UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Pre-processing approaches for collaborative filtering based on hierarchical clustering

Fernando Soares de Aguiar Neto

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Fernando Soares de Aguiar Neto

Pre-processing approaches for collaborative filtering based on hierarchical clustering

Master dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Marcelo Garcia Manzato
Co-advisor: Prof. Dr. Ricardo José Gabrielli Barreto Campello

USP – São Carlos
September 2018

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

d284a

de Aguiar Neto, Fernando Soares
Abordagens de pré-processamento para filtragem colaborativa baseada em agrupamento hierárquico / Fernando Soares de Aguiar Neto; orientador Marcelo Garcia Manzato; coorientador Ricardo J. G. B. Campello. -- São Carlos, 2018.
64 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2018.

1. Sistemas de Recomendação. 2. Agrupamento de Dados. 3. Otimização. I. Garcia Manzato, Marcelo, orient. II. J. G. B. Campello, Ricardo, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Fernando Soares de Aguiar Neto

Abordagens de pré-processamento para filtragem colaborativa baseada em agrupamento hierárquico

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Marcelo Garcia Manzato
Coorientador: Prof. Dr. Ricardo José Gabrielli Barreto Campello

USP – São Carlos
Setembro de 2018

ACKNOWLEDGEMENTS

To my family that always supported me. To all professors that helped me at the development of this work, providing insight and knowledge. To my laboratory colleagues that provided valuable discussions and feedback. To my Advisor and Co-Advisor, both were crucial to the final state of this work.

Also, I would like to thank CNPq[1] and FAPESP[2] for the financial support, also thank CeMEAI for the computational aid given to our experiments by the mean of Euler-Cluster[3].

[1] Grant #132633/2016-7
[2] Grant #2016/04798-5
[3] Funded by FAPESP, grant #2013/07375-0

RESUMO

AGUIAR NETO, F. S. **Abordagens de pré-processamento para filtragem colaborativa baseada em agrupamento hierárquico**. 2018. 64 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

Sistemas de Recomendação auxiliam usuários a encontrar conteúdo relevante, como filmes, livros, músicas entre outros produtos baseando-se em suas preferências. Tais preferências são obtidas ao analisar interações passadas dos usuários, no entanto, dados coletados com esse propósito tendem a tipicamente possuir alta dimensionalidade e esparsidade. Técnicas baseadas em agrupamento de dados têm sido propostas para lidar com esses problemas de foma eficiente e eficaz ao dividir os dados em grupos similares baseando-se em características pré-definidas. Ainda que essas técnicas tenham recebido atenção crescente na comunidade de sistemas de recomendação, tais técnicas são usualmente atreladas a um algoritmo de recomendação específico e/ou requerem parâmetros críticos, como número de grupos. Neste trabalho, apresentamos três variantes de um método de propósitvo geral de extração ótima de grupos em uma hierarquia, atacando especificamente problemas em Sistemas de Recomendação. Os métodos de extração propostos não requerem parâmetros críticos e podem ser aplicados antes de qualquer sistema de recomendação. Os experimentos mostraram resultados promissores no contexto de nove bases de dados públicas conhecidas em diferentes domínios.

Palavras-chave: Sistemas de Recomendação, Agrupamento de Dados, Otimização, Dissertação.

ABSTRACT

AGUIAR NETO, F. S. **Pre-processing approaches for collaborative filtering based on hierarchical clustering**. 2018. 64 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

Recommender Systems (RS) support users to find relevant content, such as movies, books, songs, and other products based on their preferences. Such preferences are gathered by analyzing past users' interactions, however, data collected for this purpose are typically prone to sparsity and high dimensionality. Clustering-based techniques have been proposed to handle these problems effectively and efficiently by segmenting the data into a number of similar groups based on predefined characteristics. Although these techniques have gained increasing attention in the recommender systems community, they are usually bound to a particular recommender system and/or require critical parameters, such as the number of clusters. In this work, we present three variants of a general-purpose method to optimally extract users' groups from a hierarchical clustering algorithm specifically targeting RS problems. The proposed extraction methods do not require critical parameters and can be applied prior to any recommendation system. Our experiments have shown promising recommendation results in the context of nine well-known public datasets from different domains.

Keywords: Recommender Systems, Clustering, Optimization, Dissertation.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CBF	Content-Based Filtering
CF	Collaborative Filtering
FOSC	Framework for Optimal Selection of Clusters from hierarchies
HF	Hybrid Filtering
RS	Recommender Systems

CONTENTS

CHAPTER

1

INTRODUCTION

This work presents techniques that aid Recommender Systems. This chapter presents the context and motivation that lead to the development of this research, along with the presented work main contributions.

1.1 Contextualization

Recommender Systems (RS) are algorithms that filter relevant content for users, using data gathered from their personal interests and previous feedback. The need for such systems has increased with the growing amount of information generated with globalization and Internet (RICCI *et al.*, 2011).

RS can be classified into two categories, depending on their goal: i) rating prediction, in which the system predicts a score for each user-item pair; and ii) item recommendation, in which a ranking of relevant items is retrieved to the user (AGGARWAL, 2016). Both tasks are aided by filtering techniques, such as Collaborative Filtering (CF), Content-Based Filtering (CBF) and Hybrid Filtering (HF). While CF uses previous interactions of similar users or items to compute recommendations, CBF uses characteristics extracted from items to construct a users' preference profile and achieve recommendations based on such information. Hybrid methods, in turn, mix CF and CBF into a single approach in order to mitigate the weaknesses of each individual technique and improve overall performance by combining their strengths (RICCI *et al.*, 2011; BOBADILLA *et al.*, 2013; BURKE, 2007).

1.2 Motivation

Regardless of the filtering technique adopted, RS face challenges such as sparsity and high dimensionality of data (AGGARWAL, 2016). Sparsity means that much of the information about the users' preferences is missing, given that users generally do not interact with most

available items, e.g. usually one does not listen or provide feedback to most songs from a radio. High dimensionality, on the other hand, occurs because there are too many attributes to describe users, assuming users are described by their interactions with each item, and typically there is a large number of items (RICCI *et al.*, 2011).

Recently, clustering techniques have gained increasing attention as an approach to reduce high dimensionality and sparsity of data, improving the quality of recommendations (NAJAFABADI *et al.*, 2017; COSTA; MANZATO; CAMPELLO, 2016; PEREIRA; HRUSCHKA, 2015; ZAHRA *et al.*, 2015; VLACHOS *et al.*, 2014; CONNOR; HERLOCKER, 2001; UNGAR; FOSTER, 1998). Clustering makes it possible to group users and/or items with similar characteristics, discovering topics of interest while possibly also detecting and removing noise (BILGE; POLAT, 2013; LI; KIM, 2003). However, many techniques proposed in the literature are bound to their own recommendation strategy, i.e. the information extracted during the clustering step is useful only for a specific recommendation algorithm. This way, the clustering step cannot be performed as a general-purpose pre-processing step regardless of the (possibly multiple, different) RS that will be applied later. Besides, these approaches usually tackle problems such as sparsity and high dimensionality in an indirect way only, by reducing the number of users/items (COSTA; MANZATO; CAMPELLO, 2016; CONNOR; HERLOCKER, 2001) or extracting information gathered on the clustering step (NAJAFABADI *et al.*, 2017; LI; KIM, 2003). There is no guidance to actively mitigate these problems other than somehow grouping users or items. In addition, some algorithms are highly susceptible to parametrization (SUNANDA; VINEELA, 2015; SHINDE; KULKARNI, 2012), and/or require enriched information that may not be available for every application domain (COSTA; MANZATO; CAMPELLO, 2016; NAJAFABADI *et al.*, 2017). Finally, many of these methods do not take into consideration that an RS usually demands a minimum amount of information in order to work properly, especially CF approaches.

1.3 Objectives

The main objective of this research is to explore the impact of hierarchical clustering techniques applied prior to the recommendation. In particular, we aim to understand how these techniques can be used to improve the quality of recommendations by segregating dissimilar users in many groups. Therefore, our research hypothesis is that by generating user's profile groups at special abstraction levels and apply recommender algorithms independently in each of these groups can generate better recommendations than the same recommender algorithm using the whole database.

1.4 Contributions

In order to fill the gaps discussed and attain our objectives, we propose a pre-processing step for recommender systems that can be applied prior to any recommender algorithm, cate-

gorizing the users into disjoint groups or clusters. We propose three variants of a method that automatically provides an optimal partition from a hierarchy of clusters given some optimization criteria: two are based on the notion of cluster lifetime (BAGLA, 2006; CAMPELLO *et al.*, 2013), leading to a partition with more stable clusters; and the other explicitly minimizes the sparsity of the interactions within clusters, directly tacking the sparsity problem that is common in RS. All strategies ensure that there is a minimum number of users in each resulting group, in order to provide a minimum amount of information needed for any recommender that may be applied subsequently. The proposed methods in principle require very little information to operate, and they can be extended in order to use more information, when available. Last but not the least, they require no critical parameter.

In summary, the main contributions of this research are:

- A pre-processing clustering technique that: (i) relies only on information widely available in databases, e.g. history logs and ratings, but can be easily extended to use more complex information when available, and (ii) can be applied prior to any recommender algorithm, reducing the amount of information to be processed by RS;
- Three variants to optimally extract groups from a hierarchy of clusters that specifically target RS problems; two based on the notion of cluster lifetime (BAGLA, 2006; CAMPELLO *et al.*, 2013), and the other based on the sparsity of the interactions within a group;
- An optimization solution for the problem of cluster extraction from a clustering hierarchy that ensures a minimum number of users in each extracted cluster and can be adapted to operate with different clustering quality criteria, without requiring any critical parameter;
- A cluster quality criterion explicitly based on the sparsity of the data, which depends only on internal information of a cluster and can thus be pre-computed independently for each cluster in a collection of candidates.

1.5 Dissertation Organization

The remainder of dissertation is organized as follows: in Chapter 2 we discuss the background related to the presented work, concepts both from RS and clustering, including the extraction technique which is base to this work, besides related works on clustering techniques to support recommendation; in Chapter 3 the proposed cluster extraction methods are explained in detail; Chapter 4 presents the experimental evaluation, methodology and discussion of the results; lastly, in Chapter 5 we address final remarks, summarize the work and its contributions, also discuss perspectives for future work.

CHAPTER

2

LITERATURE REVIEW

In this chapter are presented the main concepts related to this work, important for a better understanding of the presented research, starting with a Recommender Systems overview, then Clustering and FOSC extraction technique, ending with works that use clustering to aid recommendation task.

2.1 Notation

In the context of recommendation algorithms, we use special letters to denote users, u and v, whereas for items we use letters i and j. The sets of users and items are denoted by U and I, respectively, whereas $|U|$ and $|I|$ denote their cardinality (numbers of users and items). For recommendation, we use r_{ui} to denote a known rating or score of a user-item interaction (between a user u and an item i), while $\hat{r}_{ui}$ is used to refer to a predicted one.

For clustering, we use C to refer to a cluster of data objects, $|C|$ is its cardinality (cluster size), and C_n is the cluster of index n. A hierarchy of clusters can be expressed as a binary-tree, where each node represents a cluster, the root-node, C_1, represents the cluster that contains all objects, and each internal node C_n have a left child, denoted by C_n^l, and a right child, denoted by C_n^r. Nodes without descendants are called external nodes or leaves.

2.2 Recommender Systems

As introduced before, Recommender Systems are algorithms that filter relevant content for users, using data gathered from their personal interests and previous feedback. They can be classified in many categories, according to the desired output (rating prediction or ranking), the filtering technique (e.g. Collaborative Filtering) and the approach used to aggregate the information (e.g. neighborhood-based) (AGGARWAL, 2016; BOBADILLA *et al.*, 2013).

For the desired output, two main tasks are attributed to RS, rating prediction and ranking (also referred as item recommendation). Rating prediction intents to predict the specific rating a user would give to each item on the system; on the other hand, item recommendation aims to rank items in order of preference for each user. Independent of the task, RS are aided by filtering techniques which help to select the desired information in order to predict the recommendations.

2.2.1 Recommender Systems Filtering

The literature presents many filtering techniques: some use item's description (Content Based Filtering); others explore the relations and similarities between users or items (Collaborative Filtering); some explore other information, e.g. demographic or context (Community and Context Based Filterings); and finally, it is possible to mix any other filtering techniques in order to achieve a Hybrid Filtering approach(BURKE, 2007).

Each filtering approach has its own advantages and limitations, being Hybrid Filtering and Collaborative Filtering very common, Content Based Filtering is also widely used, although item information might not be available for every domain. Collaborative Filtering uses similarities between users to achieve recommendations, it is based on the idea that similar people will have similar interests, therefore it is important to define a representation for each user.

2.2.2 Users' Profile

Each filtering technique may ask for specific information about each user, for instance Community-Based Filtering may ask for demographic information about a user. For CF it is used information about the preferences of each user. Such representation is usually called Users' Profile.

The needed information can be acquired mainly in two ways: by implicit interactions, where the users do not feel they are providing information; or explicit interactions, where the user actively provides information to the system. Examples of implicit interactions are: transaction history (JAWAHEER; SZOMSZOR; KOSTKOVA, 2010); listened songs; or web site access (BOBADILLA *et al.*, 2013). Implicit interactions are generated constantly by the user by simply using the system, hence there is no effort to generate them. However, even abundant, implicit interactions tend to lead to imprecise information (AMATRIAIN; PUJOL; OLIVER, 2009). On the other hand, explicit interactions require the user to actively give his opinion about some product, which usually happens only on extremes, either the user really likes or dislikes the item, concentrating this type of feedback on the edges; besides, explicit information tend to be less abundant (AMATRIAIN; PUJOL; OLIVER, 2009).

