UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Pedro Mariano Sousa Bezerra

Autoencoder-based Pattern Mining Applied to Recommender Systems

Autoencoders para Mineração de Padrões Aplicados em Sistemas de Recomendação

Campinas

2025

Pedro Mariano Sousa Bezerra

Autoencoder-based Pattern Mining Applied to Recommender Systems

Autoencoders para Mineração de Padrões Aplicados em Sistemas de Recomendação

Thesis presented to the School of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering, in the area of Computer Engineering.

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor/Orientador: Prof. Dr. Fernando José Von Zuben

Este exemplar corresponde à versão final da tese defendida pelo aluno Pedro Mariano Sousa Bezerra, e orientada pelo Prof. Dr. Fernando José Von Zuben .

Campinas

2025

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca da Área de Engenharia e Arquitetura
Vanessa Evelyn Costa - CRB 8/8295

Bezerra, Pedro Mariano Sousa, 1990-
B469a Autoencoder-based pattern mining applied to recommender systems / Pedro Mariano Sousa Bezerra. – Campinas, SP : [s.n.], 2025.

Orientador: Fernando José Von Zuben.
Tese (doutorado) – Universidade Estadual de Campinas (UNICAMP), Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas de recomendação (Filtragem da informação). 2. Filtragem colaborativa. 3. Autoencoders. 4. Redes neurais (Computação). 5. Mineração de dados (Computação). I. Von Zuben, Fernando José, 1968-. II. Universidade Estadual de Campinas (UNICAMP). Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações complementares

Título em outro idioma: Autoencoders para mineração de padrões aplicados em sistemas de recomendação
Palavras-chave em inglês:
Recommender systems
Collaborative filtering
Autoencoders
Neural networks
Data mining
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora:
Fernando José Von Zuben [Orientador]
Denis Gustavo Fantinato
Emely Pujólli da Silva
Marcelo Garcia Manzato
Leonardo Chaves Dutra da Rocha
Data de defesa: 23-05-2025
Programa de Pós-Graduação: Engenharia Elétrica

Objetivos de Desenvolvimento Sustentável (ODS)
ODS: 3. Saúde e bem-estar

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-1491-145X
- Currículo Lattes do autor: http://lattes.cnpq.br/6733367490243236

Prof. Dr. Fernando José Von Zuben (Presidente)

Prof. Dr. Denis Gustavo Fantinato

Dra. Emely Pujólli da Silva

Prof. Dr. Marcelo Garcia Manzato

Prof. Dr. Leonardo Chaves Dutra da Rocha

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

To my dearest father, who will be watching from the heavens.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Prof. Fernando José Von Zuben, for the guidance, the patience, and the valuable contributions during the process of becoming a doctor.

I would like to thank as well my mother for the continuous and unconditional support, and my colleagues of the Laboratory of Bioinformatics and Bioinspired Computing (LBiC) and the Viva Bem group for the academic and non-academic discussions.

Finally, I would like to acknowledge the organizations that funded this research. Part of the results presented in this thesis were carried out as part of the project "*Hub de Inteligência Artificial em Saúde e Bem-Estar – VIVA BEM*", funded by *Samsung Eletrônica da Amazônia Ltda*, under the terms of IT Law (Federal Law 8.248/91). This work was also partially supported by the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (grant 157999/2019-0).

"The greatest challenge to any thinker is stating the problem in a way that will allow a solution."

(Bertrand Russell)

Resumo

Filtragem Colaborativa é uma das técnicas mais utilizadas em Sistemas de Recomendação (RS - *Recommender Systems*) e pode ser considerada uma estratégia competitiva na geração de recomendações para usuários. Entretanto, alguns desafios limitam a eficiência de técnicas baseadas em Filtragem Colaborativa na medida em que o conjunto de dados aumenta, como a esparsidade dos dados e um aumento significativo na demanda por recursos computacionais. Para tratar estas questões, técnicas de mineração de dados têm sido aplicadas para encontrar partições mais informativas do conjunto de dados, capazes de reduzir efetivamente o custo computacional. Nesta tese, nós propomos um método de filtragem colaborativa baseado em *autoencoders*, denominado APCF, do inglês *Autoencoder Pattern-based Collaborative Filtering*, um novo tipo de aplicação de autoencoder (AE) em RS. Diferentemente dos paradigmas tradicionais que usam autoencoders para extrair representações latentes ou diretamente reconstruir avaliações, o APCF utiliza um autoencoder como técnica de mineração de dados. Com a ajuda do método *BinaPs*, um AE eficiente para mineração de padrões, obtém-se uma partição informativa do conjunto de dados. Então, por meio de uma abordagem de K-vizinhos mais próximos baseada em itens (IBKNN - *Item-Based K-Nearest Neighbors*), chega-se à predição das notas baseada na similaridade entre itens pertencentes a uma mesma partição dos dados. Foram conduzidos experimentos comparativos entre o método proposto e outros sistemas de recomendação em diferentes conjuntos de dados. De modo geral, o APCF apresentou resultados comparáveis aos dos métodos SVD++ e NMF e superiores aos do método de referência IBKNN. Em um conjunto grande de dados sintéticos projetado para exibir alguns comportamentos regulares, o APCF apresentou uma grande vantagem em escalabilidade, obtendo os melhores resultados entre os métodos que foram capazes de executar a tarefa. O potencial do método APCF foi também avaliado ao ser aplicado em recomendações na área da saúde em dois contextos diferentes: atividade física e diagnóstico de sarcopenia. Apesar de ter sido concebido para uso em grandes conjuntos de dados, o APCF apresentou resultados semelhantes aos dos concorrentes em um problema do mundo real com poucos dados. Os resultados corroboram a consistência, escalabilidade e robustez do método APCF em fornecer recomendações de alta qualidade em diferentes cenários, e demonstram a eficiência de uma nova abordagem para aproveitar a capacidade de autoencoders para solucionar problemas recorrentes no desenvolvimento de sistemas de recomendação.

Palavras-chaves: sistemas de recomendação, filtragem colaborativa, *autoencoders* para mineração de padrões.

Abstract

Collaborative Filtering (CF) is the most common approach for Recommender Systems (RS) and can be considered a competitive strategy to provide personalized recommendations to users. However, some challenges limit the effectiveness of Collaborative Filtering techniques as data volume escalates, such as data sparsity and a significant increase in demand for computational resources. To help alleviate these issues, data mining techniques have been applied to properly find more informative dataset partitions, which are capable of effectively reducing the computational burden. Here, we propose an *Autoencoder Pattern-based Collaborative Filtering* (APCF) method, a novel kind of autoencoder (AE) application in RS. Unlike traditional paradigms that use autoencoders to extract latent factors or to directly reconstruct ratings, APCF employs an autoencoder as a data mining technique. With the help of the *BinaPs* algorithm, an efficient AE technique for pattern set mining, we find an informative dataset partition. Then, we adopt an *Item-Based K-Nearest Neighbors* (IBKNN) approach to predict ratings based on the similarity between items within each partition. We carried out comparative experiments involving our method and other recommender systems on benchmark and synthetic datasets. Overall, our method performed similarly to the SVD++ and NMF methods and outperformed the baseline IBKNN. On a large synthetic dataset designed to exhibit some regular behaviors, APCF demonstrated a significant advantage in scalability, outperforming all competitors that could run the task. We also evaluated APCF's applicability for health recommendations in two different contexts: physical activity and sarcopenia diagnosis. Despite being intended for use on large datasets, APCF performed similarly to the contender methods in real-world problems with reduced-size data. The results corroborate our method's consistency, scalability, and robustness in providing high-quality recommendations in different scenarios, and demonstrate the effectiveness of a novel approach to leveraging the power of autoencoders in addressing persistent challenges in recommender systems.

Keywords: recommender systems, collaborative filtering, autoencoder-based pattern mining.

List of Figures

List of Tables

List of Nomenclature

ACF Autoencoder-based Collaborative Filtering

AE Autoencoder

APCF Autoencoder Pattern-based Collaborative Filtering

ASM Appendicular Skeletal Muscle Mass

BBCF Bicluster-Based Collaborative Filtering

BinaPs Binary Pattern Networks

CAE Contractive Autoencoder

CBF Content-Based Filtering

CDAE Collaborative Denoising Auto-Encoder

CF Collaborative Filtering

CFN Collaborative Filtering Neural Network

CKE Collaborative Knowledge Base Embedding

CNN Convolutional Neural Networks

CVAE Collaborative Variational Autoencoder

DAE Denoising Autoencoder

DCF Deep Collaborative Filtering

DCG Discounted Cumulative Gain

IBKNN Item-Based K-Nearest Neighbors

KB Knowledge Base

MAE Mean Absolute Error

MDAE Marginalized Denoising Autoencoder

MF Matrix Factorization

MSDAE Marginalized Stacked Denoised Autoencoder

NDCG Normalized Discounted Cumulative Gain

NMF Non-negative Matrix Factorization

NSS Nearest Neighbor Setup

PB Preprocessing Bias

PMF Probabilistic Matrix Factorization

QUBIC QUalitative BIClustering algorithm

RMSE Root Mean Squared Error

RS Recommender System

RSRDL Recommendation with Social Relationships via Deep Learning

SDAE Stacked Denoising Autoencoder

SMM Skeletal Muscle Mass

SPE Subjective Perception of Effort

SVD Singular Value Decomposition

TB Training Bias

TUG Timed-Up and Go

UBKNN User-based K-Nearest Neighbors

USBCF User-Specific Bicluster-based Collaborative Filtering

VAE Variational Autoencoder

Contents

1 Introduction

The amount of available information is rapidly increasing with the development of the web and the continued increase in the storage capacity of devices. Everywhere, people are compelled to decide among abundant options for products, services, activities, textual content, etc., and finding the most promising choice can be challenging and time-consuming. Many companies are committed to developing solutions to help their customers find items of interest (ILIC; KABILJO, 2015).

Recommender systems (RS) are tools devised to produce personalized recommendations to users, aiming to draw their attention to items of interest. These algorithms usually predict how a user would rate a given item. The top-rated predictions are used to generate personalized recommendations. Among several attempts in the literature, there are two main approaches to building such a method: *Content-Based Filtering* (CBF) and *Collaborative Filtering* (CF). CBF approaches explore item features to generate item recommendations based on the user's previous actions or explicit feedback (Google Machine Learning Education, 2022). CF approaches use all available ratings to make predictions based on the user-item interactions. CF methods often outperform CBF approaches, require much less detailed information from users and items, and are the preferred choice (STRUB *et al.*, 2016).

Furthermore, RS can be classified into model-based and memory-based approaches (BEHERA; NAIN, 2022). The former employs a model learned from the dataset using tools from different domains, such as graph theory, statistics, matrix factorization, and machine learning. On the other hand, memory-based approaches use the entire user's rating history to find similarities between users and/or items and generate recommendations. Neighborhood-based models are common in memory-based CF approaches (SARWAR *et al.*, 2001). These methods employ a metric to evaluate the similarity between all users or items, and build user/item neighborhoods formed by similar users/items. The predicted rating is calculated as a weighted average of the neighbors' ratings, with the weights given by the similarities.

Many companies from different business sectors are adopting recommender systems because good recommendations help improve user satisfaction, retain customers, and increase sales (ILIC; KABILJO, 2015; HARDESTY, 2019). However, RS applications are not limited to these cases. In recent years, these methods have been applied in health informatics and medicine, targeting health professionals and patients as end-users (VALDEZ *et al.*, 2016). Health Recommender Systems can help in the decision-making process in health care services, and potentially improve the usability of health care devices.

Since RS reduces the amount of information generated by these devices, their acceptance is continually increasing.

Among the issues commonly found in the development of recommender systems, we can mention sparsity and scalability (KUMAR; SHARMA, 2013). Usually, RS datasets are sparse, meaning they have a small portion of rated items per user compared to the total number of available entries. This creates a challenge for the prediction task and degrades the method's accuracy. The scalability problem refers to the fact that RSs tend to perform worse as the dataset's size increases. In neighborhood-based methods, computing similarities among every pair of users/items becomes increasingly time-consuming.

To alleviate these issues, data processing is an important step in: (1) properly partitioning the whole dataset; (2) extracting useful implicit information; and (3) mining local patterns to support recommendations. Mining techniques, such as biclustering (SINGH; MEHROTRA, 2018), have been applied in collaborative filtering to find dataset partitions containing groups of similar users and items. This way, computational resources are spared since calculations can be more efficiently performed within a reduced and highly informative subset of the whole dataset.

Other promising approaches for finding useful data representation are founded on *Autoencoders* (AE) (KRAMER, 1991). These neural network models are formed by concatenating an encoder, which is responsible for producing an informative encoding of the input data, and a decoder, which aims to retrieve the original data from the encoding. The models are trained to reconstruct the input data and implicitly learn a compact encoding representation. The representations obtained by autoencoders can be interpreted as nonlinear principal components, thus effectively capturing the input dataset's manifold (space of low dimension containing the essence of all variability) (GLOROT; BENGIO, 2010).

The application of autoencoders within RS is not new (ZHANG *et al.*, 2020). Many works in the literature have explored the use of autoencoders, typically following two main paradigms to calculate rating predictions: either directly using the reconstructed output as the predicted ratings or using the representations obtained by the autoencoder in latent factor-based models.

This research focuses on analyzing a novel application of autoencoders in the development of recommender systems. Instead of using traditional dimensionality reduction strategies, we propose here the use of AEs as mining techniques to identify highly informative data partitions that enhance the characterization of user-item relations, thereby improving the recommender system's performance and scalability. With this in mind, we propose a new method called *Autoencoder Pattern-based Collaborative Filtering* (APCF). With the help of the *BinaPs* algorithm (FISCHER; VREEKEN, 2021), an efficient autoencoder technique for pattern set mining, we can obtain a highly informative dataset

partition. Then, we adopt an *Item-Based K-Nearest Neighbors* (IBKNN) (SARWAR *et al.*, 2001) approach to predict ratings based on the similarity between items within each partition. We further compare the performance of our method with that of other RS approaches in the literature across various recommendation scenarios.

The four questions guiding the research are:

- **RQ1.** Can autoencoders be successfully applied as mining techniques to support and potentially enhance the performance of recommender systems?
- **RQ2.** How do the hyperparameters affect APCF's performance?
- **RQ3.** How well does APCF perform compared to other recommender systems?
- **RQ4.** Which application scenarios are most suited for APCF?

The first chapters of this thesis are dedicated to discussing the background concepts for our work: autoencoders and recommender systems. Chapter 2 introduces neural networks, autoencoders, and the BinaPs method. In Chapter 3, we discuss recommender systems, their common approaches, and some of the issues encountered in their development. Chapter 4 presents related work and describes our proposed technique. Chapter 5 summarizes the computational experiments performed to investigate the research questions. The final chapter draws a conclusion and some ideas for future work.

2 Autoencoders

This chapter introduces autoencoders, a particular kind of neural network. We start with a brief introduction to neural networks, and then we present the BinaPs algorithm, an autoencoder-based pattern set mining technique that will be used in our proposed method.

2.1 Neural Networks

Neural networks are mathematical models consisting of a combination of multiple basic units called neurons. A neuron, in turn, is a function given by:

$$y = f(w^T \cdot x + b), \tag{2.1}$$

where $x \in \mathbb{R}^m$ is the array of input data, $y \in \mathbb{R}$ is the output, $w \in \mathbb{R}^m$ is the array of weights, $b \in \mathbb{R}$ is the bias value and $f : \mathbb{R} \to \mathbb{R}$ is an activation function. A neuron can be seen as the evaluation of the activation function at the result of the inner product between the weights and the input, added to the bias. The bias value is often not mentioned explicitly and is considered an extra weight.

A neural network is formed by cascading layers of nodes. A layer is a group of nodes connected to the same input. Each node represents a neuron with different weights, thus producing different output values. Typically, the same activation function is used for all neurons from a given layer, but different activation functions may be adopted. The input layer is formed by sensor neurons responsible for distributing all the input signals to all the neurons in the first inner layer. In a feed-forward network, the outputs of the neurons from a given layer are treated as inputs for the neurons of the following layer. On the other hand, recurrent networks are characterized by the existence of connections between the output of neurons from a given layer and the input of neurons from previous layers, creating feedback loops inside the structure. The final layer is called the output layer; the previous layers are the inner layers. Fig. 1 shows an example of a feed-forward neural network structure with a 3-dimensional input layer, an inner layer with five neurons, and a 2-dimensional output layer.

The universal approximation theorem establishes that feed-forward neural networks are universal function approximators, which implies that, given a sufficient number of neurons, the network can approximate any continuous function over compact subsets of $\mathbb{R}^n$ with an approximation error as small as desired (HORNIK *et al.*, 1989). This property

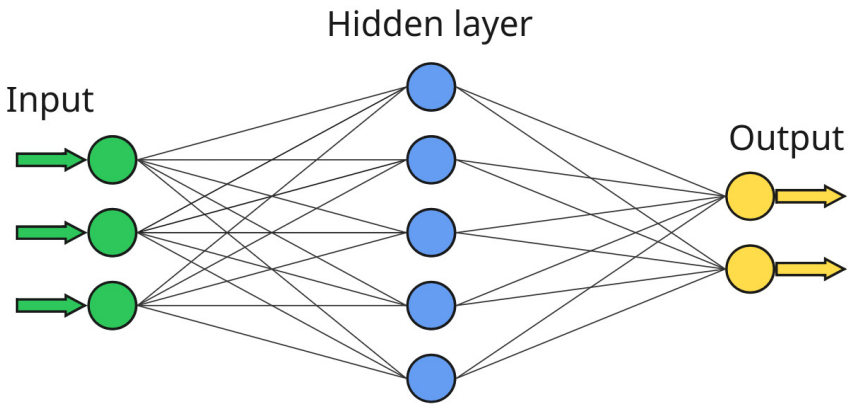

Figure 1 – Example of a feed-forward neural network structure

makes neural networks very useful when a mathematical model for an unknown mapping is needed, and we have only the input data and the expected values for the outputs.

