Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Exploring Machine Learning Techniques in Collaborative Filtering Recommender Systems

Diane Chounlamountry

A research paper submitted to the University of Dublin, in partial fulfilment of the requirements for the degree of Master of Science Interactive Digital Media

Supervised by Radek Przedpelski

May 12th 2025

Declaration

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at: http://www.tcd.ie/calendar

I have also completed the Online Tutorial on avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

I declare that the work described in this research Paper is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: [illegible]

Diane Chounlamountry

12/05/2025

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this research Paper upon request.

Signed: [illegible]

Diane Chounlamountry

12/05/2025

Acknowlegments

I would like to thank my supervisor Radek Przedpelski for his guidance, helpful advice, and continuous reassurance throughout the writing of this research paper.

Abstract

Recommender systems (RS) refer to software tools or algorithms that suggest relevant items to users based on their behaviour. They are used in various domains such as e-commerce, education, or social media, and have proven effective in personalizing and improving user experiences on digital platforms in the era of big data and information overload. This paper focuses on collaborative filtering (CF), one of the most widely used techniques in RS, which generates predictions by analysing user-item interactions. More specifically, this paper examines model-based CF – a category of CF that applies machine learning (ML) techniques to discover latent structures in the data and train predictive models. The study presents fundamental ML algorithms used in CF, such as matrix factorization and clustering techniques. Furthermore, some of these methods are implemented and evaluated using MovieLens 1M – a widely used public dataset containing one million anonymous movie ratings in a 1-5 star format. These experiments aim to explore and compare different ML-based approaches, highlighting persistent challenges in model-based CF such as data sparsity and scalability. While the paper includes theoretical descriptions of the algorithms, it remains beginner-friendly and aims to serve as an accessible introduction for readers interested in collaborative filtering recommender systems CF-RS.

Table of contents

1 Introduction

The growing amount of information available to users online makes it harder to find content that aligns with individual interest. Recommender systems (RS) have addressed this issue by suggesting items and services that are tailored to the user's preferences [1]. Instead of browsing through the entire catalogue of a shopping website, a streaming platform, or educational resources, RS filter the items to only present to the users some that they are likely to engage with. To measure the relevance of these suggestions, recommender systems analyse a variety of user data. These data sources can be explicit – such as user ratings, review or social connections – or implicit – such as the history of websites visited, time spent on a platform, number of clicks. The ultimate goal of this data collection is to enable the system to recommend items that reflect the user's preferences. By doing so, the user engagement and satisfaction are likely to increase which is beneficial for the digital platforms in a time where competitors are often one click away. This marketing benefit is one of the main reasons recommender systems are integral to so many industries today. For instance, in e-commerce, Amazon [2] suggests items based on browsing history and previous purchase. Similarly, streaming platforms like Netflix [3] and Spotify [4] rely on RS to recommend films, series, songs, or podcasts suited to the user's tastes. Social media networks also use RS algorithms to suggest specific content or friend connections. From a computational perspective, two main objectives of RS are usually distinguished: rating prediction and top-N recommendation. The first aims to predict how a user would rate an item they have not interacted with yet, while the second focuses on generating a list – sometimes ranked – of items that the user should enjoy. The top-N recommendation objective emerged as it appears users were more interested with being presented content they will like, rather than knowing whether they would like highly or poorly a specific item. However, both categories are relevant depending on the task and the platform.

There are two main types of RS: content-based filtering and collaborative filtering. Content-based filtering focuses on the characteristics of the items to recommend ones that are similar, content-wise, to those the user has enjoyed before. Therefore, the method relies on a description of the item in terms of, for example, keywords, genre, or physical attributes. Collaborative filtering (CF) [5][6] relies on relationships between users and identifies patterns among users with similar taste. It mirrors real-life decision-making where people trust the suggestions of other like-minded individuals. For example, if user A and user B have similar tastes, items liked by user B may be recommended to user A. Simply, CF algorithms rely on the principle that similar users are interested in similar items.

Collaborative filtering offers a way to look at items beyond pure content as it takes into consideration human taste. CF algorithms can further be sorted into two categories: memory-based and model-based approaches. Memory-based methods rely on direct computation of similarities between users or items using historical data, while model-based methods apply machine learning (ML) techniques to learn user-item interaction patterns and make predictions. ML has played a crucial role in modern RS with the rise of big data by enhancing their performance, particularly in the context of scalability.