Usually, users' profiles are stored and manipulated by the use of a matrix, associating each pair user-item to a value of interest, e.g. rating. However, such approach offers little information and is prone to traditional problems in RS, as for huge sparsity of information and

high data dimensionality. In this context, much research has explored ways to enrich users' profiles, sometimes simply by acquiring more information. For instance in (ADOMAVICIUS *et al.*, 2011) the authors explore the use of context information, i.e. each transaction is associated to the occasion it occurred. Another example, in (NAJAFABADI *et al.*, 2017), the authors enrich information about items, and use clustering and association rules combined to achieve a better representation of the user and therefore a better recommendation.

The information by itself is not sufficient to generate good recommendations, an algorithm to process all information and then generate recommendations is used, and many approaches are being explored.

2.2.3 Recommender Systems Algorithms

Recommender Systems need algorithms to process the information and generate recommendations. Each filtering technique has its algorithms and approaches, given that each filtering attacks the recommendation task in a different way. CF has been widely explored (BOBADILLA *et al.*, 2013; RICCI *et al.*, 2011) and will be explored in this work, motivated because it needs little information about users or items, being easy to implement in every domain. Therefore, this subsection explains some approaches used in CF also used in the evaluation of this work. Also it is worth to notice that some approaches described here can be adapted to work with other filtering techniques and incorporate more information; also, the solution described in this work is not limited to CF.

There are three main approaches to CF: Neighborhood-based algorithms that explore the ratings given by a subset of similar users; Latent Factor approaches, which assume that there are latent factors that describe the item and the users preferences; and also global recommendations, which provide a non-personalized recommendation for every user. Each described in more detail below, alongside with one example algorithm which was applied to this work:

- **Neighborhood-based:** Selects a subset of the data which are more similar to a user or item, its so-called neighborhood, then, only the information pertinent to the neighborhood is used to generate the recommendations. One example is the User-Knn algorithm (KOREN, 2010), the prediction of an unknown score for a user-item pair $\hat{r}_{ui}$ is made by checking if the k most similar users to u interacted with item i. We denote $It_{vi} = 1$ if user v interacted with item i, and 0 otherwise. The neighborhood is generated using some similarity function between users, denoted $sim(u,v)$. Finally, it predicts the score $\hat{r}_{ui}$ considering the neighborhood $N(u)$ and their interactions, as follows:

$$\hat{r}_{ui} = \sum_{v \in N(u)} It_{vi} sim(u,v) \tag{2.1}$$

User-Knn then sorts the scores $\hat{r}_{u.}$ to generate a ranking of items to be recommended to user u.

- **Latent Factor or Matrix Factorization:** Assumes that the user-item interactions matrix $R : U \times I$, is incomplete but is a product of two lower rank matrices, hence $\hat{R} = WH^t$, where $W : |U| \times k$ and $H : |I| \times k$, being k the dimensionality/rank of the approximation, and each row W_u or H_i can be interpreted as users or items described by latent factors, that exist but are unobserved. Therefore, a prediction using MF is defined as in Equation (2.2).

$$\hat{r}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if} \tag{2.2}$$

  One example of algorithm that is latent factor based is BPRMF, which stands for Bayesian Personalized Ranking Matrix Factorization (RENDLE *et al.*, 2012). It consists in optimizing a ranking (BPR) of items for each user by the use of implicit information, i.e. a user interacted or not with a given item, which helps to infer the relative order of preference between two items. It can be used to optimize different recommender models, such as Matrix Factorization (MF).

- **Non-Personalized:** Is the simplest of all approaches, simply recommending based on a global view of each item. For instance, the Most Popular algorithm recommends the items by its global popularity, independent of each user's interactions (ADOMAVICIUS *et al.*, 2016). To recommend item i to a user u, this method counts all interactions that item i received from users in the dataset, $\hat{r}_{ui} = |T_i|$, where T_i is the set of interactions involving item i. The predicted scores $\hat{r}_{ui}$ for those items i unknown to user u (i.e., for which the entries r_{ui} in the user-item matrix are unknown) are sorted in decreasing order, and the top items are recommended to that user.

2.3 Clustering

Clustering methods categorize a dataset into clusters, in such a way that similar data objects tend to belong to the same cluster whereas objects that are not similar to each other tend to belong to different clusters (GAN; MA; WU, 2007).

2.3.1 Hierarchical Clustering

Clustering algorithms can be mainly divided into partitional and hierarchical. Partitional algorithms aim to find a "flat" clustering solution consisting of a given number of mutually exclusive clusters, usually a *partition* of the data into disjoint subsets. Depending on the algorithm, the number of clusters may be arbitrarily set by the user or automatically determined, oftentimes as a consequence of some other user-defined parameter. Hierarchical algorithms, on the other

hand, produce a complete collection of nested partitions with all possible numbers of clusters, usually without requiring any parameter. If a flat clustering solution is needed though, it must be somehow extracted from the hierarchy, which may require user-defined parameters depending on the technique adopted. In this paper, we employ an effective and efficient technique that does not require any parameter at all.

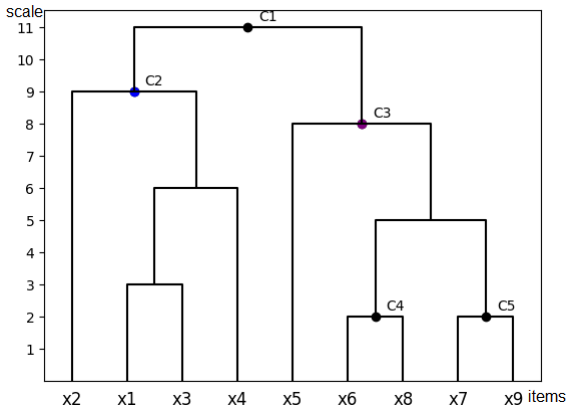


Figure 1 – Hierarchy of clusters displayed as a dendrogram.

This work explores the use of hierarchies of clusters, therefore this section is focused on hierarchical clustering approaches. Hierarchies of clusters can be obtained in many ways, for instance one algorithm can start by isolating every object into its own cluster, called singleton, then clusters can be merged pair by pair following some criteria, e.g. minimum distance between objects of each cluster. Each time clusters are merged a new partition can be extracted, and once all clusters have been merged hierarchically, it is possible to analyze many different possibilities to extract a "flat" partition from the resulting hierarchy. The hierarchy can be represented in many ways, one of them is by the means of a dendrogram. An example of dendrogram is illustrated in Figure 1, where the horizontal axis shows the objects x_n, and the vertical axis shows a scale, indicating when each cluster merger occurred, notice that at first (bottom) each object is isolated, then they merge with another cluster, for instance x_6 and x_8 merge into C_4 and x_7 and x_9 merge into C_5, clusters continue to merge until only one cluster rests, C_2 merges with C_3 forming C_1. Notice that this hierarchy presents many possibilities to extract clusters, one could extract a partition by using the clusters in a arbitrary level, e.g. if we *cut* the dendrogram at scale 10, we would have clusters $C_2 = \{x_2, x_1, x_3, x_4\}$ and $C_3 = \{x_5, x_6, x_8, x_7, x_9\}$. Many other more elegant approaches can be used to extract clusters and this work extends one of them, further explained (in Subsection 2.3.3), to the recommendation scenario. Before that, is important to explain briefly some of the clustering algorithms used in this work.

2.3.2 Algorithms

Three main clustering algorithms are related to this work: K-Means, which is a classic prototype-based algorithm; K-Medoids, which is a medoid-based variant of K-Means; and

Ward's Agglomerative Hierarchical Clustering (Ward's AHC) method, which is used as a basis for our proposed data pre-processing approaches for RS.

- **K-Means:** is a prototype-based clustering algorithm that randomly generates k cluster prototypes, where k is the user-defined number of clusters (given as input), and iteratively adjusts these prototypes as cluster centroids aiming to minimize the total within-cluster variances. The algorithm basically iterates between two steps, namely, assignment of data objects to the closest cluster centroid and adjustment of the centroids, until convergence is reached. Solutions are only guaranteed to be locally optimal, since they depend on the initial prototype initialization. This algorithm is therefore susceptible to bad initialization and usually requires multiple runs to produce more reliable results (GAN; MA; WU, 2007; BAGLA, 2006; ZAHRA *et al.*, 2015).

- **K-Medoids:** is also a prototype-based clustering algorithm, which works similarly to K-Means, except that the prototypes (called medoids), instead of unconstrained centroids, are forced to coincide with objects in the dataset (PARK; LEE; JUN, 2006). Since centroid computation is not required, K-Medoids can operate solely with pairwise distances between data objects, rather than the objects themselves, and unlike K-Means this distance does not need to be Euclidean. In fact, any (dis)similarity measure can be used, such as Pearson or Cosine, which are commonly used in recommendation.

- **Ward's AHC Method:** is an agglomerative hierarchical clustering method (WARD, 1963; GAN; MA; WU, 2007), also known as "minimum variance method". It works as follows (BAGLA, 2006): first, each data object is considered a cluster on its own, called *singleton*, so initially the number of clusters is the same as the database size. In each step, the algorithm merges a pair of clusters, thus producing a new hierarchical level that has necessarily one cluster less than the previous level, until all objects belong together to a single cluster at the highest level (root of the clustering hierarchy/cluster tree). The pair of clusters to be merged in each step consists of the two clusters such that the increment in the total variance of the resulting clusters as a result of the merger is minimal, i.e. the variation between the within-cluster variances before and after the merger is minimized. In principle, Ward's method has a geometrical interpretation only when squared Euclidean distance is used, but the algorithm is oftentimes used successfully (although heuristically) with other (dis)similarity measures. This is particularly important in recommendation because Euclidean distance is well-known not to produce the best results in this domain, due to the sparsity and high-dimensionality of the data (HOULE *et al.*, 2010), so other measures need to be in place.

2.3.3 Cluster Extraction

In this work are proposed automatic strategies designed to produce optimal data clusters for recommendation. This is achieved by selectively picking disjoint clusters from a clustering hierarchy according to a suitable optimization criterion. Our proposed strategies are specialized instances of a general-purpose framework, called FOSC (Framework for Optimal Selection of Clusters from hierarchies) (CAMPELLO *et al.*, 2013), which is explained below.

Let $\{C_1, C_2, ..., C_k\}$ be a collection of clusters in a clustering hierarchy produced by some hierarchical clustering algorithm. In the following, for the sake of simplicity and without any loss of generality, we assume that the clustering hierarchy is (or can be) represented as a binary cluster tree. This is the case, for instance, of all classic AHC algorithms, including Ward's (Section 2.3.2). The root of the cluster tree, C_1, contains the entire set of data objects. In order to select optimal clusters from this cluster tree to form the final partition, $\mathbf{P}$, an objective function $J(\mathbf{P})$ is needed. FOSC is not limited to the selection of one of the levels of the hierarchy, which corresponds to the traditional *horizontal cut* through a clustering hierarchy. Instead, it can actually perform local (rather than global) cuts that allow clusters to be extracted from different hierarchical levels along different branches of the cluster tree. This is only possible, however, if the objective function J to be maximized satisfies two properties, namely: (i) J must be **additive**, which means that it can be written as a sum of individual components $S(C_n)$, each of which is associated with a single cluster C_n of the selected partition $\mathbf{P}$; and (ii) these components must be **local**, which means that every term $S(C_n)$ must be computable locally to C_n, independent of what the other clusters that compose the candidate partition $\mathbf{P}$ are.

The property of *locality* allows the value $S(C_n)$ of every cluster in the cluster tree to be computed prior to the decision on which clusters will compose the final solution to be extracted. *Additivity* means that the objective function can be written as $J(\mathbf{P}) = \sum_{C_n \in \mathbf{P}} S(C_n)$, and the problem is then to choose a collection $\mathbf{P}$ of clusters such that $\mathbf{P}$ is a valid partition (with disjoint clusters whose union is the dataset) that is globally optimal in that it maximizes $J(\mathbf{P})$. Mathematically, the optimization problem can be formulated as (CAMPELLO *et al.*, 2013):

$$\begin{aligned} &\underset{\delta_2,\ldots,\delta_k}{\text{Maximize}} \quad J = \sum_{n=2}^{k} \delta_n S(C_n) \\ &\text{subject to:} \quad \begin{cases} \delta_n \in \{0,1\}, \;\; n = 2, \ldots, \; k \\ \sum_{m \in I_h} \delta_m = 1, \; \forall_h \; \textit{such that } C_h \textit{ is a leaf} \end{cases} \end{aligned} \tag{2.3}$$

where δ_n is an indicator that determines whether C_n belongs ($\delta_n = 1$) or not ($\delta_n = 0$) to the flat solution $\mathbf{P}$ and I_h is the set of cluster indexes on the path from a leaf cluster C_h (included) to the root (excluded). The constraints involving these paths ensure that no data object is assigned to more than one cluster in any valid solution (partition) $\mathbf{P}$, while also ensuring that every object is assigned to a cluster. Problem (2.3) can be solved very efficiently, in linear time and memory

with the number of candidate clusters in the clustering hierarchy, i.e., $O(k)$,[1] using dynamic programming (FITZGIBBON; ALLISON; DOWE, 2000). The dynamic programming algorithm proposed in (CAMPELLO *et al.*, 2013) tackles the problem by iteratively solving and aggregating increasingly bigger instances of the same problem, and is based on the observation that any subtree of the cluster tree is also a cluster tree. This approach is very efficient as it needs only two traversals through the cluster tree, one bottom-up and another one top-down. For the sake of simplicity, we describe here a conceptually equivalent, recursive solution, shown in Equation (2.4).

$$J^*_{C_n} = \begin{cases} S(C_n), \ if\ C_n\ is\ a\ leaf\ cluster\ in\ hierarchy \\ max\{S(C_n),\ \ J^*_{C^l_n} + J^*_{C^r_n}\},\ otherwise \end{cases} \tag{2.4}$$

where C^l_n and C^r_n are the left and right child nodes of an internal node C_n of the cluster tree, respectively, and $J^*_{C_n}$ is the optimal value of the objective function corresponding to the best partition that can be extracted from the subtree rooted at cluster C_n. The solution to Problem (2.3) is the collection of clusters corresponding to the sequence of recursive choices in Equation (2.4) needed to compute $J^*_{C_1}$, where C_1 is the root of the cluster tree. This solution can be interpreted as follows: if the quality of a given cluster C_n, as measured by $S(C_n)$, is better than the aggregated quality of the best possible collection of its sub-clusters, then we discard the sub-clusters and keep C_n ($\delta_n = 1$), otherwise we discard C_n instead ($\delta_n = 0$). Ties can be resolved arbitrarily as they do not affect the final value of the objective function. In our implementation, ties are resolved prioritizing the simpler model, i.e., a parent cluster is preferred to its children.