Convolutional Neural Networks (CNNs) represent a specialized class of deep neural networks that have become the standard for tasks involving grid-like data, most notably image analysis and computer vision (LECUN *et al.*, 1998). CNNs employ a unique architecture designed to automatically and adaptively learn spatial hierarchies of features from input data. The core building blocks of a CNN are its convolutional layers, which apply a set of learnable filters (or kernels) to the input. Each filter slides, or convolves, across the input's width and height, computing the dot product between the filter's entries and the input at any position to produce a feature map. This operation allows the network to detect specific features such as edges, corners, and textures. Following the convolutional layers, pooling (or subsampling) layers are typically used to progressively reduce the spatial dimensionality of the representation, which helps to decrease the number of parameters and computational complexity in the network, while also providing a degree of translational invariance. Finally, after several convolutional and pooling layers, the high-level features are fed into one or more fully-connected layers to perform classification or regression, mapping the learned features to the final output.

The first point to be approached during the conception of a neural network is to decide its structure. We need to define the number of inner layers, the number of neurons, and the type of activation function for each layer. Those decisions are often non-trivial and require previous knowledge about the problem to build an efficient neural network. We can find in the literature proposals of techniques to find and optimize models for various applications (CAI *et al.*, 2019).

2.2 Training neural networks

Model training is the process of finding the best model configuration for a desired task given a dataset. In supervised learning, the expected output values (also called labels, when they are discrete values) for the data samples are available and used to train the model. On the other hand, unsupervised learning is a type of algorithm that learns patterns from unlabeled data.

Training a neural network consists of determining the values of the weights for each neuron. Ideally, we aim to find the values that maximize the model's performance. This corresponds to the configuration minimizing the model's approximation error for supervised networks. However, in most cases, this problem does not have a solution in a closed form, given the high complexity and non-convexity of the neural network structure. Thus, we resort to iterative optimization algorithms, which update the values at each step, aiming to minimize a loss function $\mathcal{L}(x)$. The loss function $\mathcal{L}$ is chosen such that its minimization reduces the neural network approximation error for the desired task.

Traditionally, neural networks are trained using gradient descent optimization algorithms. At each iteration, the algorithm evaluates the gradient of the loss function $\mathcal{L}$ (also known as the prediction error gradient) with respect to each weight for a batch of training data. Then, the values are updated by adding a value proportional to the gradient's opposite direction. This way, the model takes a step in the direction that currently decreases the error the most. Since it is a greedy approach, it does not guarantee finding the optimal solution, i.e., the neural network configuration that minimizes the approximation error. Furthermore, depending on the initial weight values and the training batch size, the algorithm may get stuck in local minima. However, once the algorithm's convergence is assured, it is possible to train the model, or even retrain from a distinct initial condition for the weights, until it reaches a sufficiently small error for the desired application.

Feed-forward networks are trained with gradient descent optimization, using the backpropagation algorithm to obtain the gradient vector. The procedure begins by updating the weights of the output layer neurons, as it is straightforward to calculate the gradient of the loss function $\mathcal{L}$ with respect to these quantities. Then, the gradient is propagated to the previous layer and used to calculate the gradient of $\mathcal{L}$ with respect to the weights of the previous layer's neurons, which are then updated. The process repeats for the whole network, from the output layer to the first inner layer.

The bias-variance dilemma is a key concept in model training that describes the relationship between the accuracy of predictions and the generalization capability of the learning model (GEMAN *et al.*, 1992). As model complexity increases, it becomes more flexible and can better fit the training data, thereby reducing bias. At the same time,

the model's variance tends to increase, meaning that small fluctuations in the training set result in high variability in the model's predictions. This is called overfitting, and the consequence is that the model generalizes poorly to data that were not used in the training. The bias-variance trade-off is observed when we seek to improve the model's generalization. As we reduce the variance, the bias tends to increase, which may lead to the opposite situation: underfitting. To reduce the generalization error, we look for a balance between bias and variance. Standard practices include splitting the dataset into training and validation sets and interrupting the training when the prediction error for the validation set is achieved.

2.2.1 Backpropagation algorithm

Let $w^l \in \mathbb{R}^{n^l \times n^{l-1}}$ be the weight matrix for the neurons in the l^{th} layer, where n^l is the layer's dimension, and w^l_{jk} is the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. We define $b^l \in \mathbb{R}^{n^l}$ as the array of bias, and $a^l \in \mathbb{R}^{n^l}$ as the array of neuron activations, given by:

$$a^l = f(z^l) = f\left(w^l a^{l-1} + b^l\right), \tag{2.2}$$

where z^l is the *weighted input* to the neurons in layer l (NIELSEN, 2015).

We want to compute the partial derivatives of the loss function $\frac{\delta \mathcal{L}}{\delta w^l}$ and $\frac{\delta \mathcal{L}}{\delta b^l}$. We define the error δ^l_j of neuron j in layer l by:

$$\delta^l_j = \frac{\delta \mathcal{L}}{\delta z^l_j}, \tag{2.3}$$

and we denote δ^l the array of errors associated with layer l.

First, we compute the error δ^L in the output layer L, given by:

$$\delta^L = \nabla_a \mathcal{L} \odot f'(z^L), \tag{2.4}$$

where $\nabla_a \mathcal{L}$ is the array whose components are the partial derivatives $\frac{\delta \mathcal{L}}{\delta a^L_j}$, and $\odot$ is the element-wise multiplication operator. Then, we propagate the error backwards computing δ^l in terms of the errors in the next layer δ^{l+1}:

$$\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot f'(z^l), \tag{2.5}$$

Now, we can finally calculate the partial derivatives with respect to the weights and biases for all layers:

$$\frac{\delta \mathcal{L}}{\delta b^l} = \delta^l, \tag{2.6}$$

$$\frac{\delta \mathcal{L}}{\delta w^l} = \delta^l (a^{l-1})^T. \tag{2.7}$$

Gradient descent optimization is performed in batches of n training samples. We calculate the gradient for each training sample x_i in a batch, and then we update the weights for each layer by moving them in the opposite direction of the gradient's average:

$$b^l \leftarrow b^l - \frac{\gamma}{n} \sum_{i=1}^{n} \frac{\delta \mathcal{L}(x_i)}{\delta b^l},$$
$$w^l \leftarrow w^l - \frac{\gamma}{n} \sum_{i=1}^{n} \frac{\delta \mathcal{L}(x_i)}{\delta w^l},$$

where γ is the learning rate. In stochastic gradient descent, the batches are formed by a randomly selected subset of the data. A complete pass of all samples in the training data defines an *epoch*. The process repeats for a given number of epochs n_{ep}. Algorithm 1 summarizes the main steps of the backpropagation algorithm.

Algorithm 1: Backpropagation

Input: training batch $\mathcal{B}$, weight w, bias b, learning rate γ

$g_w, g_b \leftarrow (0, 0)$; `// Initialize gradient`

for x **in** $\mathcal{B}$ **do** `// For each sample`

  $a^0 \leftarrow x$;

  for $l \leftarrow 1$ **to** L **do** `// Forward pass`

    $z^l \leftarrow w^l a^{l-1} + b^l$; `// Compute weighted input`

    $a^l \leftarrow f(z^l)$; `// Compute neuron activation`

  end

  $\mathcal{L} \leftarrow \mathcal{L}(x, w, b)$; `// Compute loss`

  $\delta^L = \nabla_a \mathcal{L} \odot f'(z^L)$; `// Compute output error`

  $g_w^L \leftarrow g_w^L + \delta^L (a^{L-1})^T$; `// Output gradient`

  $g_b^L \leftarrow g_b^L + \delta^L$;

  for $l \leftarrow L - 1$ **to** 1 **do** `// Backward pass`

    $\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot f'(z^l)$; `// Compute error`

    $g_w^l \leftarrow g_w^l + \delta^l (a^{l-1})^T$; `// Weight gradient`

    $g_b^l \leftarrow g_b^l + \delta^l$; `// Bias gradient`

  end

end

for $l \leftarrow 1$ **to** L **do** `// Gradient descent`

  $w^l \leftarrow w^l - \frac{\gamma}{|\mathcal{B}|} g_w^l$; `// Update weight`

  $b^l \leftarrow b^l - \frac{\gamma}{|\mathcal{B}|} g_b^l$; `// Update bias`

end

2.3 Autoencoders: definition and applications

Autoencoders (AE) are feed-forward neural networks made popular by Kramer, whose structure has two parts: an encoder and a decoder (KRAMER, 1991). The first inner layers make the encoder, which is responsible for producing an encoding or representation of the input data. The final layers belong to the decoder, which aims to retrieve the original data from the encoding. The general purpose of an autoencoder is then to reconstruct the input information at the output. A proper reconstruction requires a highly informative code generation at the output of the encoder.

Typically, the encoding produced by autoencoders has the property of dimensionality reduction, i.e., the representation has a dimension smaller than the dimension of the input data. To achieve this behavior, the encoder layers are built with a decreasing number of units, while the opposite holds for the decoder layers. This creates a bottleneck in the neural network situated at the connection between the encoder and the decoder.

Given that the encoder and decoder are trained simultaneously, a high-quality reconstruction at the decoder should be associated with a high-quality coding step at the output of the encoder. That is why the autoencoder structure requires a coding step followed by a reconstruction step, which maps the code at the bottleneck to a version as close as possible to the input content. The model implicitly learns a compact representation, carrying enough information to retrieve the original data. Fig. 2 shows an example of an autoencoder structure that may include convolutional and fully connected layers to obtain low-dimensional matrix representations.

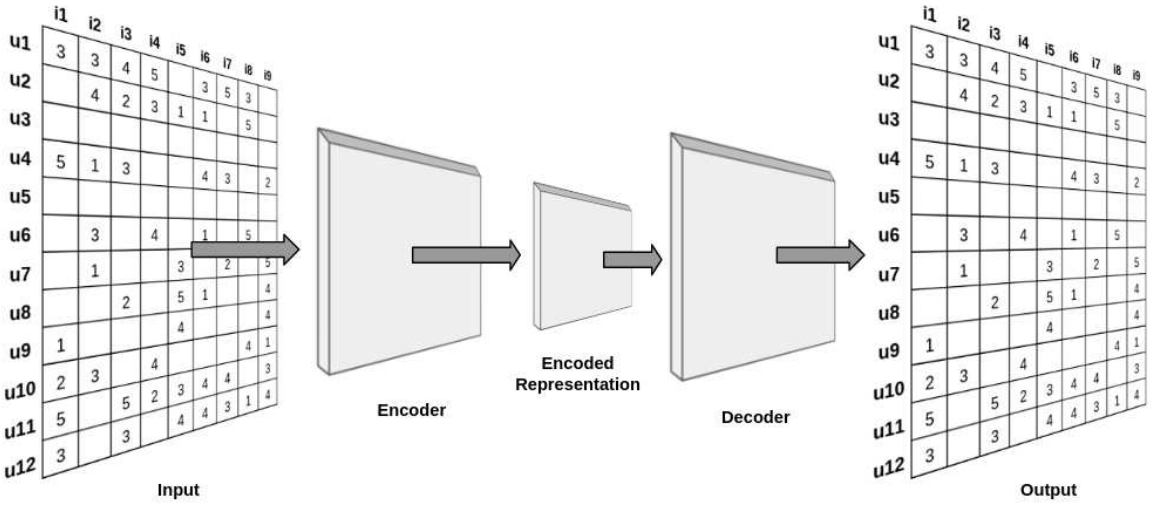

Figure 2 – Illustrative example of an autoencoder structure for two-dimensional data.

The dimensionality reduction introduced by autoencoders makes them good candidates to implement a cascade of knowledge representation filters in deep neural

networks (GLOROT; BENGIO, 2010; TONG *et al.*, 2021).

Another interesting application for autoencoders can be found in recommender systems (ZHANG *et al.*, 2020), a class of methods that will be introduced in the next chapter. The highly informative and compact representations obtained at the bottleneck of autoencoders often lead to competitive recommender systems, both in terms of performance and scalability, especially in large datasets, where traditional methods scale poorly.

2.4 BinaPs: Differentiable Pattern Set Mining

The pattern set mining techniques succeed in finding small sets of valuable and informative data patterns. However, this task is costly as the pattern set search space grows with double exponential complexity. Hence, the most employed techniques are heuristics, which have limitations on the number of features and no guarantee of finding optimal solutions. In (FISCHER; VREEKEN, 2021), the authors proposed the Binary Pattern Networks (BinaPs) method, an interpretable neural autoencoder for binary data. By resorting to a gradient-based approach instead of a combinatorial one, this technique stands out for its scalability and ability to handle noise and deal with sparsity.

BinaPs' network structure consists of two linear layers, one for the encoder and another for the decoder, sharing the same continuous weights and a novel binary activation function. In each forward pass, weights are binarized, and the neurons in the hidden layer are interpreted as conjunctive patterns. The training adopts a reconstruction loss that takes into account both dense and sparse data, which allows the network to be applied to many types of datasets.

Let $R \in \{0,1\}^{n \times m}$ be a binary matrix of n samples and m features, $R[i]$ the i-th row in the data matrix, $R[i,j]$ the value of feature $j \in [1,m]$ for the i-th sample, k the hidden layer's size, $W \in \mathbb{R}^{k \times m}$ the weight matrix, $W_b \in \{0,1\}^{k \times m}$ the corresponding binarized weights, $b \in \mathbb{R}^k$ the bias and $b_d \in \mathbb{Z}^k$ its discretization. BinaPs aims to find the pattern set P, where each pattern $p \in P, p \subset \{1,2,...,m\}$ is a set of correlated features. Each neuron in the hidden layer represents an encoded pattern interpreted from W_b, with $W_b[i,j]$ indicating whether feature j is part of pattern i. The pattern set P corresponds to the aggregated patterns from all neurons.

The encoding and decoding layers activation functions $f_E, f_D : \mathbb{R} \to \{0,1\}$, respectively, are given by:

$$f_E(x) = \text{round}(\text{clamp}(x + b_d, 0, 1)), \tag{2.8}$$

$$f_D(x) = \text{round}(\text{clamp}(x, 0, 1)), \tag{2.9}$$

where *clamp* is the clamping function defined by:

$$\text{clamp}(x, l_b, u_b) = \begin{cases} l_b & \text{if } x < l_b; \\ x & \text{if } l_b \leq x \leq u_b; \\ u_b & \text{if } x > u_b. \end{cases} \tag{2.10}$$

For sample $r \in R$ in the forward pass, $y = f_E(rW_b^T)$ is the result of the encoding layer, and the network output is computed by $z_r = f_D(yW_b)$. The autoencoder is trained using the traditional gradient descent approach with the continuous parameters W and b and minimizing the loss function $\mathcal{L}$ given by:

$$\mathcal{L}(R, W, b) = \sum_{r \in R} \|z_r - r\|. \tag{2.11}$$

For sparse data, a sparsity-dependent reconstruction loss $\mathcal{L}_\beta$ is adopted:

$$\mathcal{L}_\beta(r, W, b) = \sum_{j=1}^{m} [(1 - r_j)\beta + r_j(1 - \beta)] \cdot |z_{r,j} - r_j|, \tag{2.12}$$

where r_j is the j-th feature of sample r, $z_{r,j}$ is its corresponding network output, and β is the data sparsity given by:

$$\beta = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} R[i, j]}{n \cdot m}. \tag{2.13}$$

An approximation is used for the derivative of the activation functions since there is no analytical solution. For the decoder activation $f_D(x)$ with incoming gradient flow g_0, the derivative is given by the straight-through-estimator:

$$\frac{df_D}{dx} = \mathbb{1} g_0. \tag{2.14}$$

For the encoder, a gated straight-through estimator is adopted:

$$\frac{df_E}{db} = \begin{cases} g_0 & \text{if } f_E(x) = 1; \\ 0 & \text{if } f_E(x) = 0; \end{cases} \tag{2.15}$$

$$\frac{df_E}{dx} = \begin{cases} g_0 & \text{if } f_E(x) = 1; \\ \max(0, g_0) & \text{if } f_E(x) = 0. \end{cases} \tag{2.16}$$

After updating the weights, the bias is discretized according to Eq. (2.17):

$$b_d[i] = \min(\lceil b[i] \rceil, -1). \tag{2.17}$$

The weights are clamped to be in the range $[\frac{1}{m}, 1]$, and the binarized weights are drawn from a Bernoulli distribution as in Eq. (2.18). Hence, for each neuron, we expect at least 1 of the m items to be assigned to its pattern, no matter the data dimension:

$$W_b[i, j] = \mathcal{B}(W[i, j]). \tag{2.18}$$

To train BinaPs on a dataset R, we need to define the maximum number of epochs n_{ep}. We choose the number of neurons in the hidden layer k, equal to the number of features m by default. The optimizer also requires defining the following parameters: batch size l_b, initial learning rate γ, and learning rate step η. The learning rate decays by η once the number of epochs reaches one of the milestones, which are set by default to 50% and 70% of n_{ep}. For more details on the algorithm, see (FISCHER; VREEKEN, 2021).

2.5 Summary

We start this chapter with a brief introduction to neural networks, where we talk about model formulation, structure, and training. Then, we discuss Autoencoders, a particular kind of neural network model formed by the concatenation of an encoder and a decoder. Autoencoders are trained to generate an effective encoding of the input data while reconstructing its content at the output. The dimensionality reduction allows these models to be applied in many domains, such as deep learning and recommender systems. Next, we present the BinaPs algorithm, a robust and efficient autoencoder for pattern set mining on binary data, which will be a key element of the method we will propose later. BinaPs adopts a binarized version of the model weights in its forward pass and adopts a loss function suited for sparse datasets. The binary weights can be interpreted as conjunctive patterns, indicating sets of correlated features. We finish this chapter by detailing the Binaps training procedure, a gradient descent-based approach, which makes the technique stand out compared to traditional combinatorial approaches for pattern set mining.

3 Recommender systems

This chapter discusses Recommender Systems, a class of algorithms devoted to personalized recommendations. We present the main concepts, some variations, and the main challenges encountered in their development.

3.1 Definition and applications

Web development and technology evolution have increased the amount of available information, leading to a wide range of products and services. Recommender Systems (RS) are tools devised to produce personalized recommendations to users, aiming to draw their attention to items of interest, e.g., music tracks, movies, books, etc. Many companies from different business sectors are adopting recommender systems because good recommendations help improve user satisfaction, retain customers, and increase sales. Examples of big companies making extensive use of recommender systems are Netflix (BENNETT; LANNING, 2007), Facebook (ILIC; KABILJO, 2015), and Amazon (HARDESTY, 2019).

