.95
3840		7657	4379	3.48 ± 4.64	3.53 ± 0.94
4320		8836	5164	3.52 ± 4.78	3.56 ± 0.95
4800		9968	5802	3.60 ± 5.13	3.58 ± 0.94
4800	18	16	1	34.00 ± 0.00	2.00 ± 0.00
	36	67	9	5.89 ± 9.96	2.56 ± 0.50
	54	250	61	5.85 ± 9.20	2.84 ± 0.66
	72	1071	504	4.29 ± 6.13	3.14 ± 0.71
	90	1646	889	3.91 ± 5.21	3.17 ± 0.74
	108	2427	1425	3.71 ± 4.60	3.21 ± 0.79
	126	3905	2333	3.69 ± 4.72	3.32 ± 0.83
	144	7773	4367	3.74 ± 5.62	3.58 ± 0.92
	162	8243	4660	3.71 ± 5.54	3.58 ± 0.93

Table A.2: Properties of the order-preserving biclustering solutions varying the size of the input data

Appendix B

USBCF

B.1 Cross-Validation Results

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
UBCF	nnbrs=5	0.981 ± 0.003	0.768 ± 0.002	**99.72**	**0.023 ± 0.004**	**0.004 ± 0.001**	**0.020 ± 0.001**	**0.006 ± 0.001**
	nnbrs=10	0.950 ± 0.005	0.743 ± 0.004	99.72	0.016 ± 0.002	0.002 ± 0.000	0.017 ± 0.000	0.004 ± 0.001
	nnbrs=15	0.941 ± 0.005	0.736 ± 0.004	99.72	0.011 ± 0.002	0.002 ± 0.000	0.016 ± 0.001	0.003 ± 0.001
	nnbrs=20	0.938 ± 0.006	0.733 ± 0.005	99.72	0.009 ± 0.002	0.001 ± 0.000	0.015 ± 0.001	0.002 ± 0.000
	nnbrs=25	0.937 ± 0.005	**0.732 ± 0.005**	99.72	0.008 ± 0.002	0.001 ± 0.000	0.014 ± 0.001	0.002 ± 0.000
	nnbrs=30	**0.936 ± 0.005**	0.732 ± 0.005	99.72	0.007 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.002 ± 0.000
	nnbrs=35	0.936 ± 0.005	0.732 ± 0.005	99.72	0.006 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=40	0.936 ± 0.006	0.732 ± 0.005	99.72	0.006 ± 0.001	0.001 ± 0.000	0.015 ± 0.001	0.001 ± 0.000
	nnbrs=45	0.936 ± 0.005	0.733 ± 0.005	99.72	0.005 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=50	0.937 ± 0.005	0.733 ± 0.005	99.72	0.005 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=55	0.937 ± 0.005	0.733 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=60	0.937 ± 0.005	0.733 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
	nnbrs=65	0.937 ± 0.005	0.734 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.002	0.001 ± 0.000
	nnbrs=70	0.938 ± 0.005	0.734 ± 0.005	99.72	0.004 ± 0.001	0.001 ± 0.000	0.014 ± 0.001	0.001 ± 0.000
IBCF	nnbrs=5	0.947 ± 0.003	0.737 ± 0.002	**99.51**	0.041 ± 0.002	0.007 ± 0.000	0.026 ± 0.001	0.013 ± 0.001
	nnbrs=10	0.919 ± 0.003	0.718 ± 0.002	99.51	**0.042 ± 0.003**	**0.007 ± 0.001**	**0.026 ± 0.001**	**0.013 ± 0.001**
	nnbrs=15	0.913 ± 0.003	0.715 ± 0.003	99.51	0.042 ± 0.003	0.007 ± 0.001	0.026 ± 0.001	0.012 ± 0.001
	nnbrs=20	0.911 ± 0.003	**0.713 ± 0.003**	99.51	0.042 ± 0.003	0.007 ± 0.001	0.026 ± 0.001	0.012 ± 0.001
	nnbrs=25	**0.910 ± 0.002**	0.713 ± 0.002	99.51	0.041 ± 0.004	0.006 ± 0.001	0.026 ± 0.001	0.011 ± 0.001
	nnbrs=30	0.911 ± 0.002	0.714 ± 0.002	99.51	0.040 ± 0.004	0.006 ± 0.001	0.025 ± 0.002	0.011 ± 0.002
	nnbrs=35	0.911 ± 0.002	0.715 ± 0.002	99.51	0.040 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.011 ± 0.001
	nnbrs=40	0.912 ± 0.003	0.715 ± 0.002	99.51	0.040 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.011 ± 0.001
	nnbrs=45	0.912 ± 0.003	0.716 ± 0.003	99.51	0.039 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.011 ± 0.002
	nnbrs=50	0.913 ± 0.003	0.717 ± 0.003	99.51	0.039 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.010 ± 0.001
	nnbrs=55	0.913 ± 0.003	0.717 ± 0.003	99.51	0.038 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.010 ± 0.001
	nnbrs=60	0.914 ± 0.003	0.718 ± 0.003	99.51	0.038 ± 0.004	0.006 ± 0.001	0.025 ± 0.001	0.010 ± 0.001
	nnbrs=65	0.914 ± 0.003	0.718 ± 0.003	99.51	0.038 ± 0.004	0.006 ± 0.001	0.024 ± 0.001	0.010 ± 0.001
	nnbrs=70	0.914 ± 0.003	0.718 ± 0.003	99.51	0.037 ± 0.004	0.006 ± 0.001	0.024 ± 0.001	0.010 ± 0.001