In (CAMPELLO *et al.*, 2013), the authors proposed a cluster quality measure, $S(C_n)$, based on the notion of lifetime of an object belonging to a cluster in the clustering hierarchy. Clustering hierarchies produced by most hierarchical clustering algorithms, including AHCs such as Ward's, bring a horizontal bar with a scale associated with the different hierarchical levels. In AHCs, this scale represents a related measure of dissimilarity between the pair of clusters that have been merged at each hierarchical level. As an example, consider the clustering hierarchy of a collection of 9 data objects show in Figure 2-c. The root of the tree, cluster C_1, results from the merger of clusters C_2 and C_3, which occurs at value 11 of the scale. Cluster C_2, in turn, results from another merger that occurs at value 9 of the scale. Therefore, notice that the scale span of any data object belonging to cluster C_2, namely x_1, x_2, x_3 or x_4, is the interval $[9,11]$, which has length 2, so we say that the *lifetime* of an object x in cluster C_2, $L(x,C_2)$, is 2. Since more prominent clusters are expected to have more objects with longer lifetimes, the measure of cluster *Stability* was proposed in (CAMPELLO *et al.*, 2013) as the sum of the

[1] Assuming that the hierarchy with the values $S(C_n)$ of each cluster is provided as input.

lifetimes of the objects in the cluster:

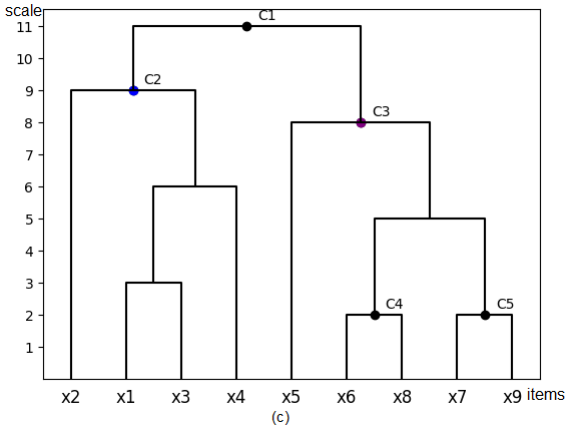
$$S(C_n) = \sum_{x \in C_n} L(x, C_n) \tag{2.5}$$

Scale	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
11	1	1	1	1	1	1	1	1	1
10	2	2	2	2	3	3	3	3	3
9	2	2	2	2	3	3	3	3	3
8	2	2	2	0	3	3	3	3	3
7	2	2	2	0	0	3	3	3	3
6	2	2	2	0	0	3	3	3	3
5	2	0	2	0	0	3	3	3	3
4	2	0	2	0	0	4	5	4	5
3	2	0	2	0	0	4	5	4	5
2	0	0	0	0	0	4	5	4	5
1	0	0	0	0	0	0	0	0	0

(a)

Scale	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
11	1	1	1	1	1	1	1	1	1
10	2	2	2	2	3	3	3	3	3
9	2	2	2	2	3	3	3	3	3
8	2a	2a	2a	2b	3	3	3	3	3
7	2a	2a	2a	2b	3a	3b	3b	3b	3b
6	2a	2a	2a	2b	3a	3b	3b	3b	3b
5	2aa	2ab	2aa	2b	3a	3b	3b	3b	3b
4	2aa	2ab	2aa	2b	3a	4	5	4	5
3	2aa	2ab	2aa	2b	3a	4	5	4	5
2	2aaa	2ab	2aab	2b	3a	4	5	4	5
1	2aaa	2ab	2aab	2b	3a	4a	5a	4b	5b

(b)



(c)

Figure 2 – Three alternative representations of the same clustering hierarchy: (a) The simplified hierarchy filtered with $M_{CS} = 2$; (b) The complete hierarchy, without any prior filtering ($M_{CS} = 1$); (c) The complete hierarchy displayed as a dendrogram, with the clusters of the simplified hierarchy indicated by labels in the corresponding nodes. In (a) and (b), the optimal FOSC solution according to the Stability criterion is highlighted in colors (clusters C_2, in blue, and C_3, in purple)

From Equation (2.5), it is clear that, since $L(x_1, C_2) = L(x_2, C_2) = L(x_3, C_2) = L(x_4, C_2) = 2$ in the example of Figure 2-c, then it follows that $S(C_2) = 4 * 2 = 8$. Not all clustering hierarchies are such that the lifetimes of all objects in a cluster are necessarily the same though. For instance, Figures 2-a and 2-b display two alternative representations of the same hierarchical representation (dendrogram) in Figure 2-c. Rows correspond to hierarchical levels, columns correspond to data objects, and entries represent cluster indexes (labels). While Figure 2-b is a complete hierarchy that is fully equivalent to the dendrogram in Figure 2-c, i.e., a binary tree where every single internal or leaf node is deemed a different cluster, this is not the case in the so-called simplified hierarchy shown in Figure 2-a. This type of hierarchy simplification, as proposed in (CAMPELLO *et al.*, 2013), is obtained by defining an *optional* minimum threshold, called minimum cluster size (M_{CS}), below which a group of objects is not considered a cluster. This way, when interpreting the hierarchy in a top-down fashion, only when a cluster splits into two

clusters each of which has at least M_{CS} objects the split is considered valid and the two new clusters get new labels. When one of the child clusters (or both) does not meet the minimum size requirement, it is labeled as noise (0) and the parent cluster keeps its original label, so it only "shrinks" (or disappears) rather than splits.

This kind of simplification can significantly reduce the size of clustering hierarchies. For instance, the complete hierarchy in Figure 2-b (as well as its equivalent dendrogram representation in Figure 2-c) has $2*9-1 = 17$ different clusters, whereas the simplified one in Figure 2-a keeps only the 5 most prominent ones. A side effect of simplification is that, in a simplified hierarchy, different objects can have different lifetimes in the same cluster. For instance, notice in Figure 2-a that objects x_1 and x_3 join cluster C_2 at level 3 of the scale and stay in this cluster up to level 11, when cluster C_2 merges with C_3 giving rise to C_1, so their lifetimes is $L(x_1,C_2) = L(x_3,C_2) = 11-3 = 8$. Object x_2, however, joins cluster C_2 only at level 6, so its lifetime in this cluster is $L(x_2,C_2) = 11-6 = 5$. Similarly, we have $L(x_4,C_2) = 11-9 = 2$, and the stability of C_2 can thus be computed from Equation (2.5) as $S(C_2) = 2*8+5+2 = 23$.

Notice that Stability as defined in Equation (2.5) satisfies the property of *locality* required by FOSC, as it can be pre-computed independently for each candidate cluster in the cluster tree. If we define an objective function given by the Overall (Sum of) Stability of the clusters to be selected by FOSC to be part of the extracted partition $\mathbf{P}$, i.e., $J(\mathbf{P}) = \sum_{C_n \in \mathbf{P}} S(C_n)$, this objective function is by definition *additive* and, then, FOSC can be applied (CAMPELLO *et al.*, 2013).

In Section 3.2, we propose three alternatives to the standard, general-purpose Stability-based FOSC formulation described above, which change the way $S(C_n)$ is calculated and/or the way noise is handled, always specifically targeting the idiosyncrasies of the recommendation scenario.

2.4 Related Work

In this section, we review previous studies that apply clustering to traditional recommendation problems. In most of these studies, e.g. (NAJAFABADI *et al.*, 2017; ZAHRA *et al.*, 2015; VLACHOS *et al.*, 2014), the clustering step is embedded as part of the recommendation system, binding the clustering phase to a specific recommender algorithm, preventing or hindering the use of a different algorithm.

The use of data clustering to support recommendation is not a recent idea. In a 1998 paper (UNGAR; FOSTER, 1998), the authors introduced a method to predict if a user will have interest in a given item, which works essentially by generating groups of users and items, with the aid of statistical models in order to find spots with high probability of a user-item interaction. Similarly, recent studies make explicit use of co-clustering/bi-clustering techniques, which cluster users and items simultaneously, in order to guide decision making in recommendation. For instance, in (VLACHOS *et al.*, 2014) the authors apply co-clustering by first grouping users and items

independently with K-Means, then the groups in each dimension are combined and the resulting co-clusters are used to detect user-item pairs for which the user is likely to have interest in the item. However, in order to rank these pairs, the authors combine a variety of external information about users, which may not be available in many RS applications/databases.

Other studies perform clustering of either items or users in order to support recommendation. In (LI; KIM, 2003), the authors describe a K-Means clustering-based recommender strategy that recommends items to users based on pairwise similarities between items. This approach needs attributes to describe the items, which may not be available, and it is sensitive to parameters as well as to the randomness involved in the initialization of K-Means. Instead of items, Shinde and Kulkarni (SHINDE; KULKARNI, 2012) cluster users by their rating tendencies, namely, users that tend to give high ratings, low ratings, or neutral. The proposed method applies a centroid-based clustering algorithm that requires two critical parameters in order to obtain three groups and centroids which are then used to predict the rating that a user would give to each item. In a similar approach, Zahra et al. (ZAHRA *et al.*, 2015) explore many centroid-based strategies to cluster users, where a cluster centroid can be interpreted as the mean rating given by the users of the corresponding group, recommendations are then provided based on the nearest k centroids. In (NAJAFABADI *et al.*, 2017), the authors propose the use of hierarchical clustering in order to categorize items based on external attributes (e.g. author, year, etc. of a music). Once the items have been categorized, users are related to groups of items based on their interactions, using association rules, and users' preferences are estimated. All these approaches require extended information beyond an ordinary user-item matrix, for instance (SHINDE; KULKARNI, 2012; ZAHRA *et al.*, 2015) use explicit feedback of scores whereas (NAJAFABADI *et al.*, 2017) use enriched description of the items. Extended information may not be available in every application domain. In addition, the clustering phase is often bound to the particular recommender algorithm adopted. For instance, it is not possible to apply a matrix factorization recommender system when using the method in (ZAHRA *et al.*, 2015), given the way the data is handled at the clustering phase of that method.

The use of clustering to segment the database as a pre-processing step for recommendation can be motivated by different reasons, such as reduce sparsity (CONNOR; HERLOCKER, 2001), reduce computational time (SUNANDA; VINEELA, 2015), or to improve quality of predictions by performing more personalized, group-focused recommendations (COSTA; MANZATO; CAMPELLO, 2016). This type of approach allows the application of any recommender system subsequently to the clustering step. In (CONNOR; HERLOCKER, 2001), the authors segment items using Average-Linkage hierarchical clustering, then recommendations are performed considering only the interactions within subgroups of items, which is motivated by the reduction in the sparsity of the data, since items in the same group are expected to be similar in terms of interactions. Similarly, the authors in (SUNANDA; VINEELA, 2015) also use Average-Linkage to cluster items, but in this case items are clustered based on their textual description, and the main motivation is to reduce computational time in the recommendation step, since the number

of items in each recommender is reduced. Both aforementioned studies, however, arbitrarily choose critical parameters to select clusters from the Average-Linkage clustering hierarchy.

Aiming to improve the quality of recommendations, the authors in (COSTA; MANZATO; CAMPELLO, 2016) use different types of feedback, e.g. ratings and accesses to items, in order to produce multiple group-focused recommendations. K-Medoids clustering is applied to each type of feedback in order to group users, then for each group of users a recommender is applied to the corresponding interactions, and finally the predictions obtained from each type of feedback are combined in order to generate a single list of recommendations for each user. This approach, like many others reviewed here, require enriched information, in this case different types of feedback, which may not be available in every application domain. In addition, a recommender must be trained for each type of feedback, which may increase the computational cost of the recommender step.

In summary, clustering has been used to support recommendation, either as a pre-processing step where the data is processed in such a way that any recommender can be applied (CONNOR; HERLOCKER, 2001; SUNANDA; VINEELA, 2015; COSTA; MANZATO; CAMPELLO, 2016), or as a tool to infer or elicit information to be used in a particular recommender algorithm (PEREIRA; HRUSCHKA, 2015; VLACHOS *et al.*, 2014; LI; KIM, 2003; SHINDE; KULKARNI, 2012; ZAHRA *et al.*, 2015; NAJAFABADI *et al.*, 2017). The first approach is more general in scope since it permits any recommender system to be applied, tacking sparsity and reducing computational burden. This is the approach we follow in this paper.