Algorithms for recommender systems usually predict how a user would rate an unseen item based on the ratings he and other users provided, and on user/item information, such as user demographics, item characteristics, etc. (LAMPROPOULOS; TSIHRINTZIS, 2013) To formulate the recommendation problem, we must define $U = \{u_1, u_2, ..., u_n\}$ as the set of n users and $I = \{i_1, i_2, ..., i_m\}$ as the set of m items that can be recommended. Let f be a utility function establishing how useful item i is to user u:

$$f : U \times I \to V, \tag{3.1}$$

where V is a totally ordered set, e.g., the real numbers in the interval $[0, 5]$, representing the ratings, usually stored in a matrix R, called the *ratings matrix*. Each row of R contains the ratings a user gives for each item, and each column represents the rating users give for a specific item. The matrix R is usually sparse, as users only provide ratings for a small portion of the available items.

In a recommendation problem, our goal is to find the item $j_u \in I$ that maximizes the utility function for each user $u \in U$:

$$\forall u \in U, j_u = \arg\max_{i \in I} f(u, i), \tag{3.2}$$

Two approaches are usually adopted to generate predictions: model-based and memory-based (BEHERA; NAIN, 2022). The former employs a model learned from the

dataset using tools from different domains, such as graph theory, statistics, matrix factorization, and machine learning. In contrast, the latter uses the entire user's rating history to find similarities between users or items and recommend unseen items. Hybrid methods combine memory-based and model-based approaches, allowing us to leverage both historical user rating data and user/item attributes. The enriched available information may improve performance but restrict applicability solely to more informative datasets.

In general, there are two main categories of recommender systems: *Content-Based Filtering* (CBF) and *Collaborative Filtering* (CF). Content-based filtering methods generate recommendations based on information extracted from item features, the user's historical records, or explicit feedback (BURKE, 2002). The model should infer user preferences without relying on information about other users. On the other hand, Collaborative Filtering approaches use available ratings for all users and items to predict ratings based on user and/or item similarities. These systems can recommend an item to user A based on two aspects: the similarity of interests when comparing the behavior of users A and B, and a high rate assigned to that specific item by user B. CF methods are often preferred over CBF methods because the former only relies on the users' ratings, while the latter requires advanced engineering on items to perform well (STRUB *et al.*, 2016).

3.2 Collaborative Filtering

CF methods assume that similar users share preferences for similar items and rely solely on the ratings provided by the users without performing content analysis. Model-based CF approaches build a probabilistic model using the underlying data, applying techniques from statistics and machine learning, for instance. The model is then used to predict ratings for unseen items and needs to be updated regularly. These techniques often focus on retrieving potential latent factors from the rating matrix (ALSHBANAT *et al.*, 2025). A well-established example is the *Singular Value Decomposition* (SVD) method, which uses matrix factorization to reduce the dimensionality of the ratings matrix. It became popular when Simon Funk introduced it during the Netflix Prize contest (FUNK, 2006). Another notable representative is the SVD++ method, which extends the classic SVD latent-factor model by incorporating implicit feedback. The SVD++ is considered a reference method in CF (KOREN, 2008).

Memory-based methods use the entire ratings matrix to find similarities between users and/or items and calculate predictions. They are usually based on the Nearest Neighbors approach and can be classified as User-based or Item-based methods. In neighborhood-based methods, a proximity metric is used to calculate the similarity between users or items and find the similar neighbors that will be used to calculate the predictions.

3.2.1 Latent Factor Models for Collaborative Filtering

Latent factor models transform both users and items into a common latent factor space of dimensionality f, making them directly comparable. Each user u is associated with a user-factors vector $p_u \in \mathbb{R}^f$, and each item i with an item-factors vector $q_i \in \mathbb{R}^f$. These factors may represent abstract dimensions, such as genre, character depth, or other uninterpretable characteristics that are automatically inferred from the data. The prediction for a rating r_{ui} is then computed as the inner product of these vectors: $\hat{r}_{ui} = p_u^T q_i$.

One of the most popular realizations of latent factor models is based on matrix factorization, famously popularized by techniques used in the Netflix Prize competition (FUNK, 2006). While applying traditional Singular Value Decomposition (SVD) is difficult due to the high number of missing ratings in the user-item matrix, modern approaches overcome this by modeling only the observed ratings and using regularization to prevent overfitting (KOREN, 2008). The predicted rating $P_{u,i}$ is calculated by taking an inner product:

$$P_{u,i} = b_{ui} + p_u^T q_i, \tag{3.3}$$

where b_{ui} is a bias term that accounts for user and item effects, given by:

$$b_{ui} = \mu + b_u + b_i, \tag{3.4}$$

where μ is the overall average rating, and b_u and b_i are the observed deviations from the average of user u and item i, respectively. The corresponding prediction error $e_{u,i}$ is given by:

$$e_{u,i} = R[u,i] - P_{u,i}. \tag{3.5}$$

The model is obtained by finding the parameters p^*, q^* and b^* minimizing the *Root Mean Squared Error* (RMSE) of the prediction given by:

$$RMSE = \sqrt{\frac{\sum\limits_{(u,i)\in\mathcal{K}} e_{u,i}^2}{|\mathcal{K}|}}, \tag{3.6}$$

where $\mathcal{K} = \{(u,i)|\exists R[u,i]\}$ is the training set formed by users and items for which a rating is known. To avoid overfitting, a regularization parameter λ_1 is introduced, guiding to the following regularized least squares problem:

$$\min_{p^*,q^*,b^*} \sum_{(u,i)\in\mathcal{K}} (R[u,i] - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_1(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2). \tag{3.7}$$

This formulation is often solved using techniques such as stochastic gradient descent to estimate the model parameters. We loop over the training set $\mathcal{K}$, and for each sample (u, i), we evaluate the predicted rating $P_{u,i}$, the corresponding prediction error $e_{u,i}$ and update the parameters by moving them in the opposite direction of the gradient with learning rate γ, yielding:

- $b_u \leftarrow b_u + \gamma(e_{u,i} - \lambda_1 b_u)$;
- $b_i \leftarrow b_i + \gamma(e_{u,i} - \lambda_1 b_i)$;
- $q_i \leftarrow q_i + \gamma(e_{u,i} p_u - \lambda_1 q_i)$;
- $p_u \leftarrow p_u + \gamma(e_{u,i} q_i - \lambda_1 p_u)$.

The SVD++ enriches the basic SVD approach by incorporating implicit user feedback into the latent factor model. Natural implicit feedback can be easily obtained by binarizing the ratings matrix, setting 1 to entries with known ratings and 0 otherwise. In other words, the ratings matrix implicitly tells us all the items a user has interacted with, regardless of their ratings. Incorporating this and other implicit data significantly improves the model's prediction accuracy (KOREN, 2008).

The prediction rule for the SVD++, given by Eq. (3.8), is modified to accommodate the implicit data $N[u]$ for user u:

$$P_{u,i} = b_{ui} + q_i^T \left(p_u + |N[u]|^{-\frac{1}{2}} \sum_{j\in N[u]} y_j \right), \tag{3.8}$$

where $y_j \in \mathbb{R}^f$ is a factor-array associated with item j. The user factor p_u from the basic model given by Eq. (3.3) is now added to the term $|N[u]|^{-\frac{1}{2}} \sum_{j\in N[u]} y_j$ that accounts for implicit feedback. The model is obtained by solving the following least squares problem:

$$\min_{p^*,q^*,y^*,b^*} \sum_{(u,i)\in\mathcal{K}} \left(R[u,i] - \mu - b_u - b_i - q_i^T \left(p_u + |N[u]|^{-\frac{1}{2}} \sum_{j\in N[u]} y_j \right) \right)^2 + \lambda_2(b_u^2 + b_i^2) + \lambda_3 \left(\|p_u\|^2 + \|q_i\|^2 + \sum_{j\in N[u]} \|y_j\|^2 \right). \tag{3.9}$$

We obtain the parameters with gradient-descent optimization by computing:

- $b_u \leftarrow b_u + \gamma(e_{u,i} - \lambda_2 b_u)$;
- $b_i \leftarrow b_i + \gamma(e_{u,i} - \lambda_2 b_i)$;
- $q_i \leftarrow q_i + \gamma(e_{u,i}(p_u + |N[u]|^{-\frac{1}{2}} \sum_{j \in N[u]} y_j) - \lambda_3 q_i)$;
- $p_u \leftarrow p_u + \gamma(e_{u,i} q_i - \lambda_3 p_u)$;
- $\forall j \in N[u] : y_j \leftarrow y_j + \gamma(e_{u,i}|N[u]|^{-\frac{1}{2}} q_i - \lambda_3 y_j)$.

3.2.2 User-based Collaborative Filtering

User-based Collaborative Filtering systems are memory-based methods that make predictions based on user similarities, i.e., they deal with the rows of the rating matrix R. In this approach, to generate recommendations for a target user, we build a user neighborhood formed by customers who have shown similar preferences to the target user. Then, the items the neighbors like and that are unknown to this user are recommended.

In order to build the neighborhood, we calculate the similarity between each pair of users in the ratings matrix using a proximity metric. The *Pearson correlation coefficient* (RESNICK *et al.*, 1994) evaluates the similarity between users u_1 and u_2 according to the following equation:

$$\text{sim}(u_1, u_2) = \frac{\sum\limits_{i \in T}(R[u_1, i] - \overline{R}[u_1])(R[u_2, i] - \overline{R}[u_2])}{\sqrt{\sum\limits_{i \in T}(R[u_1, i] - \overline{R}[u_1])^2 \sum\limits_{i \in T}(R[u_2, i] - \overline{R}[u_2])^2}}, T = I_{u_1} \cap I_{u_2}, \tag{3.10}$$

where $R[u, i]$ is the rating given by user u to item i, I_u is the set of items rated by user u, and $\overline{R}[u]$ is the user's average rating for a collection of items T, calculated as:

$$\overline{R}[u] = \frac{\sum\limits_{i \in T} R[u, i]}{|T|}. \tag{3.11}$$

The *cosine-based similarity* (BREESE *et al.*, 1998) is another similarity metric that computes the cosine angle between two vectors:

$$\text{sim}(u_1, u_2) = \frac{\sum\limits_{i \in T} R[u_1, i] R[u_2, i]}{\sqrt{\sum\limits_{i \in T} R[u_1, i]^2 \sum\limits_{i \in T} R[u_2, i]^2}}, T = I_{u_1} \cap I_{u_2}. \tag{3.12}$$

The Pearson correlation coefficient tends to perform better than the cosine similarity for User-based CF methods (BREESE *et al.*, 1998).

As we evaluate the similarity between each pair of users, we build a *user similarity matrix* S_u, where each matrix's entry $S_u[i, j]$ stores the similarity between users u_i and u_j, i.e., $S_u[i, j] = \text{sim}(u_i, u_j)$. Then, we can build the neighborhood for a target user u_t adopting a *center-based scheme*, in which the neighbors are the most similar users to u_t. An alternative is to use an *aggregate formation scheme*, in which the neighborhoods are defined by *centroids*, and are gradually built by adding each user to the neighborhood with the closest centroid (LAMPROPOULOS; TSIHRINTZIS, 2013).

In order to generate recommendations for the target user u_t, we need to predict the ratings for items not rated by u_t. The predicted values must be in the same range as those observed in the ratings matrix R. Let i_t be an item for which we want to predict the unknown rating P_{u_t,i_t}. The prediction only considers the subset of users M, selected from those who have rated the item i_t and are in the target user's neighborhood u_t. The predicted value is calculated as a weighted sum of the ratings of i_t given by the selected users, as shown in Eq. (3.13):

$$P_{u_t,i_t} = \overline{R}[u_t] + \frac{\sum\limits_{u \in M} (R[u, i_t] - \overline{R}[u]) \cdot \text{sim}(u_t, u)}{\sum\limits_{u \in M} |\text{sim}(u_t, u)|}. \tag{3.13}$$

In *User-based K-Nearest Neighbors* (UBKNN) approaches (HERLOCKER *et al.*, 2004), the subset M contains the k most similar users to the target user who have rated the item i_t, where k is a parameter to be defined.

The RS will recommend the items with the best-predicted ratings. For instance, a value t_r may be defined as a *relevance threshold*, and the output will include all items whose predicted ratings are above t_r. Alternatively, the output can be a *top-N recommendation* list, obtained by sorting the predicted ratings in descending order and selecting the first N items.

3.2.3 Item-based Collaborative Filtering

Item-based CF approaches work with the relationship between the ratings matrix's columns instead of rows, i.e., items instead of users. User relations are much more subject to changes, as users are constantly interacting with new items. This increases the computational cost of user-based CF approaches, as keeping the user similarity matrix updated is computationally expensive (LAMPROPOULOS; TSIHRINTZIS, 2013).

The item-based approach is similar to the user-based approach, except that we use the similarities between items rather than users. To evaluate the similarity between a pair of items i_1 and i_2, we can use the Pearson correlation coefficient, given by Eq. (3.14), and the cosine similarity, given by Eq. (3.15).

$$\text{sim}(i_1, i_2) = \frac{\sum\limits_{u \in V} (R[u, i_1] - \overline{R}[i_1])(R[u, i_2] - \overline{R}[i_2])}{\sqrt{\sum\limits_{u \in V} (R[u, i_1] - \overline{R}[i_1])^2 \sum\limits_{u \in V} (R[u, i_2] - \overline{R}[i_2])^2}}, V = U_{i_1} \cap U_{i_2}; \tag{3.14}$$

$$\text{sim}(i_1, i_2) = \frac{\sum\limits_{u \in V} R[u, i_1] R[u, i_2]}{\sqrt{\sum\limits_{u \in V} R[u, i_1]^2 \sum\limits_{u \in V} R[u, i_2]^2}}, V = U_{i_1} \cap U_{i_2}; \tag{3.15}$$

where U_i is the set of users who rated item i, and $\overline{R}[i]$ is the item's average rating for a collection of users V, calculated as:

$$\overline{R}[i] = \frac{\sum\limits_{u \in V} R[u, i]}{|V|}. \tag{3.16}$$

These metrics do not consider the variation in rating scales among different users. The *Adjusted Cosine Similarity* (SARWAR *et al.*, 2001), given by Eq. (3.17), addresses this issue by subtracting the user's average from each co-rated pair of ratings:

$$\text{sim}(i_1, i_2) = \frac{\sum\limits_{u \in V} (R[u, i_1] - \overline{R}[u])(R[u, i_2] - \overline{R}[u])}{\sqrt{\sum\limits_{u \in V} (R[u, i_1] - \overline{R}[u])^2 \sum\limits_{u \in V} (R[u, i_2] - \overline{R}[u])^2}}, V = U_{i_1} \cap U_{i_2}, \tag{3.17}$$

where $\overline{R}[u]$ is the average rating of all the items rated by user u, calculated using $T = \{i : \exists R[u, i]\}$ in Eq. (3.11).

Once the *item similarity matrix* S_i is obtained, for which $S_i[j, k] = sim(i_j, i_k)$, we find the item neighborhoods using the same schemes adopted in user-based approaches. To predict the rating P_{u_t, i_t} for the target user u_t on item i_t, we consider a subset of items L, selected from the items rated by the target user u_t in the neighborhood of the item i_t. The predicted value is calculated as a weighted sum of the ratings of the items in L given by the target user, as shown in Eq. (3.18):

$$P_{u_t, i_t} = \frac{\sum\limits_{j \in L} R[u_t, j] \cdot \text{sim}(i_t, j)}{\sum\limits_{j \in L} |\text{sim}(i_t, j)|}. \tag{3.18}$$

In *Item-based K-Nearest Neighbors* (IBKNN) approaches (SARWAR *et al.*, 2001), the subset L contains the k most similar items to the item i_t rated by the target user u_t. Eq. (3.19) shows an alternative version for calculating the predictions, using the centered ratings:

$$P_{u_t,i_t} = \overline{R}[i_t] + \frac{\sum\limits_{j \in L} (R[u_t, j] - \overline{R}[j]) \cdot \text{sim}(i_t, j)}{\sum\limits_{j \in L} |\text{sim}(i_t, j)|}. \tag{3.19}$$

3.2.4 User-based VS Item-based Recommendation

Five criteria must be considered when choosing between a user-based or an item-based collaborative filtering approach. First, the recommendation *Accuracy* depends on the ratio between the number of users and items in the dataset. Usually, we prefer a small number of high-confidence neighbors over many neighbors with non-reliable similarity values. Therefore, if the number of users is much greater than the number of items, an item-based approach is preferable, while a user-based approach is more adequate for systems with fewer users than items (DESROSIERS; KARYPIS, 2011).

Second, we consider the recommender system's *Efficiency*, which also depends on the previously mentioned user/item ratio. Similarly, item-based approaches outperform user-based approaches when the number of users is significantly higher than the number of items, which is typically the case, as they have smaller space and time complexities, making them more scalable.

Another criterion to be considered is the dataset *Stability*. We prefer an item-based approach when the list of available items is less subject to change than the users in the dataset. If the opposite is true, a user-based approach is more suitable.

The recommender system's *Explicability* is also a factor. Item-based approaches are more explicable, as it is easier for the target user to understand the recommendations when they are justified by presenting the similar items used to calculate the prediction. This allows the user to participate in the recommendation process, which is less likely in user-based approaches, as the target user does not know the other users employed in the process.

Finally, we consider the recommendations *Serendipity* or novelty. Item-based approaches usually recommend items similar or related to those the target user appreciates, resulting in safe recommendations. User-based approaches, on the other hand, are more likely to provide serendipitous recommendations, as similar users may have interacted with items from different categories, which allows the target user to discover new preferences.

3.3 Common Issues in Recommender Systems

The *Cold-Start Problem* (SCHEIN *et al.*, 2002) is a common issue in Collaborative Filtering approaches, and occurs when making recommendations for a new user

or a new item. The RS struggles to predict ratings for users with unknown preferences and items that have not been rated by any user. This problem is related to the system's *coverage*, a metric evaluating the domain of items for which the RS can provide a recommendation. Solutions for this problem include using content-based or hybrid approaches to infer item similarities from content information and introducing randomness in the system to avoid recommending items similar to those already rated (LAMPROPOULOS; TSIHRINTZIS, 2013).

In RS, we simultaneously seek *Serendipity* and *Quality* in recommendations. However, there is a trade-off between these goals. The quality of the recommendations is associated with how much the user trusts them (SARWAR *et al.*, 2000), and the RS should avoid recommending non-desirable items when seeking high-quality recommendations. However, the novelty rate is increased by prioritizing fresh items over old-fashioned ones, possibly leading to poor-quality recommendations since users won't be familiar with them.