Table B.1: Results of the User-based CF and Item-based CF on the cross-validation varying the maximum number of neighbors' parameter.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
FunkSVD (reg = 0.02)	nfeatures = 5	0.940 ± 0.003	0.743 ± 0.003	**99.81**	**0.069 ± 0.002**	**0.012 ± 0.001**	**0.035 ± 0.001**	**0.023 ± 0.001**
	nfeatures=10	**0.939 ± 0.003**	0.742 ± 0.003	99.81	0.068 ± 0.002	0.011 ± 0.001	0.034 ± 0.001	0.021 ± 0.001
	nfeatures=20	0.939 ± 0.004	**0.741 ± 0.003**	99.81	0.069 ± 0.002	0.010 ± 0.000	0.033 ± 0.001	0.021 ± 0.001
	nfeatures=50	0.940 ± 0.003	0.742 ± 0.003	99.81	0.043 ± 0.004	0.008 ± 0.001	0.028 ± 0.002	0.015 ± 0.001
	nfeatures=100	0.943 ± 0.004	0.745 ± 0.003	99.81	0.005 ± 0.002	0.002 ± 0.001	0.025 ± 0.003	0.002 ± 0.001
	nfeatures=200	0.945 ± 0.004	0.746 ± 0.003	99.81	0.001 ± 0.000	0.000 ± 0.000	0.011 ± 0.003	0.000 ± 0.000
	nfeatures=500	0.946 ± 0.005	0.746 ± 0.003	99.81	0.000 ± 0.000	0.000 ± 0.000	0.010 ± 0.004	0.000 ± 0.000
SVD-ALS (reg=0.1)	nfeatures=5	0.927 ± 0.001	0.726 ± 0.001	**99.81**	0.053 ± 0.002	0.009 ± 0.000	0.031 ± 0.001	0.018 ± 0.001
	nfeatures=10	0.927 ± 0.004	0.725 ± 0.003	99.81	0.057 ± 0.001	0.010 ± 0.000	0.031 ± 0.000	0.019 ± 0.000
	nfeatures=20	0.924 ± 0.004	0.724 ± 0.003	99.81	0.062 ± 0.002	0.011 ± 0.000	0.032 ± 0.001	0.021 ± 0.001
	nfeatures=50	0.917 ± 0.004	0.720 ± 0.003	99.81	0.069 ± 0.003	0.012 ± 0.000	0.034 ± 0.001	0.025 ± 0.001
	nfeatures=100	**0.915 ± 0.003**	0.719 ± 0.003	99.81	**0.071 ± 0.002**	**0.012 ± 0.000**	**0.035 ± 0.001**	**0.026 ± 0.001**
	nfeatures=200	0.915 ± 0.003	**0.718 ± 0.003**	99.81	0.071 ± 0.002	0.012 ± 0.000	0.035 ± 0.001	0.026 ± 0.001
	nfeatures=500	0.915 ± 0.003	0.718 ± 0.003	99.81	0.070 ± 0.002	0.012 ± 0.000	0.035 ± 0.001	0.026 ± 0.001
SVD-SGD (reg=0.5)	nfeatures=5	0.970 ± 0.006	0.762 ± 0.005	**99.81**	**0.022 ± 0.006**	**0.003 ± 0.001**	**0.018 ± 0.002**	**0.005 ± 0.001**
	nfeatures=10	0.969 ± 0.007	0.762 ± 0.005	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
	nfeatures=20	0.967 ± 0.006	0.760 ± 0.005	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
	nfeatures=50	**0.965 ± 0.006**	**0.759 ± 0.004**	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
	nfeatures=100	0.965 ± 0.006	0.759 ± 0.005	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
	nfeatures=200	0.965 ± 0.006	0.759 ± 0.004	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
	nfeatures=500	0.965 ± 0.006	0.759 ± 0.004	99.81	0.022 ± 0.006	0.003 ± 0.001	0.018 ± 0.002	0.005 ± 0.001
ImplicitMF (reg=0.01)	nfeatures=5	-	-	-	0.149 ± 0.003	0.045 ± 0.001	0.070 ± 0.001	0.067 ± 0.001
	nfeatures=10	-	-	-	0.136 ± 0.004	0.045 ± 0.001	0.069 ± 0.001	0.062 ± 0.001
	nfeatures=20	-	-	-	0.138 ± 0.002	0.048 ± 0.001	0.071 ± 0.001	0.063 ± 0.001
	nfeatures=50	-	-	-	0.165 ± 0.001	0.053 ± 0.002	0.077 ± 0.001	0.076 ± 0.002
	nfeatures=100	-	-	-	0.201 ± 0.001	0.057 ± 0.001	0.084 ± 0.001	0.091 ± 0.001
	nfeatures=200	-	-	-	0.231 ± 0.002	0.061 ± 0.001	0.090 ± 0.001	0.103 ± 0.002
	nfeatures=500	-	-	-	**0.252 ± 0.002**	**0.064 ± 0.002**	**0.095 ± 0.002**	**0.112 ± 0.003**