Our proposal differs from others reviewed here in section 2.4, e.g. (SUNANDA; VINEELA, 2015; NAJAFABADI *et al.*, 2017; VLACHOS *et al.*, 2014; COSTA; MANZATO; CAMPELLO, 2016), for not requiring enriched information or any specific type of feedback, while being able to use enriched information when it is available. Furthermore, previous studies rely heavily on critical user-defined parameters, which can be relatively simple such as the number of clusters (CONNOR; HERLOCKER, 2001; COSTA; MANZATO; CAMPELLO, 2016) or more complex thresholds that are hard to tune for each particular dataset (SHINDE; KULKARNI, 2012). Our proposal does not suffer from these limitations.

2.5 Final Remarks

In this chapter concepts important for a better understanding of this research were presented, specially the FOSC cluster extraction technique which is the foundation of this work.

Concepts regarding Recommender Systems were also presented, starting with a brief contextualization, then explaining filtering methods and three approaches to Collaborative Filtering: neighborhood-based; latent-factor and non-personalized, including representative examples of those. Moreover, clustering was contextualized, some algorithms were presented, two partitional (K-Means and K-Medoids) and one hierarchical (Ward's AHC) were described,

the concept of hierarchical clustering was explained, alongside with a dendrogram representation for hierarchies. In addition, a cluster extraction approach that optimizes the quality of the selected clusters was presented, such an approach will be extended in this work and its understanding is fundamental. Lastly, the related work was described, in particular the literature that applies clustering to aid the recommendation task, highlighting the main issues in the area, such as: the binding between clustering and recommender phases; the amount of information needed to apply such techniques; or parameter sensitivity. With each solution being addressed in this work.

The following chapter addresses the three proposed cluster extraction approaches that where developed.

CHAPTER

3

PROPOSAL

In this chapter are presented three novel variants of the standard FOSC formulation described in Section 2.3.3, which are specifically designed to directly tackle RS problems, namely data sparsity and the need to ensure that there is a minimum amount of user information in each group in order to successfully perform group-based recommendation.

3.1 Contextualization

Even though the cluster extraction techniques investigated in this work can in principle be applied to both users or items, here we focus on the problem of clustering users as a pre-processing step prior to the application of RS. The groups of users generated by our clustering techniques can be inputted into an RS according to the following scheme:

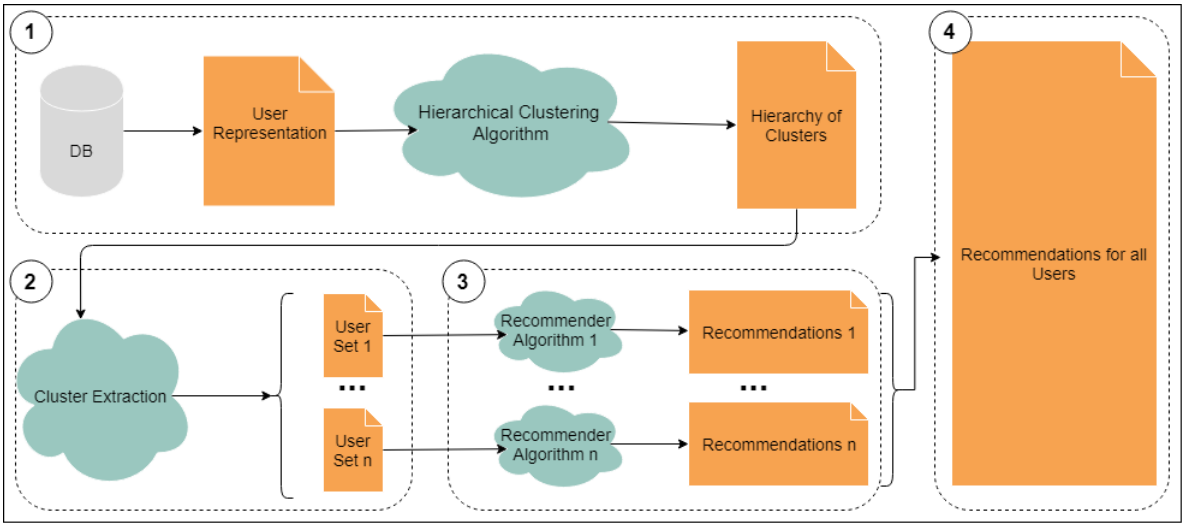


Figure 3 – Scheme to apply hierarchical clustering to aid RS.

1. Apply a hierarchical clustering algorithm to the rows of the user-item interaction matrix in order to produce a clustering hierarchy of users according to their interactions with the items;[1]

2. Extract a partition of disjoint clusters of users from the hierarchy using one of our new variants of FOSC suitable for the RS domain;

3. For each extracted cluster, apply a recommender algorithm, considering only information regarding the users in that cluster;

4. Concatenate the generated rankings (of user-item pairs) in order to perform the final recommendation.

Notice that the scheme above can indirectly help reduce data dimensionality in RS because, by considering subsets of similar users independently, certain categories of items may not have any interactions at all inside certain groups of users, and can thus be ignored by the respective recommender. Also, notice that the automatic extraction of clusters from a clustering hierarchy is advantageous when compared to the direct partitioning clustering approach: first, the number of clusters does not need to be estimated or provided as input by the analyst; second, hierarchical algorithms are parameterless and deterministic, so they are not susceptible to initialization of prototypes or any other source of randomness that may affect other algorithms (e.g. K-Means); third, the clustering and the extraction can be treated as separate problems, so different clustering and extraction techniques can be combined; lastly, FOSC can extract clusters from multiple hierarchical/granularity levels, which may contain clusters with highly different densities and sizes, which is a difficult problem for most flat clustering algorithms.

3.2 Cluster Extraction for RS

In principle, the standard FOSC formulation described in Section 2.3.3 could be applied in Step 2 of the scheme proposed above. However, the standard formulation has not been designed to directly tackle RS problems. One of these problems is the need to ensure that there is a minimum number of users in each group in order to successfully perform group-based recommendation. Recall that an optional, minimum cluster size threshold, $M_{CS} > 1$, can be applied to simplify the clustering hierarchy prior to the application of FOSC (see the example in Figure 2). In other words, clusters that do not meet the minimum size requirement can be disregarded as noise in the simplified hierarchy, as a pre-processing mechanism that is already available in the original FOSC formulation. Nevertheless, this mechanism may not be suitable in the RS domain. In fact, it may not be suitable to use such a simplified hierarchy in recommender systems because some data objects may end up as noise in the extracted partition. For instance,

[1] A suitable (dis)similarity measure between users, as rows of the user-item interaction matrix, is required. Pearson or Cosine are recommended in RS applications.

if a solution $\mathbf{P} = \{C_2, C_4, C_5\}$ was extracted from the simplified hierarchy in Figure 2-a, object x_5 would be left unclustered as noise (0). This may be inappropriate in the RS context because the corresponding user would be excluded from the recommendation, provided that it does not belong to any cluster. Besides, it is meaningless to put all noise objects (i.e., all unclustered users) into a common "rag bag cluster", because noise-labeled users can be completely unrelated to each other.

To avoid the occurrence of noise, we always use a complete hierarchy, such as in Figure 2-b, when performing cluster extraction for recommendation. Therefore, we need an alternative mechanism to enforce that the extracted clusters meet the minimum size threshold and thereby ensure that there is enough user information in each group to perform recommendation. Our solution is to enforce the threshold M_{CS} during the optimization process to extract clusters, rather than as a pre-processing of the clustering hierarchy.

The difference between enforcing $M_{CS} > 1$ before or during FOSC can be illustrated using once again the example in Figure 2. For instance, in the simplified hierarchy in Figure 2-a, produced by applying $M_{CS} = 2$ prior to any cluster extraction, solution $\mathbf{P} = \{C_2, C_4, C_5\}$ is a valid candidate as it satisfies all the constraints of Problem (2.3). If we use instead the complete hierarchy in Figure 2-b and enforce $M_{CS} = 2$ during cluster extraction, this is no longer a feasible solution because by extracting C_2, C_4, and C_5, cluster C_{3a} would necessarily have to be extracted; it cannot be extracted though, since it contains a single object, x_5 ($|C_{3a}| = 1 < M_{CS}$).

It is worth noticing that there is an upper bound to the value of M_{CS} above which no feasible solution exists. In particular, if M_{CS} is larger than the size of any of the child nodes directly descendant from the root (C_1), then only C_1 would be a feasible solution, but this "solution" is intentionally blocked in the optimization formulation and FOSC algorithm as it does not represent any clustering of the data at all.

The way we adjust FOSC to incorporate M_{CS} into the optimization process will be explained in the sequel. We propose two variants of this technique. The first one, called *LifetimeMCS*, uses the same optimization criterion as in the original FOSC publication (CAMPELLO *et al.*, 2013) — *Stability* (see Section 2.3.3) — the only difference is the use of M_{CS} *during* (rather than prior to) the optimization process. The second variant, called *SparsityMCS*, goes further and also changes the nature of the optimization criterion originally used by FOSC. In particular, it uses a quality measure based on the sparsity of the user-item interactions within clusters, as a new domain-specific criterion that can be shown to satisfy the properties of locality and additivity required by FOSC, while directly addressing the sparsity problem in RS during the clustering extraction stage. Additionally, we propose a noise handling approach in order to use the original FOSC formulation in RS, called LifetimeMCSGlobal.

3.2.1 LifetimeMCS

LifetimeMCS stands for lifetime with minimum cluster size. This approach uses the same lifetime-based *Stability* criterion proposed in the original FOSC publication (CAMPELLO *et al.*, 2013) (see Section 2.3.3), but modified so that the optimization criterion itself enforces the constraint that a cluster must have at least M_{CS} objects. Adding this constraint can be accomplished by including a condition in the computation of the quality of a cluster C_n, $S(C_n)$. Specifically, for every cluster C_n with fewer than M_{CS} objects, we set $S(C_n) = -\infty$, otherwise $S(C_n)$ is calculated normally as previously shown in Equation (2.5). Notice that, in practice, this has the same effect as including an additional explicit constraint into Problem (2.3), because a cluster with $S(C_n) = -\infty$ will not be selected when trying to maximize J.[2] The modified cluster quality measure can then be written as:

$$S(C_n) = \begin{cases} -\infty, \ if \ |C_n| < M_{CS} \\ \sum_{x \in C_n} L(x, C_n), \ \ otherwise \end{cases} \tag{3.1}$$

where $L(x, C_n)$ is the lifetime of object x as part of cluster C_n and $|C_n|$ is the cardinality (number of data objects) of this cluster. Recalling that (i) this equation will always be applied to a complete (not simplified) clustering hierarchy, and (ii) the lifetime of any object in a complete clustering hierarchy (such as the one in Figure 2-b) is the same, then Equation (3.1) can be simplified as:

$$S(C_n) = \begin{cases} -\infty, \ if \ |C_n| < M_{CS} \\ |C_n| \cdot Lifetime(C_n), \ \ otherwise \end{cases} \tag{3.2}$$

where $Lifetime(C_n)$ is the lifetime of cluster C_n. For instance, in the example of Figure 2-b, cluster C_4 exists between the scale values 2 (when it is born as a union of clusters C_{4a} and C_{4b}) and 5 (when it merges with C_5 giving rise to C_{3b}), so $Lifetime(C_4) = 5 - 2 = 3$. This cluster has $|C_4| = 2$ objects. If $M_{CS} \leq 2$, then $S(C_4) = 2 * 3 = 6$, otherwise $S(C_4) = -\infty$.

This adjustment does not affect the *locality* and *additivity* properties of the objective function required by FOSC because the additional condition, $|C_n| < M_{CS}$ in Equation (3.2), can be checked independently for each cluster. It does not change the computational complexity of FOSC either, which is linear with the number of candidate clusters in the clustering hierarchy, i.e. $O(k)$, provided that the clustering hierarchy and quality measure for each cluster, $S(C_n)$, are given as input. In AHC algorithms, such as Ward's and other similar algorithms, the values $S(C_n)$ can be straightforwardly computed simultaneously to the construction of the clustering hierarchy itself; all one needs is to keep track of the number of objects in each cluster and the values of the scale where the cluster appears and disappears. This doesn't change the computational complexity of the clustering algorithm. Most AHCs can be implemented in $O(N_o^2 \ log \ N_o)$ or $O(N_o^2)$ time,

[2] The same principle can be used to rule out the root C_1 as a valid candidate and force a solution with at least two clusters, by just setting $S(C_1) = -\infty$.

where N_o is the number of data objects. These algorithms produce a complete hierarchy with $k = 2N_o - 1$ clusters, so FOSC runs in $O(k) \rightarrow O(N_o)$ time. In our context, $N_o = |U|$ (number of users in the database).