Usually, RS datasets have a *Sparsity* problem: there is a small portion of rated items per user compared to the total number of available entries. This creates a challenge for the prediction task, degrading the method's accuracy. In user-based approaches, the similarity is calculated only between users who have rated items in common, which reduces the number of neighbors in sparse datasets, compromising the coverage. Among the alternatives proposed to address this issue are content-based, item-based approaches, and hybrid approaches (BURKE, 2002).

Scalability is also a common problem in recommender systems. As the size of the database increases, the method requires more computational resources (KUMAR; SHARMA, 2013). In neighborhood-based methods, computing similarities between every pair of users/items becomes increasingly time-consuming. The scalability problem can be addressed using data mining techniques to partition the entire dataset into subsets of individuals who share similar interests. For instance, the market segmentation strategy (KOTLER; ARMSTRONG, 2010) creates small groups of users sharing similar preferences for specific products, thus allowing the generation of group-specific recommendations (PARK; NAM, 2019). Since the dimensions of the rating matrix are usually very high, it makes sense to resort to the similarity between subsets of users based on their preferences for a subset of items. The similarity of users and items is evaluated within partitions instead of the entire ratings matrix, thereby reducing the computational burden.

3.4 Summary

In this chapter, we introduced Recommender Systems, a class of methods developed to generate personalized recommendations for users. They are divided into two main categories: Content-Based Filtering (CBF) and Collaborative Filtering (CF). The former

generates recommendations based on users' historical records or explicit feedback. On the other hand, the latter explores all the available ratings for all users to make predictions based on user/item similarities. We discussed some of the most important approaches for CF, starting with the Singular Value Decomposition (SVD) method and its extension, the SVD++, two model-based techniques that use matrix factorization and gradient descent optimization to generate predictions. Then, we discussed neighborhood-based approaches, which focus on identifying similar users (in User-based CF) or similar items (in Item-based CF) to build user/item neighborhoods. We presented the most common metrics used to evaluate the similarity between users and items, as well as the prediction formulation in each case. We follow this with a discussion of points to consider when choosing between a User-based approach and an Item-based approach, such as the recommender system's accuracy and efficiency, which depend on the dataset's user-to-item ratio. We conclude the chapter by pointing out some of the common issues encountered during the development of recommender systems, including sparsity and scalability.

4 Autoencoder-based Recommender Systems

In this chapter, we present a literature review on autoencoders and alternative subspace approaches applied in recommender systems. Subsequently, we introduce our proposed recommender system, properly highlighting the main advantages over the existing approaches. It consists of a Collaborative Filtering method using the IBKNN approach, which utilizes data representations obtained with the help of an autoencoder, specifically the BinaPs algorithm, detailed in Section 2.4. In this case, we find a representation of the ratings matrix given by a pattern set. Our intention is to take advantage of BinaPs' high scalability and robustness to sparse data. We will refer to the proposed method as APCF (*Autoencoder Pattern-based Collaborative Filtering*).

4.1 Autoencoders in Recommender Systems

The adoption of deep learning techniques, most notably autoencoders, has emerged as an alternative to address the limitations observed in the development of recommender systems. Their architecture's capability to learn complex, non-linear relationships within data makes it particularly well-suited for addressing the challenges inherent to RS, such as dealing with sparse and large-sized user-item interaction matrices and capturing nuanced user preferences (BANK *et al.*, 2021). By learning a compressed representation of user-item interactions, autoencoders can effectively reduce the dimensionality of the data, mitigating the sparsity issue and improving the efficiency of recommendation algorithms (HUANG *et al.*, 2024). Furthermore, the non-linear nature of autoencoders allows them to capture complex relationships between users and items that linear methods may fail to identify.

Various autoencoder architectures have been explored for recommender systems, including *Denoising Autoencoders* (DAE) (VINCENT *et al.*, 2008) and *Variational Autoencoders* (VAE) (KINGMA; WELLING, 2019). DAE models are trained to reconstruct the input from a corrupted version, being employed to enhance the robustness of the learned representations and improve generalization performance. *Stacked Denoising Autoencoders* (SDAE) stack various DAEs to achieve higher-level representations. In contrast, *Marginalized Denoising Autoencoders* (MDAE) alleviate their high computational cost by marginalizing stochastic feature corruption (ZHANG *et al.*, 2020). VAE models extend the basic autoencoder architecture by introducing probabilistic elements, making them particularly useful in recommender systems, as these models are robust to data

sparsity and compatible with other deep learning-based models when dealing with multi-modal data. In addition, the deep generative structure of VAEs helps to perform Bayesian inference in an efficient manner (LIANG *et al.*, 2024).

Within recommendation systems, autoencoders are typically employed in one of two main paradigms. The first approach primarily utilizes the autoencoder for feature extraction, where the compressed representation from the encoder's bottleneck layer serves as a set of rich, nonlinear latent factors for users and items. These factors can then be used in subsequent prediction models, analogous to how traditional matrix factorization models operate, such as *Singular Value Decomposition* (SVD) (KOREN, 2008). The second, more direct approach, uses the reconstruction capability of the autoencoder for prediction. In this framework, an incomplete rating vector for a user or item is fed into the network, and the reconstructed, dense output vector is used to predict the missing ratings directly. Seminal models such as AutoRec and its extensions (HUANG *et al.*, 2024) are prime examples of this reconstruction-based methodology.

4.1.1 Autoencoders in latent factor-based models

We present in this section some methods exploring autoencoders for feature extraction in RS. Latent factor models, in particular, have shown great promise in the development of recommender systems. They function by transforming high-dimensional and sparse user-item interaction data into a dense, lower-dimensional latent space where users and items can be directly compared. Early successes in this area were driven by Matrix Factorization (MF) techniques, such as SVD, *Non-negative Matrix Factorization* (NMF) (LUO *et al.*, 2014), and *Probabilistic Matrix Factorization* (PMF) (SALAKHUTDINOV; MNIH, 2007). However, the inherent linearity of these models can limit their ability to capture the complex, non-linear relationships present in user preference data. Autoencoders have been integrated into collaborative filtering frameworks to learn latent representations of users and items from the observed interaction data. Regularization techniques play a crucial role in preventing overfitting and ensuring that the learned compressed representation is meaningful (BANK *et al.*, 2021).

Deep Collaborative Filtering (DCF) (LI *et al.*, 2015) is a method that integrates PMF and a *Marginalized Stacked Denoising Autoencoder* (MSDAE) , using both the rating matrix and side information. User and item latent features are learned through PMF, and MSDAE is used to extract contextual features, which are then connected with the latent features in the model. *Recommendation with Social Relationships via Deep Learning* (RSRDL) (RAFAILIDIS; CRESTANI, 2017) uses a similar approach, combining MF and MSDAE, but using implicit feedback for MF and social relationships data for MSDAE. It employs a joint objective function that enforces the latent user-item features to be as close as possible to the representations of social relationships obtained by the autoencoder.

Collaborative variational autoencoder (CVAE) (LI; SHE, 2017) is a Bayesian generative model designed to integrate item content information (e.g., text from reviews, movie plots, or other multimedia features) with rating data. The CVAE architecture works in two main stages. First, it uses a VAE to learn a deep, latent probabilistic representation of the items' content data in an unsupervised fashion. In the second stage, these learned content representations are integrated with user rating information within a unified collaborative filtering framework. The model learns implicit relationships between users and items by considering both the features derived from the content and the explicit signals from the ratings.

Another powerful approach to enhancing latent factor models involves leveraging external, structured knowledge. The *Collaborative Knowledge Base Embedding* (CKE) (ZHANG *et al.*, 2016) exemplifies this by fusing collaborative filtering with information from a Knowledge Base (KB). KBs contain a wealth of factual information about items, such as a movie's genre, director, and actors, structured as a graph of entities and relations. CKE integrates AE with other embedding methods to obtain semantic representations from structural, textual, and visual information in the KB. By jointly optimizing the objectives of both CF and KB embedding, the model learns item latent factors that are shaped not only by user preferences but also enriched with structured, real-world attributes, thereby improving the quality of RSs.

The AutoSVD++ model (ZHANG *et al.*, 2017) enhances the robust SVD++ framework by replacing its traditional item correlation component with a more sophisticated model learned by a *Contractive Autoencoder* (CAE). A contractive autoencoder is a specialized variant that is explicitly regularized to be robust to small perturbations in its input data, forcing it to learn more significant and stable features. The SVD++ model effectively captures user-specific biases and implicit feedback, but its modeling of item-item relationships is relatively simple. AutoSVD++ addresses this by training a CAE on the item-rating vectors from the user-item matrix. These learned item representations are then integrated into the SVD++ prediction formula, replacing the simpler item modeling component, and thus leading to significant improvements in prediction accuracy.

4.1.2 Autoencoders in reconstruction-based models

In this section, we introduce some examples of AEs used in reconstruction-based RS. *Autoencoder-based Collaborative Filtering* (ACF) (OUYANG *et al.*, 2014) is a user-specific AE-based RS that uses an item-sparse rating vector instead of the original integer rating as input data, where each position represents a rating value and only one position is non-zero. The prediction is computed based on the reconstructed values, which give the probability of the item being rated at each value represented in the vector. Stacking several autoencoders leads to better prediction accuracy, but it is computation-

ally expensive.

The *AutoRec* (SEDHAIN *et al.*, 2015) model has two variants: user-based and item-based. It uses an autoencoder to reconstruct the user rating vector and the item rating vector, respectively, and both approaches use a similar basic AE structure. The item-based version typically performs better because there are more ratings on average for each item than for each user, and increasing the number of hidden neurons generally improves performance.

Collaborative Filtering Neural Network (CFN) (STRUB *et al.*, 2016) incorporates user or item side information in an SDAE to enhance recommendations. This approach aims to address the cold-start problem, where the system has little to no rating data for a new user. The model architecture takes a corrupted user's sparse rating vector as its primary input. The core innovation lies in how side information (e.g., user demographic features like age and gender, or item attributes) is integrated into the learning process. The side information vector is injected directly at the model's bottleneck by concatenating it with the latent representation of the user's ratings learned by the encoder. This fused vector, which combines collaborative signals from ratings with content-based signals from user features, is then passed to the decoder to reconstruct the rating vector. By enriching the latent space with explicit features, the model can learn more robust user representations and make more reliable predictions for users with sparse histories.

The *Collaborative Denoising Auto-Encoder* (CDAE) (WU *et al.*, 2016) method adopts a denoising autoencoder with one hidden layer and is mainly used for ranking prediction. The model uses partial observed implicit feedback as input, and its corrupted version is drawn from a conditional Gaussian distribution. CDAE also incorporates a user-specific latent vector, which is explicitly added as an input node to the model's hidden layer. This effectively conditions the entire model on the identity of the user, allowing it to learn personalized preferences more effectively than a standard denoising autoencoder. A negative sampling technique is proposed to select items that the user did not interact with, aiming at reducing time complexity.

While foundational models like AutoRec demonstrated the potential of deep learning for CF, they often inherited the limitations of their predecessors, particularly a vulnerability to biases and inherently exhibited by user behavior data. User data is often skewed by multiple factors:

- **User Bias**: Users have different rating tendencies; a 4-star rating might be an ordinary score for one user but the highest possible for another, more critical user.
- **Item Bias**: Popular items tend to receive more exposure and ratings, leading to imbalanced data distributions.

- **Selection Bias**: Users typically rate items they are interested in or items with which they had a bad experience, meaning that the absence of a rating is not at random.
- **Outliers**: Malicious or unusually strict users can assign extremely high or low ratings that distort the underlying data patterns.

The *AutoRec++* model (HUANG *et al.*, 2024) was proposed as a comprehensive solution to these challenges, enhancing the standard AutoRec architecture with two key innovations: an integrated debiasing framework and a robust loss function.

AutoRec++ introduces the concepts of Preprocessing Bias (PB) and Training Bias (TB) to comprehensively address user and item biases. The PB components are calculated from the data before training and include the global average rating, item-specific deviations, and user-specific deviations, which are conceptually similar to the baseline estimates used in SVD models. These biases are subtracted from the input ratings before they are fed into the autoencoder. The TB components are parameters learned concurrently with the network weights. They are added back to the model's output before the final prediction, enabling the model to capture nuanced preferences during the training process. This joint approach was demonstrated to significantly enhance prediction accuracy and accelerate model convergence, without necessitating structural changes to the underlying autoencoder.

A second limitation of many DNN-based models is their reliance on an L_2-norm loss function (i.e., minimizing squared error), which is highly sensitive to outliers. While an L_1-norm loss is more robust to such outliers, it can suffer from instability during training. To gain the benefits of both, AutoRec++ employs a self-adaptively weighted loss function that combines the L_1 and L_2 norms. The objective function becomes:

$$L = \gamma_1 \cdot \|\text{error}\|_{L_1} + \gamma_2 \cdot \|\text{error}\|_{L_2}^2 + \text{regularization}. \tag{4.1}$$

The weights γ_1 and γ_2 are updated dynamically during training to favor the norm that is performing better, ensuring that the final model is both stable and robust to outliers. Empirical results demonstrated that this approach significantly improves the model's resilience to noisy data compared to other state-of-the-art methods.

4.2 Alternative Subspace Approaches: Biclustering

While autoencoders learn latent subspaces through neural network optimization, an alternative line of research uses biclustering (or co-clustering) to discover these

subspaces through combinatorial techniques. Biclustering differs from traditional clustering in that it simultaneously groups users and items, identifying *biclusters* — submatrices of the user-item matrix where a specific group of users exhibits coherent preferences across a specific group of items. This approach is well-suited to CF, as it directly addresses the problem of preference locality, where user preferences are often correlated only on a subset of items. Fig. 3 shows examples of contiguous biclusters in a binary matrix, such as the one outlined in red formed by the user subset $\{u_1, u_2\}$ and the item subset $\{i_2, i_3, i_4\}$, and the one outlined in green formed by the user subset $\{u_5, u_6\}$ and the item subset $\{i_4, i_5\}$. Note that biclusters do not need to be contiguous.

	i1	i2	i3	i4	i5	i6	i7
u1	1	1	1	1	0	1	1
u2	0	1	1	1	1	1	0
u3	0	1	1	0	0	0	0
u4	1	1	1	0	0	1	1
u5	0	0	0	1	1	0	0
u6	0	1	0	1	1	1	0
u7	0	1	0	0	1	0	1

Figure 3 – Examples of contiguous biclusters in a binary matrix

The *Biclustering-based Collaborative Filtering* (BBCF) (BBCF) framework uses biclustering as a preprocessing step to tackle the scalability and sparsity issues of memory-based CF (SINGH; MEHROTRA, 2018). The process involves several steps:

- A biclustering algorithm (e.g., QUBIC (*QUalitative BIClustering algorithm*) (LI *et al.*, 2009)) is run on the rating matrix to generate a set of biclusters.
- For each active user, the K-nearest biclusters are identified based on a similarity metric that considers the overlap between the user's rated items and the items within each bicluster.
- These K-nearest biclusters are merged to form a new, smaller, and denser user-item matrix, referred to as the *Nearest Neighbor Setup* (NSS).
- A traditional CF algorithm, such as item-based KNN, is then applied to this personalized NSS to generate predictions.

By operating on these smaller, denser subspaces instead of on the entire sparse matrix, BBCF can improve both scalability and accuracy.

A significant drawback of biclustering-based approaches, such as BBCF, is their limited coverage. Because biclusters may not encompass all users or items, the model may be unable to generate predictions for many user-item pairs. The *User-Specific Bicluster-based Collaborative Filtering* (USBCF) approach was designed to overcome this limitation while improving upon the core methodology (SILVA *et al.*, 2022).

USBCF introduces two key enhancements. First, it employs a more sophisticated user-bicluster similarity metric that considers not only the item overlap (match score) but also the correlation between the user's rating pattern and the bicluster's inherent preference pattern (fit score). The final similarity is a product of these two scores, ensuring that selected biclusters are highly relevant to the active user. Second, to address the coverage problem, USBCF incorporates a secondary prediction mechanism. For any user-item pair that falls outside a user's personalized bicluster-derived subspace, it falls back to a co-clustering model that provides an exhaustive partitioning of the rating space, guaranteeing 100% predictive coverage. Experiments showed that USBCF significantly increased predictive coverage compared to BBCF while achieving competitive accuracy against state-of-the-art matrix factorization and memory-based models.

4.3 Overview of Related Work

The application of latent factor and subspace models to collaborative filtering has evolved significantly, moving from linear matrix factorization to more powerful, nonlinear deep learning architectures and combinatorial approaches. Foundational matrix factorization models, such as SVD, have established the effectiveness of learning low-dimensional latent representations of users and items. Innovations such as the inclusion of baseline estimates and implicit feedback in models like SVD++ further refined this paradigm, leading to significant accuracy gains.

Autoencoders represent a natural extension of this philosophy in deep learning. This review covered several key architectures, including deterministic models such as AutoRec, deeper architectures like ACF, and hybrid autoencoders that incorporate side information. Furthermore, it explored generative models, such as CVAE, for fusing content information, ranking-focused models, like CDAE, that utilize denoising for robustness, and Graph Autoencoders that integrate KG topology directly into the learning process. Hybrid frameworks, such as AutoSVD++, further demonstrate the power of combining the strengths of matrix factorization with the nonlinear modeling capabilities of autoencoders.

However, the real-world utility of these models hinges on their ability to handle the imperfections of behavioral data. The AutoRec++ model offers a comprehensive framework for this, integrating debiasing techniques and a robust, hybrid loss function to systematically tackle data biases and outliers.

Finally, biclustering approaches, such as BBCF and USBCF, provide an alternative approach to addressing scalability and data sparsity. By identifying and modeling dense, coherent subspaces within the user-item matrix, these methods create personalized data environments that enable traditional CF algorithms to thrive. While they introduce their challenge in the form of predictive coverage, hybrid solutions, such as those in USBCF, demonstrate a viable path toward mitigating this issue.

However, even though biclustering has the potential to reduce the time complexity for the CF stage of RS, there is still a challenge in finding scalable solutions for biclustering itself (VERONEZE; VON ZUBEN, 2021). Motivated by this, we propose a method here that is inspired by the alternative subspace approach, but utilizes a different mining technique to leverage local information from the data: the BinaPs algorithm, an autoencoder-based pattern mining method introduced in Section 2.4. We can utilize the pattern set identified by BinaPs as a data partition and calculate rating predictions based on user-item similarities within each pattern, which essentially represents a subset of correlated features. Moreover, BinaPs' scalability and ability to cope with sparse datasets allow it to perform efficiently. This is the basis of the recommender system we propose here. To the best of our knowledge, this kind of autoencoder application in RS has not yet been explored in the literature, as we are not interested in using either latent-factor representations or the reconstructed output to calculate rating predictions.