Table B.2: Results of the Matrix Factorization models on the cross-validation varying the number of features.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF	nnbics=10	0.935 ± 0.006	0.731 ± 0.007	27.85	**0.248 ± 0.010**	**0.020 ± 0.001**	**0.038 ± 0.001**	**0.053 ± 0.004**
	nnbics=20	0.917 ± 0.005	0.719 ± 0.005	35.30	0.232 ± 0.013	0.018 ± 0.001	0.035 ± 0.002	0.049 ± 0.003
	nnbics=30	0.908 ± 0.005	0.712 ± 0.004	38.95	0.218 ± 0.012	0.017 ± 0.001	0.033 ± 0.002	0.047 ± 0.002
	nnbics=40	0.904 ± 0.005	0.708 ± 0.003	41.61	0.211 ± 0.011	0.017 ± 0.001	0.032 ± 0.001	0.045 ± 0.003
	nnbics=50	0.902 ± 0.006	0.706 ± 0.004	43.55	0.203 ± 0.010	0.016 ± 0.001	0.031 ± 0.001	0.044 ± 0.002
	nnbics=60	0.899 ± 0.006	0.705 ± 0.004	44.96	0.196 ± 0.011	0.015 ± 0.001	0.030 ± 0.001	0.043 ± 0.002
	nnbics=70	**0.897 ± 0.005**	**0.703 ± 0.003**	**46.16**	0.188 ± 0.011	0.015 ± 0.001	0.029 ± 0.001	0.040 ± 0.002

Table B.3: Sensitivity of the BBCF to the number of biclusters in the neighborhood.

B.2 User-based split Cross-Validation Results

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
Bias	0.945 ± 0.028	0.745 ± 0.022	100.00	0.015 ± 0.006	0.013 ± 0.004	0.054 ± 0.008	0.008 ± 0.003
Random	-	-	-	0.013 ± 0.001	0.013 ± 0.002	0.048 ± 0.004	0.011 ± 0.001
Popular	-	-	-	0.154 ± 0.007	0.167 ± 0.005	0.158 ± 0.006	0.167 ± 0.011