3.2.2 SparsityMCS

SparsityMCS stands for sparsity with minimum cluster size. Like LifetimeMCS, it also enforces M_{CS} as part of the optimization process. The difference is the optimization criterion itself: instead of using the lifetime of a cluster, we now aim to directly minimize the sparsity of the within-cluster interactions. Consider a binary user-item matrix where each entry (u,i) is set equal to 1 if user u has interacted with item i, or 0 otherwise. Let $Inter(C_n)$ denote the total number of interactions by users belonging to cluster C_n, and let $Items(C_n)$ be the subset of all items for which at least one user in C_n has had an interaction with. We aim to select clusters with low sparsity of interactions, because sparsity is a major drawback in recommendation. Notice, however, that Problem (2.3) is a *maximization* problem, so in order to minimize sparsity we can equivalently maximize the *density* of within-cluster interactions. For a user-item matrix of interactions and a given cluster C_n of users, we define the *density* of cluster C_n, $density(C_n)$, as:

$$density(C_n) = \frac{Inter(C_n)}{|C_n| \cdot Items(C_n)} \tag{3.3}$$

This equation can be interpreted as the ratio between the total number of observed interactions by the users in the cluster to the maximum number of possible interactions by those same users with the same items. For the sake of illustration, let us consider the matrix in Figure 4, which contains only users of a given cluster C_n along with their interactions. In this example, it follows that $Inter(C_n) = 17$ (the sum of non-zero entries in the table), $Items(C_n) = 8$, $|C_n| = 5$, and, therefore, $density(C_n) = 17/(5 \cdot 8) = 17/40 = 0.425$ (17 out of 40 elements in the table are non-zero).

C_n	i_5	i_{420}	i_{677}	i_{908}	i_{1082}	i_{1234}	i_{1589}	i_{2034}
u_{32}	1	0	0	1	0	1	0	0
u_{58}	0	1	0	1	0	0	0	0
u_{87}	0	0	1	0	1	0	1	1
u_{128}	0	1	0	0	1	0	1	0
u_{235}	1	1	0	1	0	1	0	1

Figure 4 – Example of user-item interaction sub-matrix for users of a cluster C_n.

Density alone, however, does not take into account another relevant aspect in recommendation, which is the representability of the group of users. For instance, if two groups of users have similar density of interactions but one group is larger than the other, the larger group is preferred as it has similar density yet more information about users. To account for both density and representability, we can weight the density of a cluster by its cardinality, in the same way that LifetimeMCS in Equation (2.5) weights the lifetime of a cluster by its number of objects.

Like LifetimeMCS, we can also enforce the M_{CS} constraint as part of objective function, thus giving rise to the following cluster quality measure, referred to here as *SparsityMCS*:

$$S(C_n) = \begin{cases} -\infty, \ if \ |C_n| < M_{CS} \\ |C_n| \cdot density(C_n), \ \ otherwise \end{cases} \tag{3.4}$$

The rationale behind this measure is to favor clusters that meet the M_{CS} requirement while being at the same time as dense and as large as possible. In our example, $S(C_n) = 5 \cdot (17/40) = 17/8 = 2.125$ if $M_{CS} \leq 5$, or $S(C_n) = -\infty$ otherwise. Although this measure is easy to interpret, it is not as easy to understand whether or not it meets the requirements of the FOSC framework. In the following, we will rewrite this measure in a fully equivalent way, whereby the corresponding optimization criterion can be trivially shown to satisfy the assumptions of locality and additivity required by FOSC.

First of all, condition $|C_n| < M_{CS}$ can be checked/enforced locally to each cluster and does not violate the desired properties, then we will focus on the quality of those clusters that satisfy this condition, as measured by $S(C_n) = |C_n| \cdot density(C_n)$. We can rewrite $S(C_n)$ in a fully equivalent way that emphasizes the contribution of each data object individually to the density of a cluster. To that end, let us define the density of an object (user) u_m that belongs to a cluster (group of users) C_n, $d_s(u_m, C_n)$, as the fraction of items in $Items(C_n)$ (i.e., within the collection of items with interactions by users in C_n) for which user u_m has an interaction with. For instance, in our example in Figure 4, it turns out that user u_{32} has interactions with 3 out of the $Items(C_n) = 8$ items in its group, therefore $d_s(u_{32}, C_n) = 3/8$. Similarly, $d_s(u_{56}, C_n) = 2/8$, $d_s(u_{67}, C_n) = 4/8$, $d_s(u_{128}, C_n) = 3/8$, and $d_s(u_{235}, C_n) = 5/8$. We can then rewrite the weighted density of cluster C_n, $S(C_n) = |C_n| \cdot density(C_n)$, as the sum of the individual densities of its objects, whereby SparsityMCS in Equation (3.4) can be rewritten as:

$$S(C_n) = \begin{cases} -\infty, \ if \ |C_n| < M_{CS} \\ \sum_{u_m \in C_n} d_s(u_m, C_n), \ \ otherwise \end{cases} \tag{3.5}$$

which, in our example, is $S(C_n) = d_s(u_{32}, C_n) + d_s(u_{56}, C_n) + d_s(u_{67}, C_n) + d_s(u_{128}, C_n) + d_s(u_{235}, C_n) = 3/8 + 2/8 + 4/8 + 3/8 + 5/8 = 17/8 = 2.125$ (if $M_{CS} \leq 5$, or $S(C_n) = -\infty$ otherwise).

Two conclusions can be drawn:

- The measure in Equations (3.4) and (3.5) is structurally equivalent to the ones in Equations (3.2) and (3.1), respectively, just replacing *lifetime* of an object/cluster with *density* of an object/cluster; and
- They can be pre-computed locally and independently for each cluster in the clustering hierarchy. Therefore, if we define an objective function given by the Overall (Sum of)

$S(C_n)$ values of the clusters to be selected by FOSC to be part of the extracted partition $\mathbf{P}$, i.e., $J(\mathbf{P}) = \sum_{C_n \in \mathbf{P}} S(C_n)$, this objective function is by definition additive and, thus, FOSC can be applied (CAMPELLO *et al.*, 2013).

Complexity: Computationally, we can calculate SparsityMCS efficiently by taking advantage of the fact that, in practical applications of RS, the user-item matrix is very sparse. We can keep this matrix in a suitable sparse matrix data structure. In particular, we assume a data structure consisting of a list of $|U|$ lists, one per user, each of which consists of a linked collection of items for which the respective user has interacted with (as well as its size, accessible in O(1)). Then, to compute the SparsityMCS measure for a given cluster C_n, we need the number of users in this cluster, $|C_n|$, the total number of interactions in this cluster, $Inter(C_n)$, and the cardinality of the subset of items involved in those interactions, $Items(C_n)$. Notice that, for a parent cluster C_n in the clustering hierarchy, the first two terms, namely $|C_n|$ and $Inter(C_n)$, are both given by the sum of the corresponding terms of this cluster's children, i.e. $|C_n| = |C_n^l| + |C_n^r|$ and $Inter(C_n) = Inter(C_n^l) + Inter(C_n^r)$. Therefore, we can compute these terms in a single bottom-up pass through the clustering hierarchy, just adding the individual figures from the children to compute the aggregated tally for the corresponding parent. The complexity of this phase is the same as FOSC, i.e., $O(k) \rightarrow O(2|U|-1) \rightarrow O(|U|)$.

Term $Items(C_n)$, however, does not hold this property, because there may be items shared by the users in the children clusters. For this reason, $Items(C_n) = Items(C_n^l \cup C_n^r)$, which is only equal to $Items(C_n^l) + Items(C_n^r)$ if the two respective subsets of items are disjunct. Since this condition cannot be verified beforehand, we need to explicitly determine the subsets of items for the users in each cluster as the union of the subsets of items of its children. This can be achieved efficiently as follows: starting from the individual lists of items for each user, we compute a list of items involving the users of any parent cluster C_n as an union of the corresponding lists previously computed for its children; term $Items(C_n) = Items(C_n^l \cup C_n^r)$ is simply the cardinality of the resulting list. Once the union list for a parent cluster is computed, the lists in the children can be discarded to free memory. This approach is particularly efficient in real application scenarios where interactions are very sparse.

In terms of asymptotic bounds, the (unrealistic) worst-case scenario corresponds to a full user-item matrix, where every user has interacted with every item. In this case, for each cluster in the cluster tree, we need to traverse two lists with $|I|$ items in the children to compute their union, which will also have $|I|$ elements. This takes $O(|I|)$ time per cluster. Since the cluster tree has $2|U|-1$ clusters, the overall time complexity is $O(|U| \cdot |I|)$, i.e., it is linear with the size of the user-item matrix. This is, however, a conservative upper-bound, because the user-item matrix tends to be very sparse in practical applications of RS, so the lists of items per user and their unions in the corresponding clusters tend to be much smaller than $|I|$, except in the upper levels of the tree (where the number of clusters and lists to be merged is smaller though).

3.2.3 Outliers Handling Alternative

Both methods proposed before are based on the premise that all clusters need to have at least M_{CS} users in order to be applied to a recommender and this is enforced at the optimization criterion. Although the restriction $-\infty,\ if\ |C_n| < M_{CS}$ in both equations (3.2) and (3.4) can be removed, this approach might be desirable because stable clusters with a little amount of users might indicate *outliers*, i.e. users that may impact negatively on the quality of the recommendations. By allowing such small clusters, is possible to isolate and treat them in a special way, preventing the impact of outliers at recommendation step. However, allowing such small clusters to be present in the final extraction presents some problems: (i) these groups might be too small, providing little information for the recommender to be applied, and (ii) as stated before, there is no meaning to group together every noise cluster since they may represent very different users. These problems might be solved by using the information of the outlier user's closest noise-free cluster. However, this is undesirable because a recommender model would be needed for every single user labeled as noise leading to a overhead on the recommendation step.

One approach that solves these problems is to use global information to recommend to these outlier users, i.e. for these users the recommendation is accomplished in the exact same way as it would be if used the entire database, although for the users in larger groups the recommendation would be local and presumably improved. The M_{CS} ensuring can be done *after* the extraction step and prior to the recommendation, simply filtering the users in groups with less than M_{CS} users and for every user in such groups recommend using the entire database.

It is worth to notice that removing the restriction $-\infty,\ if\ |C_n| < M_{CS}$ has no meaning when applied to SparsityMCS (3.4), as the extraction will lead to a set of many clusters with 100% density given that, for a single user, the density as calculated in (3.3) will be maximal. Therefore, no information is left for the recommender and such clusters, besides their small length, have no room for recommendation. Thus, this outliers handling alternative approach will be applied only for lifetimeMCS (3.2) and will be called LifetimeMCSGlobal. Notice that a M_{CS} parameter is still needed in order to filter the users that will use the global recommendation.

3.2.4 Parameter Setting

Notice that all three approaches, LifetimeMCS, SparsityMCS and LifetimeMCSGlobal, incorporate a *Minimum Cluster Size* parameter, M_{CS}. Preliminary experiments have shown that this parameter is not critical as small changes in its value have limited impact over the final recommendation. Basically, M_{CS} only needs to be big enough in order to ensure that each extracted cluster has enough users (and, accordingly, interactions) for the application of the recommender system of choice — for instance, a minimum number of users for the suitable computation of neighborhood in neighborhood-based recommender algorithms. It is worth remarking that, even though clustering is an unsupervised task, one can perform model selection to choose the value of M_{CS} that results in the best recommendation result according to a given

recommender algorithm and (supervised) recommendation evaluation measure of choice. In the pre-processing phase, the clustering hierarchy would not need to be recomputed as it is not affected by M_{CS}. Only FOSC would need to be rerun for different values of M_{CS}.

3.3 Final Remarks

In this chapter were presented the proposed techniques, starting with a general scheme which helps to understand the link between the pre-processing proposed and the recommendation step. Then we proposed two novel formulations to handle the quality of the extracted groups, focusing on aspects important to RS, such as minimum information and sparsity reduction. Beside that a noise handling approach was proposed and explained, which may help to reduce the impact of outliers in the recommendation step. Finally, we showed that the parameter asked by all the proposed methods, M_{CS}, is robust and small changes do not affect the final recommendations. Therefore the same value might be used in many different databases, as we did on experimentation phase that follows in the next chapter.

CHAPTER

4

EXPERIMENTS AND RESULTS

In this chapter are presented the experiments conducted in order to evaluate the impact of our cluster extraction techniques, also discussing the obtained results.

4.1 Experimental Evaluation

To evaluate the impact of our proposed techniques for cluster extraction from hierarchies, we provided the obtained groups of users to three well-known recommender algorithms: User-KNN, BPR-MF and Most Popular (MP), with their default parameters suggested by the original authors. The quality of the recommendations obtained are measured in different clustering pre-processing scenarios:

- **No split:** As a baseline, we apply the recommenders without any clustering pre-processing at all, i.e., recommendation is performed using the complete user-item matrix as it is;
- **K-Means:** As another competitor, users are clustered using the classic K-Means algorithm, which has been previously applied in the context of recommendation — e.g. see (ZAHRA *et al.*, 2015; VLACHOS *et al.*, 2014). Recommenders are then applied separately to each group of users;
- **K-Medoids (K-Med):** Users are clustered using K-Medoids, instead of K-Means, as an additional competitor. This algorithm has also been previously applied in the context of recommendation — e.g. (COSTA; MANZATO; CAMPELLO, 2016);
- **Our Proposal:** Users are clustered using the techniques introduced in this paper, LifetimeMCS (LT), SparsityMCS (Spar) and LifetimeMCSGlobal (LTG), applied to a clustering hierarchy obtained by the Ward's AHC method. Recommenders are then applied separately to each resulting group of users.

4.2 Databases

The experiments were performed using nine datasets from different domains: music, movies, jokes and books, varying in size as well as in their configurations of users, items and interactions. Each dataset is described below and summarized in Table 1.