4.4 APCF: Autoencoder Pattern-based Collaborative Filtering

We present here APCF, an autoencoder-based RS method using the BinaPs method, introduced in Section 2.4. It consists of a Collaborative Filtering method endowed with the IBKNN approach, which resorts to a pattern set obtained by the autoencoder to identify item subspaces. The innovative aspect of our proposal is that the autoencoder is applied in a distinct and promising paradigm, apart from the ones usually adopted in autoencoder-based RS, where either the latent representations or the reconstructed output are used to generate recommendations. Instead, the autoencoder here serves as a data mining technique, and we utilize the interpretable weight matrices to identify data partitions. By doing so, we expect APCF to leverage BinaPs' high scalability and robustness in handling sparse data, thereby achieving competitive performance and promising scalability properties when applied to recommendation tasks.

Since BinaPs requires binary data, we need to binarize the ratings matrix when working with non-binary values, which is often the case for recommendation problems. We define a binarization threshold t_{bin}. The binary matrix is obtained by setting 1 to entries whose rating is above t_{bin} and 0 elsewhere. Alternatively, we may use the implicit binary matrix, obtained by setting 1 to all entries whose rating is known, and 0 elsewhere.

Then, we train the autoencoder with the BinaPs method to obtain the pattern set for the binary dataset.

Once the patterns are obtained, we calculate the *user-pattern similarity* between each user u and each pattern p. This metric is given by dividing the number of items they have in common by the pattern's dimension (SYMEONIDIS *et al.*, 2007):

$$\text{sim}(u, p) = \frac{|I_u \cap p|}{|p|}, \tag{4.2}$$

where I_u is the set of items rated by user u. Depending on the problem, a pattern selection step may be necessary to filter spurious patterns. Here, we adopt a pattern selection based on the number of associated users. A user u is associated with a pattern p if $\text{sim}(u, p) > 0.5$. We select the patterns with at least two associated users.

To generate recommendations for a target user u_t, we first sort the selected patterns by user-pattern similarity and find the n_{pat} most similar patterns to u_t. Then, we merge these patterns in a subset of items M_{u_t} and evaluate the item similarity $\text{sim}(i, j)$ between each pair of items i and j in M_{u_t}, using the Adjusted Cosine Similarity given by Eq. (3.17).

To predict the rating P_{u_t,i_t} of item i_t given by the target user u_t, we find the neighborhood L of item i_t, defined by the k most similar items to i_t in the subset M_{u_t} that were rated by the target user u_t. Predictions are calculated as a weighted sum of ratings of similar items in L, given by Eq. (4.3):

$$P_{u_t,i_t} = \overline{R}[u_t] + \frac{\sum\limits_{j \in L} (R[u_t, j] - \overline{R}[u_t]) \cdot \text{sim}(i_t, j)}{\sum\limits_{j \in L} |\text{sim}(i_t, j)|}, \tag{4.3}$$

where $\overline{R}[u_t]$ is the average value for all items rated by target user u_t. The ratings are subtracted by this value as in the adopted similarity metric.

Finally, we define a threshold parameter denoted t_{rat}, defining the minimum rating for an item to be recommended. Items whose rating is strictly above t_{rat} are called *relevant* for a given user. Recommendations are generated for relevant rating predictions. Figure 4 shows the outline of the APCF rating prediction approach, and Algorithm 2 summarizes the main steps to calculate P_{u_t,i_t}.

In some cases, depending on the dataset and the pattern selection step, the neighborhood L may be empty for a given item i_t, meaning that no similar items can be found among the retained patterns for the target user u_t. If that occurs, an alternative approach may be adopted to calculate the prediction. Here, we use the average rating $\overline{R}[u_t]$ as the predicted rating in these cases.

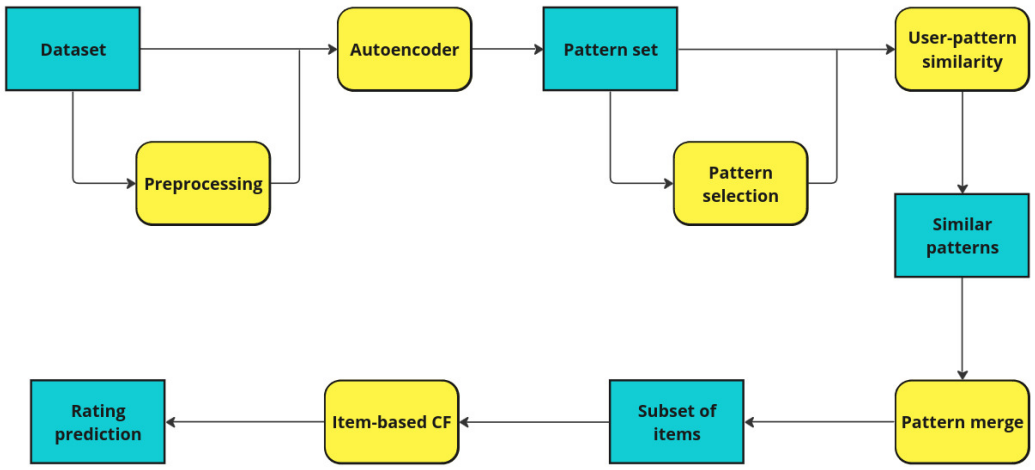

Figure 4 – Outline of the APCF rating prediction approach.

4.5 Summary

We started this chapter by presenting a literature review on autoencoder-based RS and related work. We introduced some notable examples of AE-based techniques, which usually follow two main paradigms for generating recommendations: latent-factor representations and reconstruction-based predictions. We also discussed biclustering as an alternative subspace approach for RS and presented the motivation for our proposed method: APCF (Autoencoder Pattern-based Collaborative Filtering). APCF is outlined as an alternative to alleviate some of the issues encountered in RS by utilizing the BinaPs algorithm, an efficient autoencoder for pattern set mining, as detailed in Chapter 2. We use the pattern set generated by the autoencoder as a data partition and apply an *Item-Based K-Nearest Neighbors* (IBKNN) approach to predict ratings based on item similarity within each partition. This way, APCF requires fewer computational resources for computing similarities than traditional KNN methods, which evaluate the metric for the whole dataset; furthermore, our method profits from BinaPs' scalability and robustness to sparsity.

Algorithm 2: APCF prediction

Input: dataset R, target (u_t, i_t), pattern set P, k, n_{pat}
Result: prediction P_{u_t,i_t}
for p **in** P **do** // For each pattern
  $S_{u_t,p} \leftarrow \text{sim}(u_t, p)$; // User-pattern similarity
end
$\text{sort}(P, S_{u_t})$; // Sort patterns by similarity
$P' \leftarrow \text{select\_pattern}(P, n_{pat})$; // Select similar patterns
$M_{u_t} \leftarrow \text{merge}(P')$; // Merge items
for j **in** M_{u_t} **do**
  $S_{i_t,j} \leftarrow \text{sim}(i_t, j)$; // Item similarity
end
$\text{sort}(M_{u_t}, S_{i_t})$; // Sort items by similarity
$M'_{u_t} \leftarrow \text{select\_rated}(R, u_t, M_{u_t})$; // Select items rated by u_t
$L \leftarrow \text{select\_neighbor}(M'_{u_t}, k)$; // Select k neighbors
$num \leftarrow 0, den \leftarrow 0$;
for j **in** L **do** // Weighted average
  $num \leftarrow num + (R[u_t, j] - \overline{R}[u_t]) \cdot S_{i_t,j}$;
  $den \leftarrow den + |S_{i_t,j}|$;
end
$pred \leftarrow \overline{R}[u_t]$;
if $L \neq \emptyset$ **then**
  $pred \leftarrow pred + num/den$;
end
$P_{u_t,i_t} \leftarrow pred$;

5 Computational experiments

This chapter presents the computational experiments conducted using APCF to evaluate its performance and compare it with that of other relevant recommender systems. We aim to answer the following research questions:

- **RQ1.** Can autoencoders be successfully applied as mining techniques to support and potentially enhance the performance of recommender systems?
- **RQ2.** How do the hyperparameters affect APCF's performance?
- **RQ3.** How well does APCF perform compared to other recommender systems?
- **RQ4.** Which application scenarios are most suited for APCF?

5.1 Performance evaluation of Recommender Systems

We present here the metrics used to evaluate the performance of Recommender Systems, which are usually adopted in the literature (SHANI; GUNAWARDANA, 2011).

For measuring the accuracy of predictions, we evaluate the *Mean Absolute Error* (MAE), which is the deviation of predictions from the actual ratings given by the user. MAE is given by Eq. (5.1):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - r_i|, \tag{5.1}$$

where p_i is the predicted rating for the i-th sample of the test set of size n and r_i is the actual rating.

Two frequently used metrics to evaluate the quality of recommendations are *Precision@N* and *Recall@N*. For each user u, we define R_u as the set of relevant items and P_u^N as the top N recommended items. *Precision@N* computes, for all users, the fraction of relevant recommendations among the top N recommended items. *Precision@N* is given by Eq. (5.2):

$$Precision@N = \frac{\sum\limits_{u\in U}|P_u^N \cap R_u|}{\sum\limits_{u\in U}|P_u^N|}, \tag{5.2}$$

where U is the set of users in the test set. On the other hand, *Recall@N* measures the fraction of retrieved recommendations among the relevant items. *Recall@N* is given by Eq. (5.3):

$$Recall@N = \frac{\sum\limits_{u \in U} |P_u^N \cap R_u|}{\sum\limits_{u \in U} |R_u|}. \tag{5.3}$$

The *Normalized Discounted Cumulative Gain* (NDCG) is a measure of ranking quality used to evaluate the performance of a search engine, recommender system, or other information retrieval system (WANG *et al.*, 2013). The principle of NDCG is that the more relevant items must be ranked better than the less relevant items. NDCG is maximized whenever the relevant items are ranked higher. There are many formulations to calculate the *Discounted Cumulative Gain* (DCG) for a list of n items. One of them is given by Eq. (5.4):

$$DCG = \sum_{i=1}^{n} \frac{\text{gain}(i)}{\log_2(i+1)}, \tag{5.4}$$

where $gain(i)$ is the graded relevance of the result at position i. The NDCG is calculated by normalizing the DCG by the ideal value $IDCG$, which is the best possible DCG value obtained for a perfect ranking. NDCG is given by Eq. (5.5):

$$NDCG = \frac{DCG}{IDCG}. \tag{5.5}$$

In the context of recommendations, we evaluate the *NDCG@N* metric for each user on the test set in the order induced by the predicted ratings, with the gain for each of the top N recommended items given by the actual rating. The gain of an index falling inside a tied group is replaced by the average gain within this group. Then, we take the average value of all users' *NDCG@N*.

Finally, the *Coverage* is the fraction of items for which the recommender system is able to calculate a prediction.

5.2 Experimental Setting

The experiments were divided into two parts. In the first part, we carry out three rounds of experiments on public and synthetic data, each focusing on answering specific research questions. In the second part, we perform experiments on private datasets to assess APCF's applicability in the context of health recommendations [1]. In all rounds, we compare APCF's performance with six other RS techniques: the baseline IBKNN, SVD, SVD++, NMF, USBCF, and AutoRec++. The baseline IBKNN method was included, as APCF's approach is based on it, allowing for a direct comparison of the improvements

[1] Implementation is available at: <https://github.com/pedro-mariano/APCF>. Datasets are available at <https://doi.org/10.25824/redu/40OKYE>.

made by our method. SVD, SVD++, and NMF are well-established methods for RS. USBCF and AutoRec++ stand out among recent RS approaches based on biclustering and autoencoders, respectively.

If not explicitly mentioned, we use the following parameter values for APCF: $t_{bin} = 0$, $t_{rat} = 2.5$, and we train the autoencoder for 50 epochs. We perform a hyper-parameter search for n_{pat} and k, and select the configuration that minimizes the MAE for APCF. For the BinaPs (FISCHER; VREEKEN, 2021), USBCF (SILVA *et al.*, 2022), and AutoRec++ (HUANG *et al.*, 2024) methods, we utilized the implementations provided by the authors. For the IBKNN, SVD, SVD++, and NMF methods, we utilize the implementations provided by the Surprise library (HUG, 2020). We adopted the same k parameter for APCF, IBKNN, and USBCF. In all cases, we use the default parameters, unless otherwise stated. We employ cross-validation in all experiments, using 5 splits, and report the average results on the test set. The experiments were run on Python 3.8.10 and carried out on the IARA supercomputer (SIDI, 2024), which is equipped with NVIDIA DGX A100 systems. We allocated 16 CPUs, 1 GPU, and 128GB of memory for all rounds.

5.3 Experiments: Part 1

We carried out three rounds of experiments for this section. In the first round, we use the *ml-latest-small* version of the MovieLens dataset (HARPER; KONSTAN, 2015), aiming to answer RQ1 and RQ2. This dataset describes 5-star ratings and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100,836 ratings and 3,683 tag applications across 9,724 movies. These data were created by 610 users between March 29, 1996, and September 24, 2018. All selected users had rated at least 20 movies. This is a highly sparse dataset, with a ratings matrix featuring around 1.7% of non-zero entries. To analyze the algorithm's behavior in different configurations, we use two values for the binarization threshold parameter: $t_{bin} = \{2.5, 3.5\}$, and train the autoencoder for 5,000 epochs. The recommendation threshold t_{rat} is equal to the binarization threshold t_{bin} in each case.

In the second round, we address RQ3 by comparing the performance of our method with that of the six competing methods. We use two datasets in this round. The first is a stable benchmark version of the MovieLens dataset, *ml-100k*, similar to the previously mentioned version. It was released in 1998 and contains 100,000 ratings from 943 users on 1,682 movies, with 6.3% of non-zero entries in the ratings matrix. The second one is the Jester 4 dataset (GOLDBERG *et al.*, 2001), which has 106,489 joke ratings from 7,699 users on 158 jokes, with ratings going from -10.00 to +10.00. This dataset requires preprocessing, as it features rows with single entries and some ratings outside the expected range. We removed users with single ratings, clamped the ratings

to the closest extreme value when necessary, and rescaled the ratings to the interval [0.1, 5.0]. The resulting ratings matrix has 6,475 rows and 105,265 ratings, around 10.3% of non-zero entries. We use the following parameter values for the autoencoder training on this dataset: learning rate $\gamma = 0.5$ and learning rate step $\eta = 0.5$.

In the final round, our goal is to answer RQ4. We design synthetic datasets to evaluate APCF's performance in scenarios where most of the relevant items for a given user are associated with a pattern in the ratings matrix, presenting different challenges.

We create such datasets in two steps. First, we produce a binary matrix with N_p artificial patterns and m rows, where each pattern has a random number of unique columns between 2 and p_c, drawn from a uniform distribution. Then, we select a random number from a normal distribution $\mathcal{N}(\rho m, \rho m/10)$, where ρ is a density parameter, and define its closest integer as the pattern's number of rows. It may be necessary to redraw this number until it falls in the interval $]0, m[$. Unlike columns, a row can be associated with different patterns. After that, we fill all patterns' entries with ones and apply a noise function to the whole matrix to switch an entry value with a probability of p_n. Finally, we remove any rows without content.

In the second step, we transform the binary matrix into a ratings matrix with values in the range $[min_r, max_r]$. We define a set of ratings evenly spaced in this interval, with a step parameter s_r. The positive ratings have values above the recommendation threshold t_{rat}, while the negative ones have values less than or equal to t_{rat}. For each of the matrix's non-zero entries, which are mostly inside patterns, we replace its value with a positive rating randomly chosen from the set of ratings. Once finished, we get n_{pos} positive ratings. Then, we select a number from a normal distribution $\mathcal{N}(n_{pos}, n_{pos}/10)$ and define its closest integer as n_{neg}, the number of negative ratings. Finally, we pick n_{neg} random zero entries from the matrix and replace each one with a random negative rating from the set of ratings.

Fig. 5 shows an example of the ratings matrix in a synthetic pattern-based dataset created using the steps described here. The rearranged version of the matrix allows us to see more clearly how users and items are correlated inside a pattern.

Table 1 shows the parameters used to generate the synthetic datasets. The first one, called *PatRec*, is a small and relatively dense dataset, comprising 300 rows, 1,068 columns, and 83,517 ratings, which represent approximately 26% of the total number of entries in the rating matrix. Here, we aim to evaluate the performance of RS in pattern-based data with a comparable size to the *ml-100k* and *Jester 4* datasets, but with less sparsity. In the second dataset, called *PatRec-large*, we aim to evaluate RS scalability by creating a much larger and sparser dataset, comprising 30,000 rows, 11,203 columns, and 9,254,598 ratings, which represent 2.75% of the total number of entries in the rating matrix.

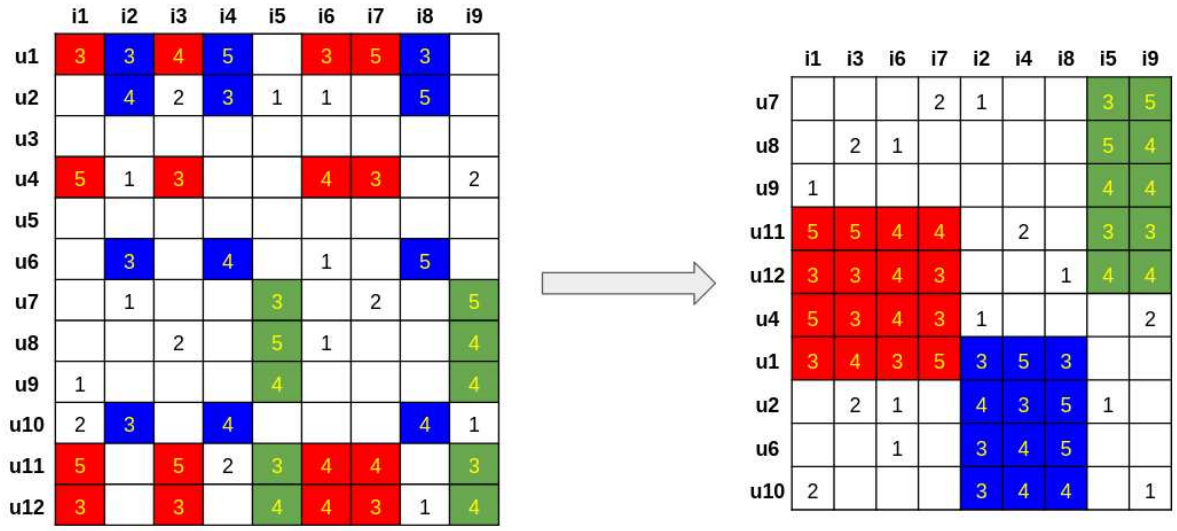

Figure 5 – Example of the ratings matrix in a synthetic pattern-based dataset with 12 users (u_1 to u_{12}), 9 items (i_1 to i_9), ratings in the scale $[0, 5]$ and $t_{rat} = 2.5$. Entries associated with a pattern have the same color. On the left, we see the matrix as created. On the right, the same matrix is displayed after rearranging rows and columns and removing empty rows.