Table B.4: Results of the basic models on the user-based cross-validation.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
UBCF	nnbrs=5	0.977 ± 0.034	0.763 ± 0.024	**99.79**	**0.026 ± 0.004**	**0.020 ± 0.002**	**0.063 ± 0.004**	**0.017 ± 0.003**
	nnbrs=10	0.947 ± 0.033	0.741 ± 0.024	99.79	0.019 ± 0.004	0.015 ± 0.001	0.055 ± 0.003	0.013 ± 0.002
	nnbrs=15	0.937 ± 0.033	0.733 ± 0.024	99.79	0.015 ± 0.003	0.011 ± 0.002	0.053 ± 0.005	0.009 ± 0.002
	nnbrs=20	0.934 ± 0.032	0.730 ± 0.024	99.79	0.012 ± 0.003	0.009 ± 0.002	0.051 ± 0.005	0.007 ± 0.001
	nnbrs=25	0.932 ± 0.032	0.729 ± 0.024	99.79	0.011 ± 0.002	0.008 ± 0.002	0.048 ± 0.007	0.007 ± 0.001
	nnbrs=30	**0.931 ± 0.032**	**0.728 ± 0.023**	99.79	0.009 ± 0.002	0.008 ± 0.002	0.047 ± 0.007	0.006 ± 0.001
	nnbrs=35	0.931 ± 0.032	0.728 ± 0.024	99.79	0.009 ± 0.002	0.007 ± 0.002	0.045 ± 0.005	0.006 ± 0.001
	nnbrs=40	0.931 ± 0.032	0.728 ± 0.024	99.79	0.008 ± 0.002	0.006 ± 0.001	0.044 ± 0.003	0.005 ± 0.001
	nnbrs=45	0.931 ± 0.031	0.728 ± 0.023	99.79	0.008 ± 0.002	0.006 ± 0.001	0.044 ± 0.003	0.005 ± 0.001
	nnbrs=50	0.931 ± 0.031	0.728 ± 0.023	99.79	0.007 ± 0.002	0.006 ± 0.002	0.043 ± 0.003	0.004 ± 0.001
	nnbrs=55	0.931 ± 0.031	0.728 ± 0.023	99.79	0.006 ± 0.002	0.005 ± 0.002	0.043 ± 0.004	0.004 ± 0.001
	nnbrs=60	0.932 ± 0.031	0.729 ± 0.023	99.79	0.006 ± 0.002	0.005 ± 0.001	0.042 ± 0.004	0.003 ± 0.001
	nnbrs=65	0.932 ± 0.031	0.729 ± 0.023	99.79	0.005 ± 0.002	0.004 ± 0.001	0.042 ± 0.004	0.003 ± 0.001
	nnbrs=70	0.932 ± 0.031	0.730 ± 0.023	99.79	0.005 ± 0.001	0.004 ± 0.001	0.041 ± 0.005	0.003 ± 0.001
IBCF	nnbrs=5	0.940 ± 0.029	0.731 ± 0.022	**99.65**	0.046 ± 0.006	0.038 ± 0.004	0.083 ± 0.006	0.038 ± 0.004
	nnbrs=10	0.914 ± 0.027	0.715 ± 0.020	99.65	**0.047 ± 0.009**	**0.039 ± 0.006**	**0.084 ± 0.009**	**0.042 ± 0.007**
	nnbrs=15	0.906 ± 0.028	**0.709 ± 0.021**	99.65	0.047 ± 0.008	0.038 ± 0.006	0.084 ± 0.006	0.041 ± 0.006
	nnbrs=20	**0.904 ± 0.028**	0.709 ± 0.021	99.65	0.046 ± 0.009	0.038 ± 0.007	0.081 ± 0.010	0.040 ± 0.007
	nnbrs=25	0.904 ± 0.029	0.709 ± 0.021	99.65	0.047 ± 0.010	0.038 ± 0.007	0.081 ± 0.011	0.039 ± 0.008
	nnbrs=30	0.904 ± 0.029	0.710 ± 0.021	99.65	0.046 ± 0.009	0.037 ± 0.007	0.082 ± 0.009	0.038 ± 0.007
	nnbrs=35	0.905 ± 0.029	0.710 ± 0.021	99.65	0.046 ± 0.010	0.037 ± 0.007	0.082 ± 0.010	0.037 ± 0.008
	nnbrs=40	0.906 ± 0.029	0.711 ± 0.021	99.65	0.045 ± 0.010	0.036 ± 0.008	0.080 ± 0.011	0.036 ± 0.008
	nnbrs=45	0.906 ± 0.029	0.712 ± 0.022	99.65	0.045 ± 0.010	0.036 ± 0.008	0.080 ± 0.012	0.036 ± 0.008
	nnbrs=50	0.907 ± 0.029	0.712 ± 0.022	99.65	0.043 ± 0.009	0.035 ± 0.007	0.077 ± 0.011	0.035 ± 0.007
	nnbrs=55	0.908 ± 0.029	0.713 ± 0.022	99.65	0.043 ± 0.009	0.035 ± 0.007	0.077 ± 0.012	0.034 ± 0.007
	nnbrs=60	0.908 ± 0.029	0.713 ± 0.022	99.65	0.043 ± 0.009	0.035 ± 0.007	0.078 ± 0.012	0.034 ± 0.007
	nnbrs=65	0.908 ± 0.029	0.714 ± 0.022	99.65	0.043 ± 0.009	0.035 ± 0.007	0.078 ± 0.011	0.034 ± 0.007
	nnbrs=70	0.909 ± 0.029	0.714 ± 0.022	99.65	0.042 ± 0.009	0.035 ± 0.007	0.077 ± 0.011	0.033 ± 0.007