- BookCrossing (BX)[1], from Ziegler et al. (ZIEGLER *et al.*, 2005) and reduced by maintaining only the items and users with at least 20 interactions as in (KUMAR *et al.*, 2014), the final version consists of 45,074 ratings given by 3217 users to 2032 books;

- FilmTrust (FT), from Guo et al. (GUO; ZHANG; YORKE-SMITH, 2013), consists of 35,497 ratings provided by 1,508 users to 2,071 movies;

- Frappe[2] is an mobile app recommender database provided by Baltrunas et al. (BALTRUNAS KAREN CHURCH, 2015), consisting of 96,203 ratings for 4082 apps given by 957 users;

- MovieLens100k (ML)[3], from Harper and Konstan (HARPER; KONSTAN, 2015), consists of 100,000 ratings provided by 943 users to 1682 movies, each user rated at least 20 movies;

- Hetrec-lastFM (LFM)[4], from Cantador et al. (CANTADOR; BRUSILOVSKY; KUFLIK, 2011), consists of 92,834 user-listened-to-artists relations involving a set of 1892 users and 17,632 artists;

- Restaurant (R)[5] provided by Vargas-Govea et al. (VARGAS-GOVEA; GONZÁLEZ-SERNA; PONCE-MEDELLIN, 2011) presents 1161 ratings for 130 restaurants given by 138 users;

- Steam Purchases (Steam)[6], where only the purchase information was used, has 12,393 users, 5155 games and 129,511 purchases.

- *Yahoo!* Music (YMus) and *Yahoo!* Movies (YMov), both from *Yahoo!* Webscope[7], consist of 365,704 interactions of 15,400 users to 1000 musics and 169,767 interactions provided by 4385 users to 4339 movies, respectively;

[1] http://www2.informatik.uni-freiburg.de/ cziegler/BX/
[2] http://baltrunas.info/research-menu/frappe
[3] https://grouplens.org/datasets/movielens/100k/
[4] http://www.lastfm.com
[5] https://www.kaggle.com/uciml/restaurant-data-with-consumer-ratings/version/1#_¯_
[6] https://www.kaggle.com/tamber/steam-video-games/data
[7] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

Table 1 – Databases

Database	Alias	Users	Items	Interactions	Sparsity (%)
BookCrossing	BX	3217	2032	45074	99.31
FilmTrust	FT	1508	2071	35497	98.86
Frappe	Frappe	957	4082	96203	97.54
Hetrec-lastFM	LFM	1892	17632	92834	99.72
MovieLens-100k	ML	943	1682	100000	93.70
Restaurant	R	138	130	1161	93.53
SteamPurchases	Steam	12393	5155	129511	99.80
Yahoo!Movies	YMov	4385	4339	169767	99.11
Yahoo!Music	YMus	15400	1000	365704	97.63

4.3 Methodology

We used the parameters suggested by the original authors when applying the recommender algorithms. Specifically, for User-KNN we set the number of neighbors to 30, and for BPR-MF we set the number of factors to 10. We used Case Recommender 0.0.20 (COSTA; MANZATO, 2016) for the User-KNN and Most Popular algorithms; MyMediaLite 3.11 (GANTNER *et al.*, 2011) was used for BPR-MF. For the M_{CS} clustering extraction parameter, we used $M_{CS} = 50$ for all experiments because, as discussed in Section 3.2.4, this parameter has little impact over the final recommendations.

For clustering algorithms, K-Medoids requires a (dis)similarity matrix between users. We used Cosine similarity, which is well-known to be more suitable than Euclidean distance in the RS domain. We have also provided the same (Cosine) similarity matrix to Ward's AHC method. As for the number of clusters explicitly required as input by K-Medoids and K-Means, we varied this parameter as $\{3,5,7,10\}$. Similar values have been successfully used in the literature; for instance, in (COSTA; MANZATO; CAMPELLO, 2016) the number of clusters was set to 3, while the authors in (BILGE; POLAT, 2013) used the same range we use. Both K-Medoids and K-Means require multiple runs from different random initializations, we did run them 5 times for each configuration and present the mean value. K-Medoids and Ward's AHC were implemented using SciPy (0.19.0)[8] whereas for K-Means we used Scikit-learn (0.19.0)[9], both libraries for Python 3.6.

In order to evaluate the results, we adopted a 5-fold cross-validation methodology that consists of spliting the databases in 5 disjoint parts of equal size (KOHAVI, 1995). One part is used as test and the others are used as training set. The process is repeated for each part, generating 5 different training-test sets. To evaluate the outcomes of the algorithms, we evaluate the results for each set and take the mean value. In the present work we measure the quality of recommendations by the Mean Average Precision-at-N ($MAP@N$) (AGGARWAL, 2016), using $N = \{1,3,5,10\}$.

In order to assess statistical significance between the results we applied a modified

[8] https://docs.scipy.org/doc/
[9] http://scikit-learn.org/stable/

Friedman-test and the pot-hoc procedures both proposed by Demšar (DEMŠAR, 2006), which allows a comparison between many different algorithms in different scenarios, here with confidence level of 95%.

4.4 Results

For the sake of clarity, we discuss the results corresponding to the different recommender algorithms in separate subsections, each one by the lens of one particular RS, considering all databases. Tables of results contain: (a) a column indicating the database used; (b) a column indicating the pre-processing strategy applied, where our proposed methods are highlighted in bold; (c) a column, #Clu, which indicates the number of clusters obtained in the pre-processing phase and subsequently used to build separate recommenders[10]; (d) columns $MAP@N$, $N = 1, 3, 5, 10$, containing the cross-validated mean of $MAP@N$. Values highlighted in bold are the highest values for a given measure in a given database. For K-Medoids and K-Means, the values displayed are the means of 5 runs from random initializations of prototypes.

4.4.1 Neighborhood-based Recommender

For the User-KNN algorithm, Table 2 shows that at least one of our proposed methods improved on the baseline in 7 databases, and outperformed the other methods in 4 of them. In 2 databases, namely BX and R, no clustering technique could improve on the baseline.

4.4.2 Latent Factor Based Recommender

For the BPR-MF algorithm, the general picture is similar. Table 3 shows that at least one of our proposed methods improved on the baseline in 6 databases, and outperformed the other methods in 4 of them. Once again, in 3 databases, namely ML, R and YM, no clustering technique could improve on the baseline. It is worth noticing that, even for these datasets, the proposed methods have still provided competitive results overall. One case that is worth noticing is dataset Steam, where SparsityMCS improved the quality of the recommendations significantly further than its competitors.

4.4.3 Most Popular Recommender

The most popular recommender is a non-personalized algorithm, i.e. it computes the same recommendations for all users within the group. Clustering prior to recommendation has improved the results in all datasets, as it can be seen in Table 4. At least one of our proposed methods has provided the best result in 7 out of 9 datasets.

[10] Notice that the proposed clustering extraction techniques may find different amount of clusters for each fold used during training; therefore, for LifetimeMCS, SparsityMCS and LifetimeMCSGlobal, #Clu presents the mean value found for all folds.

Table 2 – MAP@N of User-KNN

Database	Method	#Clu	MAP@1	MAP@3	MAP@5	MAP@10
BX	Baseline	1	**0.0817**	**0.1059**	**0.1121**	**0.1158**
	K-Means	3	0.0799	0.1043	0.1106	0.1143
	K-Med	3	0.0770	0.1002	0.1062	0.1100
	LT	2	0.0785	0.1025	0.1086	0.1122
	Spar	3.4	0.0763	0.0999	0.1060	0.1097
	LTG	3	0.0806	0.1043	0.1103	0.1139
FT	Baseline	1	0.4264	0.5024	0.5135	0.5036
	K-Means	3	**0.5074**	**0.5870**	**0.5901**	**0.5710**
	K-Med	5	0.4689	0.5493	0.5570	0.5428
	LT	3.2	0.4733	0.5547	0.5582	0.5459
	Spar	11	0.4946	0.5704	0.5743	0.5606
	LTG	4.2	0.4747	0.5562	0.5604	0.5486
Frappe	Baseline	1	0.1739	0.2451	0.2614	0.2622
	K-Means	10	0.2316	0.2876	0.2950	0.2891
	K-Med	10	0.2173	0.2708	0.2779	0.2736
	LT	3.6	0.2353	0.2851	0.2923	0.2871
	Spar	5.4	**0.2613**	**0.3145**	**0.3199**	**0.3127**
	LTG	4	0.2300	0.2902	0.2991	0.2951
LFM	Baseline	1	0.3419	0.4263	0.4305	0.4092
	K-Means	3	0.3411	0.4259	0.4306	0.4105
	K-Med	3	0.3196	0.4018	0.4077	0.3887
	LT	2	**0.3432**	**0.4322**	**0.4362**	**0.4120**
	Spar	18.4	0.3254	0.4102	0.4157	0.3939
	LTG	2	**0.3432**	**0.4322**	**0.4362**	**0.4120**
ML	Baseline	1	0.4359	0.5337	0.5340	0.4982
	K-Means	7	0.4628	0.5627	0.5584	0.5200
	K-Med	3	0.4614	0.5638	0.5603	0.5224
	LT	3	0.4611	0.5628	0.5609	0.5213
	Spar	9.4	**0.4711**	**0.5672**	**0.5644**	**0.5248**
	LTG	3	0.4611	0.5628	0.5609	0.5213
R	Baseline	1	**0.2216**	**0.3238**	**0.3474**	**0.3525**
	K-Means	7	0.1842	0.2741	0.2988	0.3066
	K-Med	3	0.1785	0.2704	0.2939	0.3056
	LT	2	0.2067	0.2985	0.3222	0.3331
	Spar	2	0.2067	0.2985	0.3222	0.3331
	LTG	0.6	0.2104	0.3144	0.3354	0.3422
Steam	Baseline	1	0.2749	0.3249	0.3286	0.3244
	K-Means	10	0.2826	0.3200	0.3221	0.3173
	K-Med	3	0.1317	0.1870	0.2017	0.2154
	LT	2	0.2965	0.3345	0.3374	0.3316
	Spar	43.6	**0.3034**	**0.3373**	**0.3398**	**0.3338**
	LTG	2	0.2965	0.3345	0.3374	0.3316
YMov	Baseline	1	0.3165	0.3995	0.4058	0.3902
	K-Means	10	**0.3219**	**0.4053**	**0.4117**	0.3953
	K-Med	3	0.2958	0.3773	0.3846	0.3709
	LT	2	**0.3219**	**0.4053**	0.4113	**0.3955**
	Spar	37.4	0.2949	0.3760	0.3845	0.3726
	LTG	2	**0.3219**	**0.4053**	0.4113	**0.3955**
YMus	Baseline	1	0.1220	0.1686	0.1822	0.1901
	K-Means	5	**0.1749**	**0.2404**	**0.2545**	**0.2561**
	K-Med	5	0.1696	0.2371	0.2526	0.2556
	LT	5.6	0.1691	0.2331	0.2473	0.2502
	Spar	162.6	0.1479	0.2109	0.2270	0.2326
	LTG	5.6	0.1691	0.2331	0.2473	0.2502

4.5 Analysis and Discussion

Each recommender algorithm responded differently to the use of clustering pre-processing and, in particular, to each of the cluster extraction methods proposed here.

For User-KNN, the improvement achieved in some datasets is possibly caused by a more focused computation of neighborhood within each cluster. For datasets where no improvement

Table 3 – MAP@N of BPR-MF

Database	Method	#Clu	MAP@1	MAP@3	MAP@5	MAP@10
BX	Baseline	1	0.0134	0.0248	0.0289	0.0335
	K-Means	10	0.0165	0.0265	0.0307	0.0352
	K-Med	7	0.0166	0.0265	0.0308	0.0348
	LT	2	0.0175	0.0295	0.0338	0.0381
	Spar	3.4	0.0149	0.0266	0.0305	0.0351
	LTG	3	**0.0211**	**0.0327**	**0.0370**	**0.0413**
FT	Baseline	1	0.4546	0.5422	0.5505	0.5338
	K-Means	5	0.4792	0.5565	0.5607	0.5428
	K-Med	3	0.4641	0.5438	0.5523	0.5378
	LT	3.2	**0.5029**	**0.5775**	**0.5811**	**0.5625**
	Spar	11	0.4764	0.5514	0.5575	0.5441
	LTG	4.2	0.5010	0.5750	0.5789	0.5616
Frappe	Baseline	1	0.1868	0.2652	0.2792	0.2782
	K-Means	10	**0.2451**	**0.3236**	**0.3323**	**0.3245**
	K-Med	10	0.1801	0.2460	0.2566	0.2558
	LT	3.6	0.2151	0.2869	0.2994	0.2960
	Spar	5.4	0.1900	0.2586	0.2707	0.2706
	LTG	4	0.2244	0.2964	0.3051	0.3012
LFM	Baseline	1	0.1691	0.2249	0.2352	0.2310
	K-Means	5	0.2293	0.2907	0.2977	0.2881
	K-Med	10	**0.2352**	**0.3024**	**0.3107**	**0.3006**
	LT	2	0.2106	0.2683	0.2766	0.2702
	Spar	18.4	0.2209	0.2892	0.2989	0.2914
	LTG	2	0.2106	0.2683	0.2766	0.2702
ML	Baseline	1	**0.4433**	**0.5397**	**0.5398**	**0.5042**
	K-Means	5	0.3927	0.5035	0.5079	0.4747
	K-Med	3	0.4181	0.5178	0.5192	0.4865
	LT	3	0.4359	0.5340	0.5334	0.4995
	Spar	9.4	0.3617	0.4712	0.4780	0.4497
	LTG	3	0.4359	0.5340	0.5334	0.4995
R	Baseline	1	**0.1528**	**0.2284**	**0.2560**	**0.2626**
	K-Means	3	0.1449	0.2151	0.2350	0.2461
	K-Med	10	0.1521	0.2274	0.2541	0.2650
	LT	2	0.1303	0.2091	0.2368	0.2466
	Spar	2	0.1303	0.2091	0.2368	0.2466
	LTG	0.6	0.1407	0.2143	0.2495	0.2572
Steam	Baseline	1	0.0990	0.1432	0.1529	0.1592
	K-Means	10	0.0992	0.1390	0.1500	0.1548
	K-Med	10	0.1332	0.1728	0.1814	0.1843
	LT	2	0.0866	0.1328	0.1430	0.1488
	Spar	43.6	**0.2217**	**0.2547**	**0.2605**	**0.2612**
	LTG	2	0.0866	0.1328	0.1430	0.1488
YMov	Baseline	1	0.2222	0.2904	0.3000	0.2918
	K-Means	10	**0.2360**	**0.3042**	**0.3127**	0.3030
	K-Med	10	0.2308	0.2992	0.3084	0.2995
	LT	2	0.2305	0.3011	0.3091	0.3003
	Spar	37.4	0.2282	0.3021	0.3121	**0.3048**
	LTG	2	0.2305	0.3011	0.3091	0.3003
YMus	Baseline	1	0.1563	**0.2297**	**0.2472**	**0.2518**
	K-Means	3	0.1551	0.2207	0.2363	0.2402
	K-Med	3	0.1398	0.2048	0.2217	0.2274
	LT	5.6	**0.1573**	0.2239	0.2400	0.2442
	Spar	162.6	0.1280	0.1861	0.2022	0.2086
	LTG	5.6	**0.157**3	0.2239	0.2400	0.2442