Table 1 – Parameter values for the generation of the synthetic datasets

	Dataset	
Parameter	*PatRec*	*PatRec-large*
N_p	100	1,000
m	300	30,000
ρ	0.1	0.01
p_n	0.05	0.004
p_c	20	
min_r	0.0	
max_r	5.0	
s_r	0.5	
t_{rat}	2.5	

5.3.1 BinaPs training

First, we investigate how the autoencoder training and pattern set mining are affected by the choice of two BinaPs parameters: the maximum number of epochs, n_{ep}, and the initial learning rate, γ. In all cases, we used the default value of the learning rate step $\eta = 0.1$. We trained the autoencoder on the *ml-100k* dataset in three configurations and evaluated the evolution of the training loss over the epochs. Figure 6 shows the curve comparison, and Table 2 shows the total number of patterns and the number of selected patterns for each configuration.

Table 2 – Pattern set size for different BinaPs parameters on the *ml-100k* dataset

n_{ep}	γ	**Total**	**Selected**
50	$1 \cdot 10^{-2}$	344	166
500	$1 \cdot 10^{-2}$	289	144
50	$2 \cdot 10^{-5}$	0	0

In the first two configurations, represented by Fig. 6a and Fig. 6b, we used the default value of $\gamma = 1 \cdot 10^{-2}$ and different values of n_{ep}. We see that using an increased number of epochs does not lead to a decrease in loss, and Table 2 shows that the pattern size does not improve either. In the third configuration, we adopted a smaller value for the learning rate, and Figure 6c shows a smoother loss evolution. However, as shown in Table 2, when γ is too small, all weights tend to zero, and no patterns are found. Therefore, we select the first configuration for the pattern set mining step of APCF in most of the experiments.

Next, we investigate how the number of patterns is affected by the binarization threshold t_{bin}. We run experiments with $t_{bin} = \{0.0, 1.5, 2.5, 3.5, 4.5\}$ on the *ml-100k* dataset. Since the ratings in this dataset are integers in the interval $[1, 5]$, the binarization parameter values cover all possible subsets of relevant ratings, starting with the full set and progressively removing ratings from 1 to 5. This means that these t_{bin} values represent all binarization threshold values in an interval of size 1.0 beginning and ending at a valid rating, respectively: $[0.0, 1.0[$, $[1.0, 2.0[$, $[2.0, 3.0[$, $[3.0, 4.0[$ and $[4.0, 5.0[$. Table 3 shows the average pattern set size obtained for each value of t_{bin} before and after pattern selection, as well as the average number of items per pattern and training samples. We observe that the number of patterns increases with t_{bin} until it reaches a maximum at 3.5, illustrated in Fig. 7. As the number of training samples decreases, data becomes more fragmented, and the autoencoder finds a higher number of patterns with fewer items per pattern. However, for a very low number of training samples, as is the case for $t_{bin} = 4.5$, this behavior changes, and the autoencoder returns to finding fewer patterns with more items per pattern.

(a) $n_{ep} = 50, \gamma = 1 \cdot 10^{-2}$

(b) $n_{ep} = 500, \gamma = 1 \cdot 10^{-2}$

(c) $n_{ep} = 50, \gamma = 2 \cdot 10^{-5}$

Figure 6 – Evolution of training loss over the epochs for different BinaPs parameters on the *ml-100k* dataset

Table 3 – Average number of patterns on the *ml-100k* dataset for different values of t_{bin} and the corresponding average of item/pattern and number of training samples

t_{bin}	Total patterns	Selected patterns	Item/pattern	Training samples
0.0	341.2	163.2	13.51	80,000
1.5	396.2	202.4	11.96	75,112
2.5	492.2	266.8	10.62	66,016
3.5	728.0	487.6	9.08	44,300
4.5	439.2	387.0	16.65	17,068.6

In the next section, we also analyze how recommendations are affected by

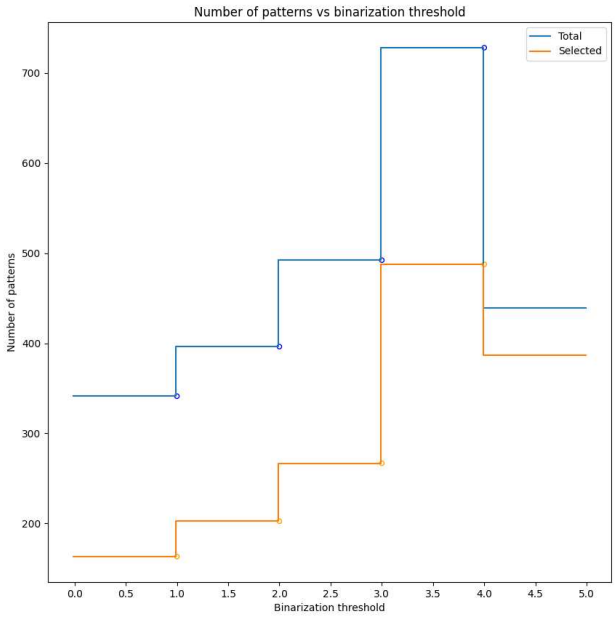

Figure 7 – Average pattern set size on the *ml-100k* dataset for different values of t_{bin}

different values of the binarization threshold.

5.3.2 Recommendation results

Tables 4 and 5 show the results for the experiments with APCF on the *ml-latest-small* dataset for $t_{rat} = \{2.5, 3.5\}$ respectively, and different values for n_{pat} and k, with the best results highlighted in bold. The method has better precision for $t_{rat} = 2.5$ than for $t_{rat} = 3.5$, while the values for the other metrics are close for both cases. Nevertheless, recall is slightly better for $t_{rat} = 3.5$, and a possible reason is that we have fewer relevant items compared to the other case, making them less likely to be found outside the top 25 recommendations.

Regarding the n_{pat} parameter, we see that performance is optimal for $n_{pat} = 20$ in terms of accuracy (MAE). For precision, recall, NDCG, and coverage, $n_{pat} = 30$ is the best configuration, except for recall when $t_{rat} = 3.5$, in which case the best setting is $n_{pat} = 10$.

As for the size of the neighborhood k, we see that the performance is worse for small values of k. The method performs better when k increases up to a certain value, after which performance stops improving. This limit value varies according to the case. For $t_{rat} = 2.5$, the limit is $k = 400$ for the accuracy and $k = 600$ for the precision, recall, and NDCG. For $t_{rat} = 3.5$, the limit is $k = 300$ for the accuracy and $k = 400$ for the precision and NDCG. Again, the recall in this case is the exception since we observe the best result for the smallest value of k. This parameter doesn't affect the coverage.

Table 4 – Results of the experiments with APCF on the *ml-latest-small* dataset for MAE, Precision@25, Recall@25, NDCG@25, and Coverage using $t_{rat} = 2.5$

n_{pat}	k	MAE	Precision (%)	Recall (%)	NDCG	Coverage (%)
10	100	0.679374	89.153516	50.015450	0.949068	95.774326
	200	0.672388	89.352558	50.126746	0.949728	95.774326
	300	0.672388	89.352558	50.126746	0.949728	95.774326
	400	0.672388	89.352558	50.126746	0.949728	95.774326
	500	0.672388	89.352558	50.126746	0.949728	95.774326
	600	0.672388	89.352558	50.126746	0.949728	95.774326
	700	0.672388	89.352558	50.126746	0.949728	95.774326
20	100	0.684528	89.204965	50.035055	0.950425	95.883414
	200	0.672808	89.409474	50.146330	0.951069	95.883414
	300	0.669728	89.520796	50.208825	0.951552	95.883414
	400	**0.669596**	89.536096	50.217393	0.951609	95.883414
	500	**0.669596**	89.536096	50.217393	0.951609	95.883414
	600	**0.669596**	89.536096	50.217393	0.951609	95.883414
	700	**0.669596**	89.536096	50.217393	0.951609	95.883414
30	100	0.681693	89.243388	50.070673	0.950510	**95.896307**
	200	0.675910	89.424345	50.163447	0.950908	**95.896307**
	300	0.671478	89.500898	50.206313	0.951371	**95.896307**
	400	0.669996	89.577287	50.249202	0.951583	**95.896307**
	500	0.669833	89.594805	50.259016	0.951618	**95.896307**
	600	0.669793	**89.596987**	**50.260238**	**0.951630**	**95.896307**
	700	0.669793	**89.596987**	**50.260238**	**0.951630**	**95.896307**

Table 6 shows the results for the experiments with APCF on the *PatRec* dataset for $t_{rat} = 2.5$. APCF has full coverage for all parameter configurations on this dataset. The optimal value of n_{pat} is 40 for accuracy, 50 for precision, and 50 for recall. Once again, performance improves as the k parameter increases up to a limit value of 200 in this case. This dataset is more challenging than *MovieLens*, as the values for MAE, precision, and NDCG in this experiment were reduced.

The results presented in Table 7 show how APCF's performance is affected by the binarization threshold t_{bin} on the *ml-100k* dataset, without changing the relevance threshold $t_{rat} = 2.5$. We conducted a grid search to minimize MAE and selected $n_{pat} = 20$ and $k = 400$. The best results are highlighted in bold, and we present the p-value from the t-test comparing the averages of the two best results for each metric. We observe similar results regardless of the value of t_{bin} for MAE, Precision, Recall, and NDCG; the Coverage remains unaffected. Furthermore, we don't have sufficient evidence to reject the null hypothesis at a significance level of 0.01 in any case, indicating that the differences in the results are not statistically significant. Therefore, the choice of the binarization threshold has a minor impact on APCF's recommendation performance.

In summary, APCF's performance is consistent, presenting similar results for different parameter configurations. We achieve better coverage values when the number

Table 5 – Results of the experiments with APCF on the *ml-latest-small* dataset for MAE, Precision@25, Recall@25, NDCG@25, and Coverage using $t_{rat} = 3.5$

n_{pat}	k	MAE	Precision (%)	Recall (%)	NDCG	Coverage (%)
10	100	0.676828	68.830354	**51.626300**	0.953822	95.747550
	200	0.672408	69.167548	51.578690	0.954193	95.747550
	300	0.672408	69.167548	51.578690	0.954193	95.747550
	400	0.672408	69.167548	51.578690	0.954193	95.747550
	500	0.672408	69.167548	51.578690	0.954193	95.747550
	600	0.672408	69.167548	51.578690	0.954193	95.747550
	700	0.672408	69.167548	51.578690	0.954193	95.747550
20	100	0.684091	69.057416	51.564922	0.954124	95.866555
	200	0.669766	69.759817	51.407418	0.955154	95.866555
	300	**0.668931**	69.908672	51.429953	0.955229	95.866555
	400	**0.668931**	69.908672	51.429953	0.955229	95.866555
	500	**0.668931**	69.908672	51.429953	0.955229	95.866555
	600	**0.668931**	69.908672	51.429953	0.955229	95.866555
	700	**0.668931**	69.908672	51.429953	0.955229	95.866555
30	100	0.684217	69.293168	51.478813	0.954464	**95.876472**
	200	0.672533	69.743739	51.249008	0.955258	**95.876472**
	300	0.669675	70.132271	51.269842	0.955652	**95.876472**
	400	0.669328	**70.191442**	51.288258	**0.955704**	**95.876472**
	500	0.669328	**70.191442**	51.288258	**0.955704**	**95.876472**
	600	0.669328	**70.191442**	51.288258	**0.955704**	**95.876472**
	700	0.669328	**70.191442**	51.288258	**0.955704**	**95.876472**

of similar patterns n_{pat} increases, as more items can be included in the neighborhood. However, that does not necessarily improve the overall performance, suggesting that not all patterns are useful.

In Table 8, we can see the performance comparison between APCF and the other methods on multiple datasets. We use $N = 10$ to assess performance metrics in the *Jester 4* dataset, and $N = 25$ in the others. The best results are highlighted in bold, and we present, for each metric, the p-value from the t-test comparing APCF's results with the best contender. On the *PatRec-large* dataset, USBCF and AutoRec++ weren't able to run due to requiring an excessive amount of memory. Our method is usually outperformed by other autoencoder-based RS, AutoRec++. At a significance level of 0.01, APCF's Recall on the *ml-100k* dataset and NDCG on the *Jester 4* dataset have no statistically significant difference from the best result. Overall, APCF's performance is similar to SVD++ and NMF, and better than the baseline IBKNN, except on the *ml-100k* dataset. On the *PatRec* dataset, APCF outperforms all the other methods, except AutoRec++. Finally, on the *PatRec-large* dataset, APCF achieves the best performance among the methods that were able to run on this dataset. In scenarios where users and items exhibit strong local correlation, such as in the synthetic datasets, APCF can leverage local information obtained through the autoencoder-based pattern mining technique, becoming more

Table 6 – Results of the experiments with APCF on the *PatRec* dataset for MAE, Precision@25, Recall@25, NDCG@25, and Coverage for $t_{rat} = 2.5$

n_{pat}	k	MAE	Precision (%)	Recall (%)	NDCG	Coverage (%)
30	100	1.178056	64.628625	54.683611	0.901422	100.000000
	200	1.178032	64.628588	54.683648	0.901487	100.000000
	300	1.178032	64.628588	54.683648	0.901487	100.000000
40	100	1.176440	65.650619	55.535270	0.903092	100.000000
	200	**1.174722**	65.764108	55.631530	**0.903917**	100.000000
	300	**1.174722**	65.764108	55.631530	**0.903917**	100.000000
50	100	1.184175	65.687565	55.522913	0.901730	100.000000
	200	1.176940	**66.001936**	**55.777814**	0.902981	100.000000
	300	1.176940	**66.001936**	**55.777814**	0.902981	100.000000
60	100	1.192979	65.649446	55.524581	0.901663	100.000000
	200	1.180222	65.794859	55.598239	0.902178	100.000000
	300	1.180222	65.794859	55.598239	0.902178	100.000000

Table 7 – APCF's performance comparison on the *ml-100k* dataset for $t_{rat} = 2.5, N = 25, n_{pat} = 20, k = 400$ and different values of t_{bin}

t_{bin}	MAE	Precision (%)	Recall (%)	NDCG	Coverage
0.000000	0.756004	88.200616	72.532970	0.953449	0.998310
1.500000	0.755503	88.151900	72.485705	**0.953773**	0.998310
2.500000	**0.755382**	88.163212	72.518332	0.953286	0.998310
3.500000	0.756686	**88.204030**	**72.566956**	0.953250	0.998310
4.500000	0.760370	88.109271	72.555157	0.953402	0.998310
p-value	0.975168	0.991130	0.981878	0.703921	-

competitive compared to other RS.

Table 9 shows the runtime comparison for the contender methods. APCF's measures show the total runtime and the time consumed only by the CF step, excluding the autoencoder training. We observe that APCF typically takes longer to complete execution than SVD, IBKNN, and NMF, but it is faster than USBCF, SVD++, and AutoRec++, except for the *Jester 4* dataset, where the last two are faster. In the *PatRec-large* dataset, APCF is not only able to run on a very large dataset with more than 9 million ratings, unlike its biggest competitor, AutoRec++, but also runs faster than SVD++ and achieves the best performance among the contenders. Furthermore, for subsequent evaluations, APCF becomes a more interesting alternative. With the pattern set already in place, there is no need to retrain the autoencoder, unless a substantial amount of new incoming data becomes available. Disregarding *BinaPs* training step runtime, APCF is faster than SVD++ and AutoRec++ on all datasets presented here.

Table 8 – Performance comparison of recommender systems on multiple datasets

Dataset	Method	MAE	Precision (%)	Recall (%)	NDCG	Coverage (%)
MovieLens ml-100k $n_{pat} = 20$ $k = 400$	APCF	0.751227	88.783964	71.807024	0.950558	99.831000
	IBKNN	0.735041	88.868556	72.292093	0.957878	99.831000
	SVD	0.737384	88.670740	72.512241	0.956233	100.000000
	SVD++	0.721346	89.109474	72.623712	0.958931	100.000000
	NMF	0.758896	88.646647	71.730110	0.952294	99.831000
	USBCF	0.768217	87.269675	72.201030	0.952475	85.602000
	AutoRec++	**0.690581**	**89.999611**	**72.804201**	**0.963240**	100.000000
	p-value	0.000001	0.003194	0.058144	0.000001	-
Jester 4 $n_{pat} = 50$ $k = 100$	APCF	0.811684	80.036612	**67.341679**	0.944229	99.462320
	IBKNN	0.827347	79.747807	64.299466	0.941390	99.462309
	SVD	0.832967	78.973193	63.832899	0.941194	100.000000
	SVD++	0.811571	**80.681903**	62.504920	**0.947170**	100.000000
	NMF	0.848747	79.756047	58.869508	0.939698	99.462309
	USBCF	0.810116	80.102738	63.602980	0.944912	99.295112
	AutoRec++	**0.793760**	79.919660	65.871261	0.946969	100.000000
	p-value	0.001560	0.000001	0.000001	0.028310	-
PatRec $n_{pat} = 40$ $k = 200$	APCF	1.174722	65.764108	55.631530	0.903917	100.000000
	IBKNN	1.227991	59.261153	47.908923	0.876576	100.000000
	SVD	1.211778	60.197250	50.825969	0.877366	100.000000
	SVD++	1.230535	60.138449	50.668080	0.875551	100.000000
	NMF	1.230938	57.754786	48.629049	0.865885	100.000000
	USBCF	1.202294	61.142957	52.470978	0.900385	31.532495
	AutoRec++	**1.095860**	**77.425919**	**59.702281**	**0.930036**	100.000000
	p-value	0.000001	0.000001	0.000001	0.000001	-
PatRec-large $n_{pat} = 15$ $k = 12$	APCF	**1.077296**	**76.405798**	**60.412165**	**0.933895**	100.000000
	IBKNN	1.289786	48.194400	37.621001	0.832244	100.000000
	SVD	1.088219	74.396401	58.603596	0.919069	100.000000
	SVD++	1.128344	68.959483	54.364312	0.896193	100.000000
	NMF	1.204516	59.595499	46.504029	0.876132	100.000000
	USBCF	-	-	-	-	-
	AutoRec++	-	-	-	-	-
	p-value	0.000001	0.000001	0.000002	0.000001	-

Table 9 – Runtime comparison of recommender systems on multiple datasets

	Dataset			
Method	*MovieLens ml-100k*	*Jester 4*	*PatRec*	*PatRec-large*
APCF (total)	1m36s	1m44s	1m08s	3h46m05s
APCF (CF only)	57s	15s	37s	2h06m59s
IBKNN	15s	**7s**	20s	33m21s
SVD	**5s**	**7s**	**3s**	**6m57s**
SVD++	1m56s	35s	2m12s	4h44m43s
NMF	6s	9s	4s	8m23s
USBCF	28m13s	4h58m25s	39m08s	-
AutoRec++	3m18s	1m43s	1m13s	-

In conclusion, APCF's main drawback is its inability to generate a prediction or recommendation for some users/items (See the coverage column in Table 8). This handicap occurs because the subspace induced by the patterns may not fully cover the original user-item data space. As we have announced, the imputation of those unpredictable entries is filled with the average rating $\overline{R}[u_t]$. However, in most cases, APCF leverages BinaPs' high scalability and robustness to sparsity, enabling our method to provide effective recommendations and making it a solid option for various recommendation scenarios.