Table B.5: Results of the User-based CF and Item-based CF on the user-based split cross-validation varying the maximum number of neighbours' parameter.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
FunkSVD (reg=0.01)	nfeatures=5	0.937 ± 0.030	0.740 ± 0.023	**99.86**	**0.072 ± 0.006**	**0.066 ± 0.004**	**0.101 ± 0.010**	0.065 ± 0.005
	nfeatures=10	0.936 ± 0.030	0.739 ± 0.023	99.86	0.072 ± 0.006	0.066 ± 0.004	0.101 ± 0.010	**0.066 ± 0.005**
	nfeatures=20	0.937 ± 0.030	0.740 ± 0.023	99.86	0.072 ± 0.006	0.065 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=50	**0.935 ± 0.031**	**0.737 ± 0.024**	99.86	0.071 ± 0.007	0.063 ± 0.005	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=100	0.936 ± 0.030	0.738 ± 0.023	99.86	0.039 ± 0.007	0.032 ± 0.005	0.072 ± 0.008	0.030 ± 0.005
	nfeatures=200	0.938 ± 0.030	0.739 ± 0.023	99.86	0.001 ± 0.000	0.001 ± 0.001	0.045 ± 0.016	0.001 ± 0.000
	nfeatures=500	0.939 ± 0.030	0.740 ± 0.022	99.86	0.000 ± 0.000	0.000 ± 0.000	0.038 ± 0.026	0.000 ± 0.000
SVD-ALS (reg=0.5)	nfeatures=5	**0.940 ± 0.029**	**0.744 ± 0.022**	**99.86**	**0.072 ± 0.007**	**0.066 ± 0.004**	**0.101 ± 0.010**	**0.065 ± 0.005**
	nfeatures=10	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=20	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=50	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=100	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=200	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
	nfeatures=500	0.940 ± 0.029	0.744 ± 0.022	99.86	0.072 ± 0.007	0.066 ± 0.004	0.101 ± 0.010	0.065 ± 0.005
SVD-SGD (reg=0.5)	nfeatures=5	0.960 ± 0.029	0.756 ± 0.024	**99.86**	0.023 ± 0.004	**0.018 ± 0.003**	**0.057 ± 0.005**	**0.014 ± 0.002**
	nfeatures=10	0.960 ± 0.029	0.756 ± 0.024	99.86	0.023 ± 0.004	0.018 ± 0.003	0.057 ± 0.005	0.014 ± 0.002
	nfeatures=20	0.958 ± 0.029	0.754 ± 0.024	99.86	0.023 ± 0.005	0.018 ± 0.003	0.057 ± 0.005	0.014 ± 0.002
	nfeatures=50	**0.957 ± 0.029**	**0.753 ± 0.024**	99.86	0.023 ± 0.005	0.018 ± 0.004	0.057 ± 0.005	0.014 ± 0.002
	nfeatures=100	0.957 ± 0.029	0.753 ± 0.024	99.86	**0.024 ± 0.005**	0.018 ± 0.004	0.057 ± 0.004	0.014 ± 0.002
	nfeatures=200	0.957 ± 0.029	0.753 ± 0.024	99.86	0.023 ± 0.005	0.018 ± 0.003	0.057 ± 0.004	0.014 ± 0.002
	nfeatures=500	0.957 ± 0.029	0.753 ± 0.024	99.86	0.023 ± 0.005	0.018 ± 0.003	0.056 ± 0.005	0.014 ± 0.002
ImplicitMF (reg=0.01)	nfeatures=5	-	-	-	0.152 ± 0.013	0.227 ± 0.009	0.175 ± 0.007	0.183 ± 0.001
	nfeatures=10	-	-	-	0.145 ± 0.013	0.231 ± 0.020	0.168 ± 0.013	0.175 ± 0.013
	nfeatures=20	-	-	-	0.144 ± 0.005	0.245 ± 0.019	0.171 ± 0.006	0.174 ± 0.010
	nfeatures=50	-	-	-	0.170 ± 0.008	0.276 ± 0.016	0.190 ± 0.004	0.210 ± 0.010
	nfeatures=100	-	-	-	0.206 ± 0.014	0.302 ± 0.014	0.216 ± 0.007	0.252 ± 0.010
	nfeatures=200	-	-	-	0.237 ± 0.017	0.319 ± 0.012	0.236 ± 0.009	0.289 ± 0.012
	nfeatures=500	-	-	-	**0.255 ± 0.017**	**0.327 ± 0.012**	**0.248 ± 0.009**	**0.307 ± 0.012**

Table B.6: Results of the Matrix Factorization models on the user-based cross-validation varying the number of features.

Model		Prediction			Recommendation			
		RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF	nnbics=10	0.908 ± 0.059	0.705 ± 0.038	16.71	**0.342 ± 0.033**	**0.124 ± 0.012**	**0.183 ± 0.021**	**0.206 ± 0.021**
	nnbics=20	0.897 ± 0.046	0.701 ± 0.029	24.61	0.307 ± 0.036	0.117 ± 0.012	0.168 ± 0.017	0.190 ± 0.023
	nnbics=30	0.888 ± 0.039	0.694 ± 0.027	30.02	0.279 ± 0.034	0.111 ± 0.010	0.158 ± 0.014	0.179 ± 0.018
	nnbics=40	**0.880 ± 0.036**	**0.687 ± 0.024**	34.14	0.253 ± 0.026	0.104 ± 0.007	0.146 ± 0.012	0.171 ± 0.015
	nnbics=50	0.884 ± 0.030	0.690 ± 0.018	37.47	0.241 ± 0.030	0.102 ± 0.009	0.142 ± 0.015	0.164 ± 0.016
	nnbics=60	0.886 ± 0.035	0.692 ± 0.022	40.24	0.230 ± 0.035	0.099 ± 0.012	0.138 ± 0.016	0.163 ± 0.017
	nnbics=70	0.882 ± 0.034	0.688 ± 0.022	**43.22**	0.224 ± 0.033	0.098 ± 0.012	0.140 ± 0.018	0.158 ± 0.017

Table B.7: Results of the BBCF on the user-based cross-validation varying the number of nearest biclusters' parameter.

Model	Prediction			Recommendation			
	RMSE	MAE	Coverage (%)	Precision	Recall	F1-Score	nDCG
BBCF-NoWeight (nnbic=40)	0.895 ± 0.033	0.700 ± 0.021	46.90	0.242 ± 0.030	0.108 ± 0.009	0.150 ± 0.013	0.168 ± 0.015

Table B.8: Results of the BBCF-NoWeight model on the user-based cross-validation.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *TCF-ContextAwareRC-Adomavicius*, RecSys '08, pages 335–336, New York, NY, USA, 2008. ACM.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998.

[3] Faris Alqadah and Raj Bhatnagar. Similarity measures in formal concept analysis. *Ann. Math. Artif. Intell.*, 61:245–256, 01 2011.

[4] Faris Alqadah, Chandan K. Reddy, Junling Hu, and Hatim F. Alqadah. Biclustering neighborhood-based collaborative filtering method for top-n recommender systems. *Knowledge and Information Systems*, 44(2):475–491, Aug 2015.