was observed, the benefits of clustering, such as sparsity reduction within each cluster, may have been outweighed by the reduction in the number of users available to each recommender (applied separately to each cluster). BPR-MF, in particular, adopts a global view of the data to optimize matrix factorization, and is apparently the most affected by the reduction in users information within each cluster. In spite of that, the algorithm still benefited from clustering in a number of databases, possibly because the resulting clusters may characterize well defined hubs that favor

Table 4 – MAP@N of Most Popular

Database	Method	#Clu	MAP@1	MAP@3	MAP@5	MAP@10
BX	Baseline	1	0.0235	0.0345	0.0388	0.0433
	K-Means	10	0.0241	0.0357	0.0403	0.0451
	K-Med	3	0.0235	0.0346	0.0388	0.0435
	LT	2	0.0291	0.0414	0.0451	0.0495
	Spar	3.4	0.0289	0.0413	0.0453	0.0498
	LTG	3	**0.0322**	**0.0440**	**0.0482**	**0.0525**
FT	Baseline	1	0.4930	0.5718	0.5755	0.5569
	K-Means	5	0.4965	0.5723	0.5749	0.5558
	K-Med	3	0.4916	0.5684	0.5731	0.5566
	LT	3.2	0.5160	0.5900	0.5938	0.5751
	Spar	11	0.5081	0.5797	0.5822	0.5659
	LTG	4.2	**0.5188**	**0.5914**	**0.5954**	**0.5778**
Frappe	Baseline	1	0.2018	0.2825	0.2918	0.2918
	K-Means	10	**0.2604**	**0.3369**	**0.3446**	**0.3369**
	K-Med	10	0.2227	0.2893	0.2979	0.2934
	LT	3.6	0.2402	0.3151	0.3265	0.3235
	Spar	5.4	0.2296	0.3019	0.3119	0.3107
	LTG	4	0.2523	0.3235	0.3316	0.3269
LFM	Baseline	1	0.1264	0.1681	0.1772	0.1764
	K-Means	3	0.2313	0.2849	0.2901	0.2817
	K-Med	10	0.2649	0.3292	0.3355	0.3220
	LT	2	0.2181	0.2699	0.2767	0.2691
	Spar	18.4	**0.2711**	**0.3387**	**0.3466**	**0.3316**
	LTG	2	0.2181	0.2699	0.2767	0.2691
ML	Baseline	1	0.2710	0.3523	0.3709	0.3551
	K-Means	10	0.3662	0.4512	0.4553	0.4292
	K-Med	10	0.4151	0.5105	0.5100	0.4776
	LT	3	0.4082	0.4974	0.4964	0.4639
	Spar	9.4	**0.4276**	**0.5212**	**0.5212**	**0.4878**
	LTG	3	0.4082	0.4974	0.4964	0.4639
R	Baseline	1	0.1116	0.1603	0.1699	0.1711
	K-Means	7	0.1151	0.1741	0.1900	0.2012
	K-Med	10	**0.1464**	**0.2163**	**0.2403**	**0.2508**
	LT	2	0.1205	0.1687	0.1834	0.1901
	Spar	2	0.1205	0.1687	0.1834	0.1901
	LTG	0.6	0.0986	0.1508	0.1618	0.1659
Steam	Baseline	1	0.0985	0.1432	0.1539	0.1604
	K-Means	10	0.1020	0.1415	0.1536	0.1588
	K-Med	10	0.0622	0.0960	0.1093	0.1236
	LT	2	0.0892	0.1375	0.1481	0.1543
	Spar	43.6	**0.2450**	**0.2833**	**0.2901**	**0.2878**
	LTG	2	0.0892	0.1375	0.1481	0.1543
YMov	Baseline	1	0.2218	0.2910	0.2978	0.2894
	K-Means	10	0.2444	0.3134	0.3211	0.3102
	K-Med	10	0.2448	0.3110	0.3183	0.3076
	LT	2	0.2320	0.3047	0.3139	0.3046
	Spar	37.4	**0.2458**	**0.3180**	**0.3277**	**0.3187**
	LTG	2	0.2320	0.3047	0.3139	0.3046
YMus	Baseline	1	0.1207	0.1753	0.1936	0.2047
	K-Means	3	0.1315	0.1902	0.2051	0.2116
	K-Med	10	0.1300	0.1872	0.2024	0.2094
	LT	5.6	**0.1478**	**0.2109**	**0.2264**	**0.2328**
	Spar	162.6	0.1409	0.2032	0.2196	0.2261
	LTG	5.6	**0.1478**	**0.2109**	**0.2264**	**0.2328**

better matrix factorization optimization.

Relatively speaking, the recommender that benefited the most from clustering was the Most Popular algorithm. This is because clustering generates small groups of like-minded users, indirectly resulting in personalized (rather than completely unpersonalized) recommendations.

The results obtained by the proposed methods improved the results of a recommender

in most of the experiments. Also, in almost all datasets, our approach provided results that are competitive or better than other clustering pre-processing techniques for at least one of the recommender algorithms tested.

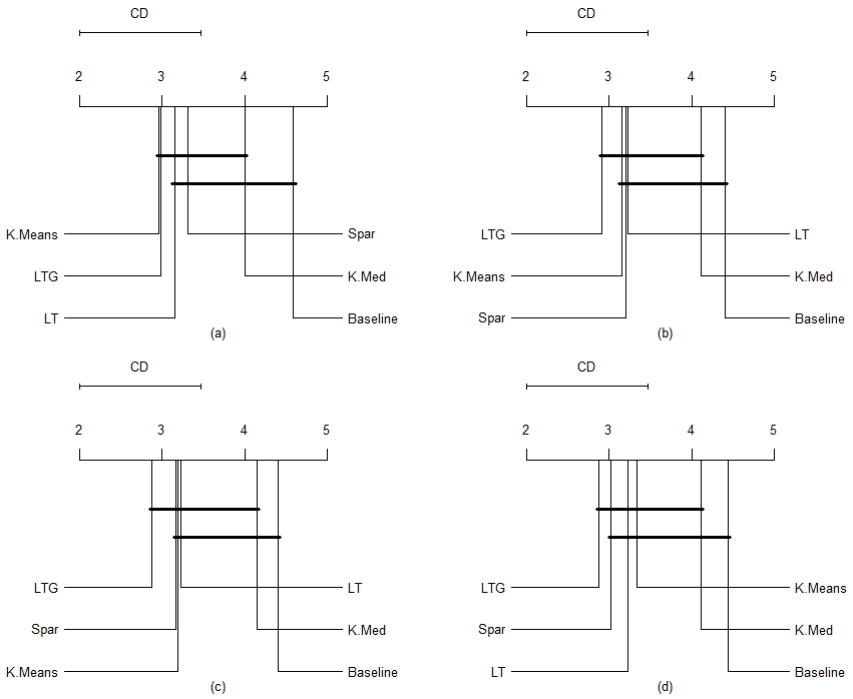


Figure 5 – This figure plots the average rank of each pre-processing methods and regards statistical differences between them, if a pair of algorithms are joined by a line it means that there is no statistical difference between them with 95% confidence. Each subfigure presents the results for a different N in MAP@N, (a) MAP@1; (b) MAP@3; (c) MAP@5 and (d) MAP@10.

Regarding the statistical analysis, as can be seen by the post-hoc test held in Figure 5, LifetimeMCSGlobal is the only approach that significantly improved the baselines. For top@1 recommendations, either LifetimeMCSGlobal or K-Means provided better overall results, as can be seen in Figure 5 (a), although as recommendation lists become larger, the quality of K-Means decreases and SparsityMCS increases. As the opposite, LifetimeMCS maintains its quality for any recommendation list size.

It is worth mentioning that for this statistical analysis, only the best configurations for K-Means and K-Medoids were provided, meaning that this discussion has meaning only when considering that the best parameter was chosen for each dataset. In addition, unlike our competitors, our approach automatically determines the number of clusters, which is a byproduct of our cluster extraction procedure. In contrast, for the competitors, which take the number of clusters as a parameter, the best choice was not universal across different databases or even for different algorithms in the same database. For instance, for database LFM, K-Means produced best results using $K = 3$ for User-KNN, $K = 7$ for BPRMF, and $K = 10$ for Most Popular. Determining the best number of cluster as an input parameter imposes additional computational load to the recommender phase.

Even though LifetimeMCSGlobal has achieved better results when compared to LifetimeMCS and SparsityMCS, there is no unique winner. On the one hand, LifetimeMCS is faster ($O(|U|)$) than SparsityMCS ($O(|U| \cdot |I|)$ and it tends to be better at top@1 recommendations, which is a desired feature in scenarios where it is expected that the user consumes only one or two products, e.g. cars and houses sales. On the other hand, SparsityMCS provides better recommendations at longer lists, being more suitable for other scenarios such as music recommendation, where a user may be able listen to 10 or 20 new songs in a single day. Thus, both approaches seem to perform quite similarly in terms of medium sized lists and since computing the clustering hierarchy needs to be done only once and is computationally more demanding than extracting clusters with these measures, the analyst can try both at the price of a relatively small additional runtime.

CHAPTER

5

FINAL REMARKS

In this chapter we address the final remarks, the main contributions and the publications resulted of this work, in addition to some ideas for future works.

5.1 Work Overview

Clustering has shown to be an useful tool for data pre-processing prior to the application of recommender systems. However, the clustering process is oftentimes bound to a specific recommender algorithm and tackles the problem of data sparsity indirectly only. In this work, three variants of an approach to automatically extract clusters from a hierarchy of candidates, which are specifically focused on RS problems, have been proposed: two by selecting more stable clusters (LifetimeMCS and LifetimeMCSGlobal) and another by directly minimizing data sparsity within clusters (SparsityMCS). The proposed methods require only, but is not limited to, a collection of user-item interactions and can be subsequently applied to different recommender systems.

The obtained results were evaluated using three different recommender systems across nine publicly known databases, and then compared with those provided by two other clustering-based competitors and the classical approach, i.e. no clustering. The proposed methods improved the results of a recommender in most of the experiments and, in almost all datasets, our approach provided results that are competitive or better than the clustering competitors for at least one of the recommender algorithms tested. Unlike our competitors, our approach automatically determines the number of clusters, which is a byproduct of our cluster extraction procedure and also ensures a minimal amount of information for the recommendation to be applied subsequently.

Given the results and statistical analysis, we validate our hypothesis, that the overall quality of recommendations is improved when applied after a guided disjunction of the datasets. Although the obtained results are not global, as can be noted by some individual results, they are

better than competitors in most cases, with 95% confidence.

More than that, the study conducted helps to understand better the behavior of some recommender techniques and the impact that different domains and database sizes has on final recommendations.

5.2 Publications

The research conducted provided content for three publications, the first is a Poster already accepted that will be presented at the 12th ACM Conference on Recommender Systems (RecSys2018)[1] in the first week of October, it presents a confidence-based technique to filter interactions generating local recommendations from a global approach; the second one is also accepted at RecSys2018, although it is a Demo presenting a framework that gathers many recommender tools and algorithms; the last one was submitted and still being reviewed in the ACM Transactions on Information Systems (ACM TOIS journal)[2], it presents the techniques described in this work, which are the final and main contributions of this research.

- **Type:** Conference Poster at RecSys2018.

  Status: Accepted. To be presented in October.

  Title: CoBaR: Confidence-Based Recommender.

  Authors: de Aguiar Neto, Fernando S.; C. Fortes, Arthur; Manzato, Marcelo G.

  Abstract: Neighborhood-based collaborative filtering algorithms usually adopt a fixed neighborhood size for every user or item, although groups of users or items may have different lengths depending on users' preferences. In this paper, we propose an extension to a non-personalized recommender based on confidence intervals and hierarchical clustering to generate groups of users with optimal sizes. The evaluation shows that the proposed technique outperformed the traditional recommender algorithms in four publicly available datasets.