5.4 Experiments: Part 2

In the second part, we want to assess APCF's applicability in providing health recommendations in two contexts: physical activity recommendations and sarcopenia diagnosis. For this purpose, we compare APCF's performance with the contender methods on two datasets organized by the *Viva-Bem* group [2], which will be detailed in the following sections.

5.4.1 Physical activity recommendations

In this experiment, we analyze APCF's recommendations for physical activity. The dataset adopted for this experiment consists of data collected from 52 volunteers performing three physical activities of varying intensity: walking, trotting, and jogging. The activities were executed in a row in the aforementioned order for 5 minutes each, totalizing 15 minutes of physical exercise. At the beginning and the end of each of the three phases, the volunteers were asked to rate the *Subjective Perception of Effort* (SPE) when performing the activity on a scale from 1 - low effort to 10 - extreme effort. The experiment was repeated twice at different moments, totaling up to 12 different ratings for each volunteer. The data was collected by the *Viva-Bem* group. We will refer to this dataset as the SPE dataset.

Table 10 shows some entries of the SPE dataset, each line representing the ratings given by a different user. We see that data formatting is not uniform. Not all users rated each activity at its beginning and end, and some of them didn't show up to repeat the experiment at a second moment. The first step of the computational experiments consisted of preprocessing this dataset. We filled the missing entries with zeros and took the average rating when two ratings were given for the same activity. This resulted in a ratings matrix of dimension 52×6 with ratings in the range $[0, 8.0]$, with approximately

[2] The protocol for data capture, storage, and use was approved by the Unicamp Research Ethics Committee (CAAE's 55532622.0.0000.5404 and 66967022.2.0000.5404). Issues related to Intellectual Property, Patents, Registrations, and data co-ownership are handled jointly by the Viva Bem project team and Samsung.

50% of the ratings below 3.0. This implies that many volunteers classified the activities as taking low effort.

Table 10 – Sample entries of the SPE dataset

First moment			Second moment		
Walking	Trotting	Jogging	Walking	Trotting	Jogging
2	3	6	1	3	5
2	4	5	1/1	2/2	4/4
3	4	7	3/3	4/4	5/6
1	2	3	-	-	-
1/2	3/3	6/8	1/2	4/4	7/8

In the next step, we proceeded to the pattern set mining of the SPE dataset with BinaPs. We set the binarization threshold $t_{bin} = 3.0$, so we get a binarized dataset with sparsity around 50% to reflect the original data's behavior. We found 4 patterns on average. In some cases, two of them included all or almost all of the 6 columns, most likely representing the portion of users who classified all activities with similar effort. The remaining patterns included only the columns corresponding to trotting and jogging, representing users who rated these activities with high effort.

Subsequently, we proceeded to the CF step to get recommendations for high-effort activities, i.e., activities with predicted ratings above $t_{rat} = 3.0$. Since the SPE dataset is relatively small, we made a few modifications to the original APCF's prediction scheme of P_{u_t,i_t}. First, we do not perform pattern selection, and all patterns can be considered for the user-pattern similarity evaluation. Second, all items are eligible for inclusion in the neighborhood L of the item i_t, even if they do not belong to the subset M_{u_t} of items in the retained patterns. Instead, we use a modified version of Eq. (3.19) to calculate the predictions, including a parameter $\alpha_{u_t,j}$ that penalizes the similarity between the target item i_t and an item j that does not belong to M_{u_t}:

$$P_{u_t,i_t} = \overline{R}[i_t] + \frac{\sum\limits_{j \in L} (R[u_t, j] - \overline{R}[j]) \cdot \alpha_{u_t,j} \cdot \text{sim}(i_t, j)}{\sum\limits_{j \in L} |\text{sim}(i_t, j)|}, \tag{5.6}$$

where $\alpha_{u_t,j}$ is given by:

$$\alpha_{u_t,j} = \begin{cases} 1 & \text{if } j \in M_{u_t}; \\ \delta & \text{if } j \notin M_{u_t}, 0 \leq \delta \leq 1. \end{cases} \tag{5.7}$$

We adopted the cosine similarity given by Eq. (3.15) for evaluating item similarity and used $\delta = 0.8$. After a hyperparameter search, we selected the number of similar patterns $n_{pat} = 2$ and the size of item neighborhood $k = 2$. We evaluated the average values of MAE, Precision @6, Recall @6, NDCG @6, and Coverage for APCF, as well

as the contender methods. Table 11 presents the results for recommendations of high effort activities, with the best values highlighted and the reported p-value from the t-test comparing APCF's results with the best contender.

Table 11 – Performance comparison of recommender systems on the SPE dataset for high effort activities ($t_{rat} = 3.0$)

Method	**MAE**	**Precision (%)**	**Recall (%)**	**NDCG**	**Coverage**
APCF	0.611301	72.765517	97.019324	0.987735	100.000000
IBKNN	0.612954	72.248276	97.019324	0.987735	100.000000
SVD	0.730624	78.307692	96.178054	0.984204	100.000000
SVD++	0.682508	76.905116	95.066943	0.985874	100.000000
NMF	0.587530	**79.984615**	92.955832	**0.992656**	100.000000
USBCF	**0.568369**	72.235140	97.019324	0.990787	100.000000
AutoRec++	0.633530	70.353950	**97.178054**	0.989520	100.000000
p-value	0.093364	0.137973	0.927480	0.384490	-

The results in Table 11 show that all methods have similar performance on this dataset. We don't have sufficient evidence to reject the null hypothesis at a significance level of 0.01 in any case, indicating that the differences in the results are not statistically significant. Although intended for use on large datasets, we find that APCF also performs well in a real-world problem with reduced-size data, achieving an average precision of over 72%. This corroborates the applicability of APCF in the context of physical activity recommendations.

5.4.2 Sarcopenia Diagnosis

Sarcopenia is a muscle disorder characterized by adverse changes in muscle strength, quantity, and quality that occur throughout a person's lifetime. It is common in older adults but can also occur earlier in life. According to the European consensus, the screening and diagnosis of sarcopenia follow a pathway to find cases, assess, confirm, and determine the severity (CRUZ-JENTOFT *et al.*, 2018). Typically, the process begins with the application of the SARC-F questionnaire, which comprises 5 self-reported questions by patients as a screening tool for sarcopenia risk. Then, muscle strength is evaluated by measuring grip strength and the time to complete the chair stand test (also called the *sit-to-stand* test), in which the patient rises five times from a seated position without using their arms. At this point, it is possible to identify probable sarcopenia.

To confirm the diagnosis, the muscle quantity is estimated, reported as the *Skeletal Muscle Mass* (SMM) or *Appendicular Skeletal Muscle Mass* (ASM), for instance. The severity is assessed with the help of physical performance tests, such as measuring the walking speed and the time to complete the *Timed-Up and Go* test (TUG), in which the patients need to rise from a chair, walk a certain distance, come back and sit down again.

The *Viva-Bem* group collected information from 59 individuals to apply the sarcopenia diagnosis pathway. The dataset contains the answer to the SARC-F questionnaire, the estimated ASM, the walking speed measure, and the result of the grip strength, sit-to-stand, and TUG tests. There are 14 features in total. The first five are binary and contain the answers to each one of the SARC-F questions, SARC-F^1 to SARC-F^5. The sixth feature is the value of ASM. The remaining 8 features correspond to the grip strength and the physical performance evaluations, assessed in two distinct moments each: Grip strength^t, Walking speed^t, TUG^t, and Sit-to-stand^t, where $t = \{1, 2\}$ represents the moment. The resulting dataset, which we refer to here as *Sarcop-D*, is sparse and contains 8.8% non-zero entries.

For this application, we aim to evaluate APCF's capacity to provide recommendations for personalized pathways in sarcopenia diagnosis. In other words, we want to find which evaluations or tests should be prioritized for each patient based on the European guidelines. First, we binarized the real-valued features of the *Sarcop-D* dataset using the sarcopenia cut-off points established in the literature for each test (CRUZ-JENTOFT *et al.*, 2018), summarized in Table 12.

Table 12 – European sarcopenia cut-off points

Test	**Sex**	**Cut-off**
Grip strength	M	< 27 kgf
	F	< 16 kgf
Sit-to-stand		> 15s
ASM	M	< 20 kg
	F	< 15 kg
Walking speed		≤ 0.8 m/s
TUG		≥ 20 s

After preprocessing, we trained the autoencoder on the *Sarcop-D* dataset and identified 5 patterns. The prevalence of features in the patterns is shown in Table 13. We observe that the SARC-F questions and the features related to muscle strength (grip strength and sit-to-stand) are among the most frequently reported features. This aligns with the relevance of these features for sarcopenia diagnosis as reported in the literature.

Table 13 – Prevalence of features in patterns for the *Sarcop-D* dataset

Features	**Prevalence (%)**
SARC-F^4 , Grip strength^1	100
SARC-F^3	80
SARC-F^2, SARC-F^5	60
SARC-F^1, Grip strength^2, Sit-to-stand^1, Sit-to-stand^2	40
ASM, TUG^1, TUG^2, Walking speed^1, Walking speed^2	0

For the CF step, we performed a hyperparameter search and selected $n_{pat} = 3$ and $k = 10$ for APCF. Table 14 shows the performance comparison with the contender

methods on the *Sarcop-D* dataset for MAE, Precision @6, Recall @6, and Coverage for $t_{rat} = 0.5$, and the p-value from the t-test comparing APCF's results with the best contender. Since the recommendations are being compared on a binary dataset, we do not evaluate the NDCG, as all relevant features have the same rating.

This dataset poses a challenging recommendation task, as shown by the low Precision and Recall values for all methods. USBCF was unable to find any biclusters in this dataset. At a significance level of 0.01, NMF has the best MAE, but there is no statistically significant difference in Precision and Recall between APCF and the best contender. Overall, despite presenting a small MAE, the values of Precision and Recall reveal that the accuracy of these methods heavily relies on predicting negative ratings, which are the most frequent. The applicability of recommender systems in this context needs to be further investigated, for instance, by performing a qualitative analysis of the recommendations by a specialist. Nevertheless, APCF's results show a promising perspective, especially because of the potentially insightful analysis that can be derived from pattern-set mining results.

Table 14 – Performance comparison of recommender systems on the *Sarcop-D* dataset

Method	**MAE**	**Precision (%)**	**Recall (%)**	**Coverage**
APCF	0.156786	**29.487179**	**19.310572**	100.000000
IBKNN	0.159162	14.318182	10.276704	100.000000
SVD	0.163630	20.000000	1.538462	100.000000
SVD++	0.158575	0.000000	0.000000	100.000000
NMF	**0.110769**	20.000000	1.176471	100.000000
USBCF	-	-	-	-
AutoRec++	0.133122	5.000000	1.333333	100.000000
p-value	0.004556	0.665241	0.085154	-

5.5 Summary

In this chapter, we presented the computational experiments performed with APCF, aiming at validating autoencoder applicability as a pattern mining technique in recommender systems, assessing the model's performance in different recommendation scenarios, and comparing it with other recommender systems in the literature: IBKNN, SVD, SVD++, NMF, USBCF and AutoRec++. In the first part, we conducted experiments on benchmark datasets for movie and joke recommendations, as well as pattern-based synthetic datasets. APCF's performance was consistent, presenting similar results for different parameter configurations, and was comparable to that of the model-based methods on benchmarks. APCF's main drawback was its inability to generate a recommendation for some users/items. On the synthetic datasets, APCF was able to leverage local information and outperformed the contenders in a large dataset. When compar-

ing runtime, we demonstrated APCF's scalability, as it was able to run efficiently on a large dataset. In the second part, we assessed APCF's ability to provide health recommendations for physical activity and sarcopenia diagnosis. Overall, although intended for use on large datasets, APCF performed similarly to the contender methods in real-world problems with reduced-size data, reassuring its robustness and efficiency in diverse recommendation scenarios.

6 Conclusion

The ever-increasing amount of information in the digital age has made recommender systems (RS) essential tools for helping users discover items of interest amidst a multitude of choices. However, the development of these systems faces significant challenges, notably the sparsity of rating matrices and the scalability to large volumes of data. This thesis addressed these challenges by proposing a novel collaborative filtering method.

The proposed method, named Autoencoder Pattern-based Collaborative Filtering (APCF), introduces an innovative approach to applying autoencoders (AEs) in recommender systems. Unlike traditional paradigms that use autoencoders to extract latent factors or to directly reconstruct ratings, APCF employs an autoencoder as a data mining technique. Specifically, the BinaPs algorithm, a robust autoencoder for pattern set mining on binary data, is used to identify highly informative data partitions—subspaces formed by correlated items. Subsequently, a collaborative filtering algorithm based on items, the Item-Based K-Nearest Neighbors (IBKNN), operates within these subspaces to generate predictions more efficiently and accurately.

The research was guided by four central questions, which were systematically investigated through a series of computational experiments, as detailed in Chapter 5. We now recap and discuss the main conclusions.

- **RQ1.** Can autoencoders be successfully applied as mining techniques to support and potentially enhance the performance of recommender systems?

The experimental results confirm that applying autoencoders as a mining tool is a viable and effective strategy. The proposed method proved to be competitive across a wide range of recommendation scenarios, involving datasets of different sizes and degrees of sparsity. Furthermore, by being based on a gradient-descent mining process, APCF demonstrated a better scalability trade-off than the top competing techniques as the dataset size increased. The use of the BinaPs algorithm allowed for the identification of coherent item patterns even in sparse datasets, and the subsequent application of a collaborative filtering method on these subspaces resulted in competitive performance, validating the proposed paradigm as a solid alternative to conventional approaches.

- **RQ2.** How do the hyperparameters affect APCF's performance?

The analysis revealed that APCF's performance is notably consistent across different hyperparameter configurations. The number of neighboring patterns (n_{pat}) was

shown to impact coverage, with larger values allowing more items to be considered for recommendation. The neighborhood size (k) influenced accuracy, with performance improving up to a certain limit. A key finding was that the binarization threshold (t_{bin}), despite significantly altering the number of mined patterns, had a minimal impact on the final recommendation quality, indicating the method's robustness concerning data preprocessing.

- **RQ3.** How well does APCF perform compared to other recommender systems?

APCF demonstrated competitive performance in multiple scenarios. In public benchmarks like *MovieLens* and *Jester*, its performance was comparable to well-established methods like SVD++ and NMF, and superior to the IBKNN baseline. Although it was generally outperformed by another autoencoder model, AutoRec++, on these datasets, APCF exhibited a crucial advantage in scalability. On a large-scale synthetic dataset, *PatRec-large*, APCF not only outperformed all competitors that could run the task but also did so with a lower execution time than SVD++. At the same time, AutoRec++ failed due to requiring an excessive amount of memory. This highlights APCF's potential for large-scale applications.

- **RQ4.** Which application scenarios are most suited for APCF?

The experiments indicate that APCF is particularly well-suited for large, sparse datasets where strong local correlations (user-item patterns) exist. This was demonstrated on synthetic datasets designed with this characteristic, where APCF excelled. Furthermore, the method proved to be robust and versatile, achieving performance similar to that of competitors in real-world problems with reduced-size data, such as in healthcare applications for physical activity recommendations and aiding sarcopenia diagnosis.

Based on the work developed, the main contributions of this thesis are:

- The proposal of a new paradigm for the use of autoencoders in recommender systems, focusing on pattern mining for the discovery of data subspaces, rather than the extraction of latent factors or the reconstruction of ratings;
- The development of the APCF method, which efficiently integrates the BinaPs pattern mining algorithm with a neighborhood-based collaborative filtering approach;
- An extensive comparative evaluation of APCF against classic and recent methods, which demonstrated its competitiveness and identified an application niche in large-scale and sparse scenarios;

- The demonstration of the scalability and applicability of APCF in real-world domains, including healthcare, where the interpretability of the mined patterns can also offer valuable insights.

One potential disadvantage of the APCF method is its limited coverage in certain cases. Because predictions are generated from subspaces defined by patterns, user-item pairs that fall outside these subspaces do not receive a calculated recommendation and must rely on a fallback value, e.g., the user's average rating.

This limitation opens promising paths for future work. A natural direction would be the development of a hybrid model, which utilizes APCF to generate recommendations within the pattern subspaces and resorts to an alternative approach for uncovered cases, such as incorporating available background knowledge associated with users and items to enrich the APCF's approach. This strategy could combine the efficiency and local precision of APCF with the global coverage of other methods.