[5] Cigdem Bakir. Collaborative filtering with temporal dynamics with using singular value decomposition. *Tehnicki Vjesnik*, 25:130–135, 02 2018.

[6] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. *Proceedings of the Third ACM Conference on Recommender Systems*, 01 2009.

[7] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, pages 25–30. Citeseer, 2009.

[8] Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. A generalized maximum entropy approach to bregman coclustering and matrix approximation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 509–514, New York, NY, USA, 2004. Association for Computing Machinery.

[9] Simon Barkow-Oesterreicher, Stefan Bleuler, Amela Prelic, Philip Zimmermann, and Eckart Zitzler. Bicat: A biclustering analysis toolbox. *Bioinformatics (Oxford, England)*, 22:1282–3, 06 2006.

[10] Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, 2007.

[11] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6, New York, August 2007. ACM.

[12] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, pages 591–596, 2011.

[13] Doruk Bozdağ, Ashwin S. Kumar, and Umit V. Catalyurek. Comparative analysis of biclustering algorithms. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, BCB '10, page 265–274, New York, NY, USA, 2010. Association for Computing Machinery.

[14] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

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

[16] Sara C . Madeira, Miguel Cacho Teixeira, Isabel Sá-Correia, and Arlindo Oliveira. Identification of regulatory modules in time series gene expression data using a linear time biclustering algorithm. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 7:153–65, 04 2010.

[17] R.J.G.B. Campello. Generalized external indexes for comparing data partitions with overlapping categories. *Pattern Recognition Letters*, 31(9):966 – 975, 2010.

[18] Yizong Cheng and George M Church. Biclustering of expression data. In *Ismb*, volume 8, pages 93–103, 2000.

[19] Papers With Code. Papers with code - movielens 100k benchmark (recommendation systems). Accessed: 2020-11-05.

[20] Guilherme Palermo Coelho, Fabrício Olivetti de França, and Fernando J. Von Zuben. A multi-objective multipopulation approach for biclustering. In Peter J. Bentley, Doheon Lee, and Sungwon Jung, editors, *Artificial Immune Systems*, pages 71–82, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[21] Fabricio Olivetti De França, Guilherme Palermo Coelho, and Fernando J Von Zuben. Coherent recommendations using biclustering. In *Proc. of the XXX Congresso Ibero-Latino-Americano de Métodos Computacionais em Engenharia (CILAMCE)*, pages 1–15, 2009.

[22] Inderjit Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 05 2001.

[23] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 89–98, New York, NY, USA, 2003. ACM.

[24] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, page 485–492, New York, NY, USA, 2005. Association for Computing Machinery.

[25] Yi Ding, Xue Li, and Maria E Orlowska. Recency-based collaborative filtering. In *Proceedings of the 17th Australasian Database Conference-Volume 49*, volume 49, pages 99–107, 2006.

[26] Michael D. Ekstrand. The LKPY package for recommender systems experiments: Next-generation tools and lessons learned from the lenskit project. *CoRR*, abs/1809.03125, 2018.

[27] Michael D. Ekstrand, Michael Ludwig, Joseph A. Konstan, and John T. Riedl. Rethinking the recommender research ecosystem: Reproducibility, openness, and lenskit. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, page 133–140, New York, NY, USA, 2011. Association for Computing Machinery.

[28] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[29] Islam Elnabarawy, Donald C. Wunsch, and Ashraf M. Abdelbar. Biclustering artmap collaborative filtering recommender system. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2986–2991, Vancouver, BC, Canada, 2016. IEEE Computer Society.

[30] Kemal Eren, Mehmet Deveci, Onur Küçüktunç, and Umit Catalyurek. A comparative analysis of biclustering algorithms for gene expression data. *Briefings in bioinformatics*, 14, 07 2012.

[31] Liang Feng, Qianchuan Zhao, and Cangqi Zhou. Improving performances of top-n recommendations with co-clustering method. *Expert Systems with Applications*, 143:113078, 2020.

[32] Simon Funk. Netflix update: Try this at home. https://sifter.org/ simon/journal/20061211.html, 2006.

[33] Bernhard Ganter and Rudolf Wille. *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.

[34] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4 pp.–, Nov 2005.

[35] Jennifer Ann Golbeck. *Computing and Applying Trust in Web-based Social Networks*. PhD thesis, University of Maryland at College Park, College Park, MD, USA, 2005. AAI3178583.

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

[37] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, Jul 2001.

[38] Songjie Gong. A collaborative filtering recommendation algorithm based on user clustering and item clustering. *JSW*, 5:745–752, 07 2010.

[39] Stephen Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1 – 47, 2013. Twenty-fifth Anniversay Commemorative Issue.

[40] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, December 2009.

[41] Neelima Gupta and Seema Aggarwal. Mib: Using mutual information for biclustering gene expression data. *Pattern Recognition*, 43(8):2692 – 2697, 2010.

[42] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[43] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[44] J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

[45] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for collaborative filtering and data visualization. *arXiv preprint arXiv:1301.3862*, 2013.