- **Type:** Conference Demo at RecSys2018.

  Status: Accepted. To be presented in October.

  Title: Case Recommender: A Flexible and Extensible Python Framework for Recommender Systems.

  Authors: C. Fortes, Arthur; Fressato, Eduardo; de Aguiar Neto, Fernando S.; Manzato, Marcelo G.; Campello, Ricardo.

  Abstract: This paper presents a polished open-source Python-based recommender framework named Case Recommender, which provides a rich set of components from which

[1] https://recsys.acm.org/recsys18/
[2] https://tois.acm.org

developers can construct and evaluate customized recommender systems. It implements well-known and state-of-the-art algorithms in rating prediction and item recommendation scenarios. The main advantage of the Case Recommender is the possibility to integrate clustering and ensemble algorithms with recommendation engines, easing the development of more accurate and efficient approaches.

- **Type:** Journal at TOIS.

  Status: Submitted in 25 of July, 2018. Being reviewed.

  Title: Pre-Processing Approaches for Collaborative Filtering Based on Hierarchical Clustering.

  Authors: de Aguiar Neto, Fernando S.; Fortes C., Arthur; Manzato, Marcelo G.; Campello, Ricardo.

  Abstract: Recommender Systems (RS) support users to find relevant content, such as movies, books, songs, and other products based on their preferences. Such preferences are gathered by analyzing past users' interactions, however, data collected for this purpose are typically prone to sparsity and high dimensionality. Clustering-based techniques have been proposed to handle these problems effectively and efficiently by segmenting the data into a number of similar groups based on predefined characteristics. Although these techniques have gained increasing attention in the recommender systems community, they are usually bound to a particular recommender system and/or require critical parameters, such as the number of clusters. In this paper, we present three variants of a general-purpose method to optimally extract users' groups from a hierarchical clustering algorithm specifically targeting RS problems. The proposed extraction methods do not require critical parameters and can be applied prior to any recommendation system. Our experiments have shown promising recommendation results in the context of nine well-known public datasets from different domains.

5.3 Future Work

For future work, we aim to experiment the use of different types of information for both cluster extraction and recommendation, e.g. using personal information to cluster users and interaction information to compute recommendations. In addition, we plan to further investigate the use of other hierarchical clustering algorithms with our extraction methods.

Another possible approach would be to apply semi-supervised active-clustering in order to generate a flat partition extraction, active clustering permits a system to generate queries that may be answered by each user in order to refine the clustering, RS provides a fertile ground for such technique given that is expected that the user interacts with a little amount of items, active clustering approach may select the interactions that better help the clustering task.

BIBLIOGRAPHY

ADOMAVICIUS, G.; BOCKSTEDT, J.; CURLEY, S.; ZHANG, J. Understanding effects of personalized vs. aggregate ratings on user preferences. **CEUR Workshop Proceedings**, v. 1679, p. 14–21, 1 2016. ISSN 1613-0073. Citation on page 26.

ADOMAVICIUS, G.; MOBASHER, B.; RICCI, F.; TUZHILIN, A. Context-aware recommender systems. p. 67–80, 2011. Citation on page 25.

AGGARWAL, C. C. **Recommender systems: The Textbook**. Cham: Springer, 2016. 498 p. ISSN 0001-0782. ISBN 9783319296573. Citations on pages 19, 23, and 49.

AMATRIAIN, X.; PUJOL, J. M.; OLIVER, N. I like it... i like it not: Evaluating user ratings noise in recommender systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 5535 LNCS, p. 247–258, 2009. Citation on page 24.

BAGLA, P. ANIL KAKODKAR INTERVIEW: Breaking Up (a Nuclear Program) Is Hard to Do. **Science**, v. 311, n. 5762, p. 765–766, feb 2006. ISSN 0036-8075. Available: <http://portal.acm.org/citation.cfm?id=SERIES10022.42779http://www.sciencemag.org/cgi/doi/10.1126/science.311.5762.765>. Citations on pages 21 and 28.

BALTRUNAS KAREN CHURCH, A. K. N. O. L. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. 2015. Citation on page 48.

BILGE, A.; POLAT, H. A comparison of clustering-based privacy-preserving collaborative filtering schemes. **Applied Soft Computing**, v. 13, n. 5, p. 2478 – 2489, 2013. ISSN 1568-4946. Available: <http://www.sciencedirect.com/science/article/pii/S1568494612005303>. Citations on pages 20 and 49.

BOBADILLA, J.; ORTEGA, F.; HERNANDO, A.; GUTIÉRREZ, A. Recommender systems survey. **Knowledge-Based Systems**, Elsevier B.V., v. 46, p. 109–132, 2013. Citations on pages 19, 23, 24, and 25.

BURKE, R. Hybrid web recommender systems. **The adaptive web**, p. 377–408, 2007. Citations on pages 19 and 24.

CAMPELLO, R. J. G. B.; MOULAVI, D.; ZIMEK, A.; SANDER, J. A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. **Data Mining and Knowledge Discovery**, v. 27, n. 3, p. 344–371, 2013. ISSN 13845810. Citations on pages 21, 29, 30, 31, 32, 39, 40, and 43.

CANTADOR, I.; BRUSILOVSKY, P.; KUFLIK, T. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In: **Proceedings of the 5th ACM conference on Recommender systems**. New York, NY, USA: ACM, 2011. (RecSys 2011). Citation on page 48.

CONNOR, M.; HERLOCKER, J. **Clustering Items for Collaborative Filtering**. 2001. Citations on pages 20, 33, and 34.

COSTA, A. F. da; MANZATO, M. G. Case recommender: A recommender framework. **In: Workshop de Teses e Dissertações (WTD), 16., 2016, Teresina. Anais do XXII Simpósio Brasileiro de Sistemas Multimídia e Web. Porto Alegre: Sociedade Brasileira de Computação, 2016. v. 2.**, 2016. Citation on page 49.

COSTA, A. F. da; MANZATO, M. G.; CAMPELLO, R. J. Group-based Collaborative Filtering Supported by Multiple Users' Feedback to Improve Personalized Ranking. In: **Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web - Webmedia '16**. New York, New York, USA: ACM Press, 2016. p. 279–286. ISBN 9781450345125. Available: <http://dl.acm.org/citation.cfm?doid=2976796.2976852>. Citations on pages 20, 33, 34, 47, and 49.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **Journal of Machine learning research**, v. 7, n. Jan, p. 1–30, 2006. Citation on page 50.

FITZGIBBON, L. J.; ALLISON, L.; DOWE, D. L. Minimum message length grouping of ordered data. In: ARIMURA, H.; JAIN, S.; SHARMA, A. (Ed.). **Algorithmic Learning Theory: 11th International Conference, ALT 2000 Sydney, Australia, December 11–13, 2000 Proceedings**. Berlin, Heidelberg: Springer, 2000. p. 56–70. ISBN 978-3-540-40992-2. Available: <http://dx.doi.org/10.1007/3-540-40992-0_5>. Citation on page 30.

GAN, G.; MA, C.; WU, J. **Data Clustering: Theory, Algorithms, and Applications**. Berlin, Heidelberg: Springer, 2007. 466 p. Citations on pages 26 and 28.

GANTNER, Z.; RENDLE, S.; FREUDENTHALER, C.; SCHMIDT-THIEME, L. MyMediaLite: A free recommender system library. In: **Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)**. New York, NY, USA: ACM, 2011. Citation on page 49.

GUO, G.; ZHANG, J.; YORKE-SMITH, N. A novel bayesian similarity measure for recommender systems. In: **Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence**. AAAI Press, 2013. (IJCAI '13), p. 2619–2625. ISBN 978-1-57735-633-2. Available: <http://dl.acm.org/citation.cfm?id=2540128.2540506>. Citation on page 48.

HARPER, F. M.; KONSTAN, J. A. The movielens datasets: History and context. **ACM Trans. Interact. Intell. Syst.**, ACM, New York, NY, USA, v. 5, n. 4, p. 19:1–19:19, Dec. 2015. ISSN 2160-6455. Available: <http://doi.acm.org/10.1145/2827872>. Citation on page 48.

HOULE, M. E.; KRIEGEL, H. P.; KRÖGER, P.; SCHUBERT, E.; ZIMEK, A. Can shared-neighbor distances defeat the curse of dimensionality? In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**. Berlim, Heidelberg: Springer, 2010. v. 6187 LNCS, p. 482–500. Citation on page 28.

JAWAHEER, G.; SZOMSZOR, M.; KOSTKOVA, P. Comparison of implicit and explicit feedback from an online music recommendation service. **Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems - HetRec '10**, p. 47–51, 2010. Citation on page 24.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: **Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (IJCAI'95), p. 1137–1143. ISBN 1-55860-363-8. Available: <http://dl.acm.org/citation.cfm?id=1643031.1643047>. Citation on page 49.

KOREN, Y. Factor in the neighbors: Scalable and accurate collaborative filtering. **ACM Transactions on Knowledge Discovery from Data**, v. 4, n. 1, p. 1–24, 2010. Citation on page 25.

KUMAR, A.; KUMAR, N.; HUSSAIN, M.; CHAUDHURY, S.; AGARWAL, S. Semantic clustering-based cross-domain recommendation. In: **2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)**. Orlando, FL, USA: IEEE, 2014. p. 137–141. Citation on page 48.

LI, Q. L. Q.; KIM, B. M. K. B. M. Clustering approach for hybrid recommender system. **Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)**, IEEE, Halifax, NS, Canada, n. November 2003, p. 33–38, 2003. Citations on pages 20, 33, and 34.

NAJAFABADI, M. K.; MAHRIN, M. N.; CHUPRAT, S.; SARKAN, H. M. Improving the accuracy of collaborative filtering recommendations using clustering and association rules mining on implicit data. **Computers in Human Behavior**, v. 67, p. 113 – 128, 2017. ISSN 0747-5632. Available: <http://www.sciencedirect.com/science/article/pii/S0747563216307518>. Citations on pages 20, 25, 32, 33, and 34.

PARK, H.-S.; LEE, J.-S.; JUN, C.-H. A k-means-like algorithm for k-medoids clustering and its performance. **Proceedings of ICCIE**, Citeseer, p. 102–117, 2006. Citation on page 28.

PEREIRA, A. L. V.; HRUSCHKA, E. R. Simultaneous co-clustering and learning to address the cold start problem in recommender systems. **Knowledge-Based Systems**, v. 82, p. 11–19, 2015. Citations on pages 20 and 34.

RENDLE, S.; FREUDENTHALER, C.; GANTNER, Z.; SCHMIDT-THIEME, L. BPR: bayesian personalized ranking from implicit feedback. **CoRR**, abs/1205.2618, 2012. Available: <http://arxiv.org/abs/1205.2618>. Citation on page 26.

RICCI, F.; ROKACH, L.; SHAPIRA, B.; KANTOR, P. B.; RICCI, F. **Recommender Systems Handbook**. Boston, MA: Springer, 2011. Citations on pages 19, 20, and 25.

SHINDE, S. K.; KULKARNI, U. Hybrid personalized recommender system using centering-bunching based clustering algorithm. **Expert Systems with Applications**, v. 39, n. 1, p. 1381 – 1387, 2012. ISSN 0957-4174. Available: <http://www.sciencedirect.com/science/article/pii/S0957417411011316>. Citations on pages 20, 33, and 34.

SUNANDA, P.; VINEELA, A. An agglomerative hierarchical clustering for hybrid recommender systems. In: **2015 Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG)**. Orlando, FL, USA: IEEE, 2015. p. 283–288. Citations on pages 20, 33, and 34.

UNGAR, L.; FOSTER, D. Clustering methods for collaborative filtering. In: **AAAI workshop on recommendation systems**. USA: AAAI Press, 1998. Citations on pages 20 and 32.

VARGAS-GOVEA, B.; GONZÁLEZ-SERNA, G.; PONCE-MEDELLIN, R. Effects of relevant contextual features in the performance of a restaurant recommender system. In: **3rd International Workshop on Context-Aware Recommender Systems**. Chicago, IL, USA: ACM, 2011. Citation on page 48.

VLACHOS, M.; FUSCO, F.; MAVROFORAKIS, C.; KYRILLIDIS, A.; VASSILIADIS, V. G. Improving co-cluster quality with application to product recommendations. In: ACM. **Proceedings of the 23rd ACM International Conference on Information and Knowledge Management**. New York, NY, USA, 2014. p. 679–688. Citations on pages 20, 32, 34, and 47.

WARD, J. H. **Hierarchical grouping to optimize an objective function**. 1963. 236–244 p. Citation on page 28.

ZAHRA, S.; GHAZANFAR, M. A.; KHALID, A.; AZAM, M. A.; NAEEM, U.; PRUGEL-BENNETT, A. Novel centroid selection approaches for kmeans-clustering based recommender systems. **Information Sciences**, v. 320, n. Supplement C, p. 156 – 189, 2015. ISSN 0020-0255. Available: <http://www.sciencedirect.com/science/article/pii/S0020025515002352>. Citations on pages 20, 28, 32, 33, 34, and 47.

ZIEGLER, C.-N.; MCNEE, S. M.; KONSTAN, J. A.; LAUSEN, G. Improving recommendation lists through topic diversification. In: **Proceedings of the 14th International Conference on World Wide Web**. New York, NY, USA: ACM, 2005. (WWW '05), p. 22–32. ISBN 1-59593-046-9. Available: <http://doi.acm.org/10.1145/1060745.1060754>. Citation on page 48.


ICMC
USP
SÃO CARLOS