Other avenues for research include exploring other autoencoder architectures for pattern mining as new techniques become available, and conducting a deeper qualitative analysis of the recommendations generated for healthcare applications, with validation from specialists to assess the clinical relevance of the patterns and predictions.

In summary, this thesis has successfully demonstrated a novel and effective approach to leveraging the power of autoencoders in addressing persistent challenges in recommender systems. By shifting the focus from latent representation to pattern mining, the APCF method provides a scalable and robust alternative, with significant potential for systems handling large volumes of data and localized user interest patterns, thereby representing a valuable contribution to the field.

Bibliography

ALSHBANAT, H. I.; BENHIDOUR, H.; KERRACHE, S. A survey of latent factor models in recommender systems. *Information Fusion*, v. 117, p. 102905, 2025. ISSN 1566-2535. Available at: <https://www.sciencedirect.com/science/article/pii/S1566253524006833>. Cited on page 29.

BANK, D.; KOENIGSTEIN, N.; GIRYES, R. *Autoencoders*. 2021. Available at: <https://arxiv.org/abs/2003.05991>. Cited 2 times on pages 38 and 39.

BEHERA, G.; NAIN, N. Trade-off between memory and model-based collaborative filtering recommender system. In: DUA, M.; JAIN, A. K.; YADAV, A.; KUMAR, N.; SIARRY, P. (Ed.). *Proceedings of the International Conference on Paradigms of Communication, Computing and Data Sciences*. Springer, 2022. p. 137–146. ISBN 978-981-16-5747-4. Available at: <https://doi.org/10.1007/978-981-16-5747-4_12>. Cited 2 times on pages 16 and 28.

BENNETT, J.; LANNING, S. *The Netflix Prize*. 2007. Retrieved from <https://api.semanticscholar.org/CorpusID:9528522>. Accessed on November 28, 2024. Cited on page 28.

BREESE, J. S.; HECKERMAN, D.; KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. (UAI'98), p. 43–52. ISBN 155860555X. Available at: <https://doi.org/10.48550/arXiv.1301.7363>. Cited on page 32.

BURKE, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, Springer, v. 12, p. 331–370, 2002. Available at: <https://doi.org/10.1023/A:1021240730564>. Cited 2 times on pages 29 and 36.

CAI, H.; GAN, C.; WANG, T.; ZHANG, Z.; HAN, S. *Once-for-All: Train One Network and Specialize it for Efficient Deployment*. arXiv, 2019. Available at: <https://arxiv.org/abs/1908.09791>. Cited on page 20.

CRUZ-JENTOFT, A. J.; BAHAT, G.; BAUER, J.; BOIRIE, Y.; BRUYèRE, O.; CEDERHOLM, T.; COOPER, C.; LANDI, F.; ROLLAND, Y.; SAYER, A. A.; SCHNEIDER, S. M.; SIEBER, C. C.; TOPINKOVA, E.; VANDEWOUDE, M.; VISSER, M.; ZAMBONI, M.; Writing Group for the European Working Group on Sarcopenia in Older People 2 (EWGSOP2); Extended Group for EWGSOP2. Sarcopenia: revised european consensus on definition and diagnosis. *Age and Ageing*, v. 48, n. 1, p. 16–31, 09 2018. ISSN 0002-0729. Available at: <https://doi.org/10.1093/ageing/afy169>. Cited 2 times on pages 63 and 64.

DESROSIERS, C.; KARYPIS, G. A comprehensive survey of neighborhood-based recommendation methods. In: RICCI, F.; ROKACH, L.; SHAPIRA, B.; KANTOR, P. B. (Ed.). *Recommender Systems Handbook*. Boston, MA: Springer US, 2011. p. 107–144. ISBN 978-0-387-85820-3. Available at: <https://doi.org/10.1007/978-0-387-85820-3_4>. Cited on page 35.

FISCHER, J.; VREEKEN, J. Differentiable pattern set mining. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21)*. New York, NY: ACM, 2021. Available at: <https://doi.org/10.1145/3447548.3467348>. Cited 4 times on pages 17, 25, 27, and 51.

FUNK, S. *Netflix Update: Try This at Home*. 2006. Retrieved from <https://sifter.org/~simon/journal/20061211.html>. Accessed on October 1, 2024. Cited 2 times on pages 29 and 30.

GEMAN, S.; BIENENSTOCK, E.; DOURSAT, R. Neural networks and the bias/variance dilemma. *Neural Computation*, v. 4, n. 1, p. 1–58, 1992. Available at: <https://doi.org/10.1162/neco.1992.4.1.1>. Cited on page 21.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. PMLR, Chia Laguna Resort, Sardinia, Italy, v. 9, p. 249–256, 13–15 May 2010. Available at: <https://proceedings.mlr.press/v9/glorot10a.html>. Cited 2 times on pages 17 and 25.

GOLDBERG, K.; ROEDER, T.; GUPTA, D.; PERKINS, C. Eigentaste: A constant time collaborative filtering algorithm. *Discover Computing*, Springer, v. 4, p. 133–151, 2001. Available at: <https://doi.org/10.1023/A:1011419012209>. Cited on page 51.

Google Machine Learning Education. *Content-based Filtering, Machine Learning*. 2022. Retrieved from <https://developers.google.com/machine-learning/recommendation/content-based/basics>. Accessed on January 16, 2023. Cited on page 16.

HARDESTY, L. *The history of Amazon's recommendation algorithm: Collaborative filtering and beyond.* 2019. Retrieved from <https://www.amazon.science/the-history-of-amazons-recommendation-algorithm>. Accessed on October 2, 2024. Cited 2 times on pages 16 and 28.

HARPER, F. M.; KONSTAN, J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, Association for Computing Machinery, New York, NY, USA, v. 5, n. 4, dec 2015. ISSN 2160-6455. Available at: <https://doi.org/10.1145/2827872>. Cited on page 51.

HERLOCKER, J. L.; KONSTAN, J. A.; TERVEEN, L. G.; RIEDL, J. T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, Association for Computing Machinery, New York, NY, USA, v. 22, n. 1, p. 5–53, jan. 2004. ISSN 1046-8188. Available at: <https://doi.org/10.1145/963770.963772>. Cited on page 33.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multi-layer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989. Available at: <https://doi.org/10.1016/0893-6080(89)90020-8>. Cited on page 19.

HUANG, T.; LIANG, C.; WU, D.; HE, Y. A debiasing autoencoder for recommender system. *IEEE Transactions on Consumer Electronics*, v. 70, n. 1, p. 3603–3613, 2024. Available at: <https://doi.org/10.1109/TCE.2023.3281521>. Cited 4 times on pages 38, 39, 42, and 51.

HUG, N. Surprise: A python library for recommender systems. *Journal of Open Source Software*, The Open Journal, v. 5, n. 52, p. 2174, 2020. Available at: <https://doi.org/10.21105/joss.02174>. Cited on page 51.

ILIC, A.; KABILJO, M. *Recommending items to more than a billion people*. 2015. Retrieved from <https://engineering.fb.com/2015/06/02/core-infra/recommending-items-to-more-than-a-billion-people>. Accessed on October 1, 2024. Cited 2 times on pages 16 and 28.

KINGMA, D. P.; WELLING, M. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, v. 12, n. 4, p. 307–392, 2019. ISSN 1935-8237. Available at: <http://dx.doi.org/10.1561/2200000056>. Cited on page 38.

KOREN, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: [s.n.], 2008. (KDD '08), p. 426–434. ISBN 9781605581934. Available at: <https://doi.org/10.1145/1401890.1401944>. Cited 4 times on pages 29, 30, 31, and 39.

KOTLER, P.; ARMSTRONG, G. *Principles of marketing*. [S.l.]: Pearson Edu, 2010. Cited on page 36.

KRAMER, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, v. 37, n. 2, p. 233–243, 1991. Available at: <https://doi.org/10.1002/aic.690370209>. Cited 2 times on pages 17 and 24.

KUMAR, A.; SHARMA, A. Alleviating sparsity and scalability issues in collaborative filtering based recommender systems. In: SATAPATHY, S. C.; UDGATA, S. K.; BISWAL, B. N. (Ed.). *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA)*. Springer: [s.n.], 2013. p. 103–112. ISBN 978-3-642-35314-7. Available at: <https://doi.org/10.1007/978-3-642-35314-7_13>. Cited 2 times on pages 17 and 36.

LAMPROPOULOS, A. S.; TSIHRINTZIS, G. A. A survey of approaches to designing recommender systems. In: TSIHRINTZIS, G. A.; VIRVOU, M.; JAIN, L. C. (Ed.). *Multimedia Services in Intelligent Environments: Advances in Recommender Systems*. Heidelberg: Springer International Publishing, 2013. p. 7–30. ISBN 978-3-319-00372-6. Available at: <https://doi.org/10.1007/978-3-319-00372-6_2>. Cited 3 times on pages 28, 33, and 36.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998. Available at: <https://doi.org/10.1109/5.726791>. Cited on page 20.

LI, G.; MA, Q.; TANG, H.; PATERSON, A. H.; XU, Y. QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic Acids Research*, v. 37, n. 15, p. e101, 2009. Available at: <https://doi.org/10.1093/nar/gkp491>. Cited on page 43.

LI, S.; KAWALE, J.; FU, Y. Deep collaborative filtering via marginalized denoising auto-encoder. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. New York, NY, USA: Association for

Computing Machinery, 2015. (CIKM '15), p. 811–820. ISBN 9781450337946. Available at: <https://doi.org/10.1145/2806416.2806527>. Cited on page 39.

LI, X.; SHE, J. Collaborative variational autoencoder for recommender systems. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2017. (KDD '17), p. 305–314. ISBN 9781450348874. Available at: <https://doi.org/10.1145/3097983.3098077>. Cited on page 40.

LIANG, S.; PAN, Z.; LIU, w.; YIN, J.; RIJKE, M. de. A survey on variational autoencoders in recommender systems. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 56, n. 10, jun. 2024. ISSN 0360-0300. Available at: <https://doi.org/10.1145/3663364>. Cited on page 39.

LUO, X.; ZHOU, M.; XIA, Y.; ZHU, Q. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, v. 10, n. 2, p. 1273–1284, 2014. Available at: <https://doi.org/10.1109/TII.2014.2308433>. Cited on page 39.

NIELSEN, M. A. How the backpropagation algorithm works. In: *Neural Networks and Deep Learning*. Determination Press, 2015. Available at: <http://neuralnetworksanddeeplearning.com/chap2.html>. Cited on page 22.

OUYANG, Y.; LIU, W.; RONG, W.; XIONG, Z. Autoencoder-based collaborative filtering. In: LOO, C. K.; YAP, K. S.; WONG, K. W.; JIN, A. T. B.; HUANG, K. (Ed.). *Neural Information Processing*. Cham: Springer International Publishing, 2014. p. 284–291. ISBN 978-3-319-12643-2. Available at: <https://doi.org/10.1007/978-3-319-12643-2_35>. Cited on page 40.

PARK, J.; NAM, K. Group recommender system for store product placement. *Data Mining and Knowledge Discovery*, Springer, v. 33, p. 204–229, 2019. Available at: <https://doi.org/10.1007/s10618-018-0600-z>. Cited on page 36.

RAFAILIDIS, D.; CRESTANI, F. Recommendation with social relationships via deep learning. In: *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2017. (ICTIR '17), p. 151–158. ISBN 9781450344906. Available at: <https://doi.org/10.1145/3121050.3121057>. Cited on page 39.

RESNICK, P.; IACOVOU, N.; SUCHAK, M.; BERGSTROM, P.; RIEDL, J. Grouplens: an open architecture for collaborative filtering of netnews. In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 1994. (CSCW '94), p. 175–186. ISBN 0897916891. Available at: <https://doi.org/10.1145/192844.192905>. Cited on page 32.

SALAKHUTDINOV, R.; MNIH, A. Probabilistic matrix factorization. In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2007. (NIPS'07), p. 1257–1264. ISBN 9781605603520. Cited on page 39.

SARWAR, B.; KARYPIS, G.; KONSTAN, J.; RIEDL, J. Analysis of recommendation algorithms for e-commerce. In: *Proceedings of the 2nd ACM Conference on Electronic*

Commerce. New York, NY, USA: Association for Computing Machinery, 2000. (EC '00), p. 158–167. ISBN 1581132727. Available at: <https://doi.org/10.1145/352871.352887>. Cited on page 36.

SARWAR, B.; KARYPIS, G.; KONSTAN, J.; RIEDL, J. Item-based collaborative filtering recommendation algorithms. In: *Proceedings of the 10th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2001. (WWW '01), p. 285–295. ISBN 1581133480. Available at: <https://doi.org/10.1145/371920.372071>. Cited 3 times on pages 16, 18, and 34.

SCHEIN, A. I.; POPESCUL, A.; UNGAR, L. H.; PENNOCK, D. M. Methods and metrics for cold-start recommendations. In: *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2002. (SIGIR '02), p. 253–260. ISBN 1581135610. Available at: <https://doi.org/10.1145/564376.564421>. Cited on page 35.

SEDHAIN, S.; MENON, A. K.; SANNER, S.; XIE, L. Autorec: Autoencoders meet collaborative filtering. In: *Proceedings of the 24th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2015. (WWW '15 Companion), p. 111–112. ISBN 9781450334730. Available at: <https://doi.org/10.1145/2740908.2742726>. Cited on page 41.

SHANI, G.; GUNAWARDANA, A. Evaluating recommendation systems. In: RICCI, F.; ROKACH, L.; SHAPIRA, B.; KANTOR, P. B. (Ed.). *Recommender Systems Handbook*. Boston, MA: Springer US, 2011. p. 257–297. ISBN 978-0-387-85820-3. Available at: <https://doi.org/10.1007/978-0-387-85820-3_8>. Cited on page 49.

SIDI. *IARA, an AI-focused supercomputer*. 2024. Retrieved from <https://www.sidi.org.br/en/about-us>. Accessed on February 4, 2025. Cited on page 51.

SILVA, M. G.; HENRIQUES, R.; MADEIRA, S. C. *User-Specific Bicluster-based Collaborative Filtering: Handling Preference Locality, Sparsity and Subjectivity*. 2022. Available at: <https://doi.org/10.48550/arXiv.2211.08366>. Cited 2 times on pages 44 and 51.

SINGH, M.; MEHROTRA, M. Impact of biclustering on the performance of biclustering based collaborative filtering. *Expert Systems with Applications*, v. 113, p. 443–456, 2018. ISSN 0957-4174. Available at: <https://doi.org/10.1016/j.eswa.2018.06.001>. Cited 2 times on pages 17 and 43.

STRUB, F.; MARY, J.; GAUDEL, R. Hybrid collaborative filtering with autoencoders. 2016. Available at: <https://doi.org/10.48550/arXiv.1603.00806>. Cited 3 times on pages 16, 29, and 41.

SYMEONIDIS, P.; NANOPOULOS, A.; PAPADOPOULOS, A.; MANOLOPOULOS, Y. Nearest-biclusters collaborative filtering with constant values. In: NASRAOUI, O.; SPILIOPOULOU, M.; SRIVASTAVA, J.; MOBASHER, B.; MASAND, B. (Ed.). *Advances in Web Mining and Web Usage Analysis*. Springer: [s.n.], 2007. p. 36–55. ISBN 978-3-540-77485-3. Available at: <https://doi.org/10.1007/978-3-540-77485-3_3>. Cited on page 46.

TONG, H.; YANG, Z.; WANG, S.; HU, Y.; SEMIARI, O.; SAAD, W.; YIN, C. Federated learning for audio semantic communication. *Frontiers in Communications and Networks*, v. 2, 2021. ISSN 2673-530X. Available at: <https://www.frontiersin.org/journals/communications-and-networks/articles/10.3389/frcmn.2021.734402>. Cited on page 25.

VALDEZ, A. C.; ZIEFLE, M.; VERBERT, K.; FELFERNIG, A.; HOLZINGER, A. Recommender systems for health informatics: State-of-the-art and future perspectives. In: HOLZINGER, A. (Ed.). *Machine Learning for Health Informatics: State-of-the-Art and Future Challenges*. Cham: Springer International Publishing, 2016. p. 391–414. ISBN 978-3-319-50478-0. Available at: <https://doi.org/10.1007/978-3-319-50478-0_20>. Cited on page 16.

VERONEZE, R.; VON ZUBEN, F. J. Scalability achievements for enumerative biclustering with online partitioning: Case studies involving mixed-attribute datasets. *Engineering Applications of Artificial Intelligence*, v. 100, p. 104147, 2021. ISSN 0952-1976. Available at: <https://doi.org/10.1016/j.engappai.2020.104147>. Cited on page 45.

VINCENT, P.; LAROCHELLE, H.; BENGIO, Y.; MANZAGOL, P.-A. Extracting and composing robust features with denoising autoencoders. In: *Proceedings of the 25th International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2008. (ICML '08), p. 1096–1103. ISBN 9781605582054. Available at: <https://doi.org/10.1145/1390156.1390294>. Cited on page 38.

WANG, Y.; WANG, L.; LI, Y.; HE, D.; LIU, T.-Y. A theoretical analysis of NDCG type ranking measures. In: *JMLR: Workshop and Conference Proceedings*. [s.n.], 2013. p. 1–30. Available at: <https://doi.org/10.48550/arXiv.1304.6480>. Cited on page 50.

WU, Y.; DUBOIS, C.; ZHENG, A. X.; ESTER, M. Collaborative denoising auto-encoders for top-n recommender systems. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2016. (WSDM '16), p. 153–162. ISBN 9781450337168. Available at: <https://doi.org/10.1145/2835776.2835837>. Cited on page 41.

ZHANG, F.; YUAN, N. J.; LIAN, D.; XIE, X.; MA, W.-Y. Collaborative knowledge base embedding for recommender systems. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2016. (KDD '16), p. 353–362. ISBN 9781450342322. Available at: <https://doi.org/10.1145/2939672.2939673>. Cited on page 40.

ZHANG, G.; LIU, Y.; JIN, X. A survey of autoencoder-based recommender systems. *Frontiers of Computer Science*, v. 14, p. 430–450, 2020. Available at: <https://doi.org/10.1007/s11704-018-8052-6>. Cited 3 times on pages 17, 25, and 38.

ZHANG, S.; YAO, L.; XU, X. AutoSVD++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2017. (SIGIR '17), p. 957–960. ISBN 9781450350228. Available at: <https://doi.org/10.1145/3077136.3080689>. Cited on page 40.