[46] Rui Henriques. *Learning from high-dimensional data using local descriptive models*. PhD thesis, PhD thesis, Instituto Superior Tecnico, Universidade de Lisboa, Lisboa, 2016.

[47] Rui Henriques, Cláudia Antunes, and Sara C. Madeira. A structured view on pattern mining-based biclustering. *Pattern Recognition*, 48(12):3941 – 3958, 2015.

[48] Rui Henriques and Sara C . Madeira. Bicpam: Pattern-based biclustering for biomedical data analysis. *Algorithms for Molecular Biology*, 9:27, 12 2014.

[49] Rui Henriques, Francisco Ferreira, and Sara C . Madeira. Bicpams: Software for biological data analysis with pattern-based biclustering. *BMC Bioinformatics*, 18, 02 2017.

[50] Rui Henriques and Sara C Madeira. Bicnet: Flexible module discovery in large-scale biological networks using biclustering. *Algorithms for Molecular Biology*, 11(1):14, 2016.

[51] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[52] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004.

[53] D. Horta and R. J. G. B. Campello. Similarity measures for comparing biclusterings. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(5):942–954, 2014.

[54] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008.

[55] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[56] Joel Irish. The mean measure of divergence: Its utility in model-free and model-bound analyses relative to the mahalanobis d-2 distance for nonmetric traits. *American journal of human biology : the official journal of the Human Biology Council*, 22:378–95, 05 2010.

[57] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[58] Nyoman Juniarta. *Mining complex data and biclustering using formal concept analysis*. PhD thesis, Université de Lorraine, 2019.

[59] Kai Yu, A. Schwaighofer, V. Tresp, Xiaowei Xu, and H. . Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):56–69, Jan 2004.

[60] Surya Kant and Tripti Mahara. Nearest biclusters collaborative filtering framework with fusion. *Journal of Computational Science*, 25:204 – 212, 2018.

[61] Adetayo Kasim, Ziv Shkedy, Sebastian Kaiser, Sepp Hochreiter, and Willem Talloen. *Applied biclustering methods for big and high-dimensional data using R*. CRC Press, 10 2016.

[62] Jon Kleinberg and Mark Sandler. Using mixture models for collaborative filtering. *Journal of Computer and System Sciences*, 74(1):49 – 69, 2008. Learning Theory 2004.

[63] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[64] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[65] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009.

[66] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.

[67] Neal Lathia, Stephen Hailes, and Licia Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 796–797, 2009.

[68] Bin Li, Qiang Yang, and Xiangyang Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 617–624, New York, NY, USA, 2009. ACM.

[69] Guojun Li, Qin Ma, Haibao Tang, Andrew H. Paterson, and Ying Xu. QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic Acids Research*, 37(15):e101–e101, 06 2009.

[70] Changyong Liang and Yajun Leng. Collaborative filtering based on information-theoretic co-clustering. *International Journal of Systems Science*, 45(3):589–597, 2014.

[71] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7:76–80, 01 2003.

[72] Nathan N. Liu, Bin Cao, Min Zhao, and Qiang Yang. Adapting neighborhood and matrix factorization models for context aware recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 7–13, New York, NY, USA, 2010. ACM.

[73] Nathan N Liu, Min Zhao, Evan Xiang, and Qiang Yang. Online evolutionary collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 95–102, 2010.

[74] Xiangyu Liu, Di Li, Juntao Liu, Zhengchang Su, and Guojun Li. RecBic: a fast and accurate algorithm recognizing trend-preserving biclusters. *Bioinformatics*, 07 2020. btaa630.

[75] Joao Lobo, Rui Henriques, and Sara C Madeira. G-tric: generating three-way synthetic datasets with triclustering solutions. *BMC Bioinformatics*, 2021.

[76] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, Jan 2004.

[77] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 17–24, New York, NY, USA, 2007. ACM.

[78] Arnd Kohrs-Bernard Merialdo. Clustering for collaborative filtering applications. *Intelligent Image Processing, Data Analysis & Information Retrieval*, 3:199, 1999.

[79] Abdul Mohd, Mohd Hameed, Omar al jadaan, and Ramachandram Sirandas. Collaborative filtering based recommendation system: A survey. *International Journal on Computer Science and Engineering*, 4, 05 2012.

[80] T Murali and Simon Kasif. Extracting conserved gene expression motifs from gene expression data. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 8:77–88, 02 2003.

[81] T Murali and Simon Kasif. Extracting conserved gene expression motifs from gene expression data. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 8:77–88, 02 2003.

[82] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, page 497–506, USA, 2011. IEEE Computer Society.

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

[84] Victor A Padilha and Ricardo J G B Campello. A systematic comparative evaluation of biclustering techniques. *BMC Bioinformatics*, 18(1):55, 2017.

[85] Gaurav Pandey, Gowtham Atluri, Michael Steinbach, Chad L. Myers, and Vipin Kumar. An association analysis approach to biclustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 677–686, New York, NY, USA, 2009. Association for Computing Machinery.

[86] A. Patrikainen and M. Meila. Comparing subspace clusterings. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):902–916, 2006.

[87] David M. Pennock, Eric J. Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach, 2013.

[88] Beatriz Pontes, Ral Girldez, and Jess S Aguilar-Ruiz. Quality measures for gene expression biclusters. *PloS one*, 10(3):e0115497, 2015.

[89] Amela Prelić, Stefan Bleuler, Philip Zimmermann, Anja Wille, Peter Bühlmann, Wilhelm Gruissem, Lars Hennig, Lothar Thiele, and Eckart Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, May 2006.

[90] Ahmed Rashed, Josif Grabocka, and Lars Schmidt-Thieme. Attribute-aware non-linear co-embeddings of graph features. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 314–321, 2019.

[91] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *arXiv preprint arXiv:1905.01395*, 2019.

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

[93] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[94] Aghiles Salah, Nicoleta Rogovschi, and Mohamed Nadif. A dynamic collaborative filtering system via a weighted clustering approach. *Neurocomputing*, 175:206 – 215, 2016.

[95] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[96] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[97] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, pages 291–324, 2002.

[98] Akdes Serin and Martin Vingron. DeBi: Discovering Differentially Expressed Biclusters using a Frequent Itemset Approach. *Algorithms for Molecular Biology*, 6(1):18, 2011.

[99] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1):3:1–3:45, May 2014.

[100] Kelvin Sim, Vivekanand Gopalkrishnan, Arthur Zimek, and Gao Cong. A survey on enhanced subspace clustering. *Data Mining and Knowledge Discovery*, 26(2):332–397, Mar 2013.

[101] Monika Singh and Monica Mehrotra. Impact of biclustering on the performance of biclustering based collaborative filtering. *Expert Systems with Applications*, 113, 06 2018.

[102] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[103] Suryakant and Tripti Mahara. A new similarity measure based on mean measure of divergence for collaborative filtering in sparse environment. *Procedia Computer Science*, 89:450 – 456, 2016. Twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.

[104] Panagiotis Symeonidis, Alexandros Nanopoulos, Apostolos Papadopoulos, and Yannis Manolopoulos. Nearest-biclusters collaborative filtering with constant values. In Olfa Nasraoui, Myra Spiliopoulou, Jaideep Srivastava, Bamshad Mobasher, and Brij Masand, editors, *Advances in Web Mining and Web Usage Analysis*, pages 36–55, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[105] Panagiotis Symeonidis, Alexandros Nanopoulos, Apostolos Papadopoulos, and Yannis Manolopoulos. Nearest-biclusters collaborative filtering based on constant and coherent values. *Inf. Retr.*, 11:51–75, 02 2008.

[106] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proceedings of the 2Nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 6:1–6:8, New York, NY, USA, 2008. ACM.

[107] Amos Tanay, Roded Sharan, and Ron Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics (Oxford, England)*, 18 Suppl 1:S136–44, 02 2002.

[108] João Vinagre and Alípio Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18, 11 2012.

[109] João Vinagre, Alípio Jorge, and João Gama. Evaluation of recommender systems in streaming environments. *CoRR*, abs/1504.08175, 2015.

[110] João Vinagre, Alípio Mário Jorge, and João Gama. An overview on the exploitation of time in collaborative filtering. *WIREs Data Mining and Knowledge Discovery*, 5(5):195–215, 2015.

[111] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 501–508, New York, NY, USA, 2006. ACM.

[112] Juan Xie, Anjun Ma, Yu Zhang, Bingqiang Liu, Sha Cao, Cankun Wang, Jennifer Xu, Chi Zhang, and Qin Ma. QUBIC2: a novel and robust biclustering algorithm for analyses and interpretation of large-scale RNA-Seq data. *Bioinformatics*, 36(4):1143–1149, 09 2019.

[113] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010.

[114] Rui Xu and Donald Wunsch. Bartmap: a viable structure for biclustering. *Neural networks : the official journal of the International Neural Network Society*, 24:709–16, 04 2011.

[115] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 114–121, New York, NY, USA, 2005. ACM.

[116] Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. A survey of collaborative filtering based social recommender systems. *Computer Communications*, 41:1 – 10, 2014.

[117] Kevin Y Yip, David W Cheung, and Michael K Ng. Harp: A practical projected clustering algorithm. *IEEE Transactions on knowledge and data engineering*, 16(11):1387–1397, 2004.

[118] Mehmet Türkay Yoldar and Uğur Özcan. Collaborative targeting: Biclustering-based online ad recommendation. *Electronic Commerce Research and Applications*, 35:100857, 2019.

[119] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. Time-aware point-of-interest recommendation. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, page 363–372, New York, NY, USA, 2013. Association for Computing Machinery.

[120] Mohammed J. Zaki. *Closed Itemset Mining and Non-redundant Association Rule Mining*, pages 365–368. Springer US, Boston, MA, 2009.

[121] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In Rudolf Fleischer and Jinhui Xu, editors, *Algorithmic Aspects in Information and Management*, pages 337–348, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[122] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

[123] F. Zuben, H. M. Ferreira, F. França, and P. Castro. Applying biclustering to perform collaborative filtering. In *2007 7th International Conference on Intelligent Systems Design and Applications*, pages 421–426, Los Alamitos, CA, USA, oct 2007. IEEE Computer Society.