Despite the progress, RS continues to face challenges such as the cold-start problem, data sparsity and scalability. The cold-start problem refers to a situation where the system has insufficient data about new users or new items [7]. Data sparsity occurs when users have rated or interacted with only a small subset of items, which prevents the system to identify meaningful patterns. Finally, the scalability issue has become more prominent with the rise of large-scale datasets and pushes for optimized algorithms.

The aim of this paper is to provide an introduction to collaborative filtering recommender systems (CF-RS), with a focus on the role of machine learning algorithms. A few algorithms are implemented using the MovieLens 1M dataset [8] to explore how each model performs across multiple evaluation metrics. This research paper covers the basics of CF-RS which makes it relevant to any reader interested in that subject. The remainder of this paper is structured as follows: In Section 2, a literature review provides the key principles of CF-RS and ML. Section 3 details the methodology undertaken to do the experiment. Section 4 presents the experimental process, with a comparison and analysis of different algorithms using various metrics. Section 5 discusses broader perspectives before the conclusion summarizes the contribution and findings of this paper, in Section 6.

2 Literature review

2.1 Basics of Recommender Systems

Recommender systems are designed to help users discover relevant content in environments where the volume of available options is overwhelming. Whether in e-commerce, streaming services, or social media, digital platforms have become spaces where users are exposed to so much information that it can be difficult to find what they like on their own. This phenomenon of information overload has made personalization not only beneficial but essential. Recommender systems have become useful tools to automatically filter and select specific items for users. Almost every industry now uses a form of RS, but it is especially prominent in e-commerce and the entertainment industry, where it increases sales. By helping users find what they are most likely to enjoy, RSs boost engagement and satisfaction, which in turn benefits businesses. This is clearly shown by the success of Amazon and its large-scale recommendation system [2].

Recommender systems can be described as the association of three entities: users, items and recommendation algorithms [9]. A user is anyone who uses the system to receive information or provide ratings on items [6]. Symmetrically, items refer to anything that can be rated – books, articles, movies, songs, applications, and so on. Ratings can be binary ("agree/disagree") or scalar; this paper focuses on the scalar MovieLens ratings of 1-5 stars. A ratings matrix is a table where each row represents a user, each column represents an items and the value at the intersection is the user's rating of that item (Table 1). Ratings matrices are usually very sparse, meaning many values are missing, because users only rate a small percentage of available items.

	Movie 1	**Movie 2**	**Movie 3**	**Movie 4**	**Movie 5**	**Movie 6**
User 1	3	...	...	4	...	3
User 2	...	1	3	3	...	...
User 3	2	2	3	...	...	5

Table 1: Example of a user-item rating matrix

Recommender systems are categorized based on the type of algorithm they use. The main categories are collaborative filtering (CF), content-based filtering, and hybrid filtering.

Content-based filtering relies on the assumption that users will prefer items similar to those they liked in the past. Each item is described by a set of attributes, and the system recommends new items that share features with those the user previously rated positively. Over time, the system builds and updates a user profile based on these preferences. The main benefits of this approach are that it avoids the cold start problem (for new users) and is not affected by data sparsity. However, it has drawbacks. First, it requires accurate content analysis - which can be difficult, especially for rich media like videos or images. Second, it tends to lead to overspecialization, recommending only very similar items and limiting the user's exposure to new or diverse content [10]. This can reinforce narrow interest loops, a phenomenon clearly visible in social media through the emergence of filter bubbles.

Collaborative filtering (CF) relies on the idea that similar users are likely to enjoy similar items.

It mirrors real-life decision-making, where someone relies on people they trust others for recommendations. For instance, if a colleague with similar tastes enjoys a movie, you might be inclined to watch it. Likewise, if you usually dislike a friend's music choices, you are unlikely to be interested in their new favourite artist. CF applies this logic by analysing user behaviour and identifying patterns of similarity across users. A major strength of CF is that it considers user opinions rather than just item characteristics, making it effective for capturing quality and taste. Breese et al. [5] distinguish between memory-based and model-based approaches to CF, which will be addressed in more detail in the next section.

Hybrid filtering combines elements of content-based and collaborative filtering. The goal is to leverage the strengths of each approach while minimizing their weaknesses. In practice, most modern recommender systems blend different methods. In [11], Burke outlines several types of hybrid strategies.

2.2 Collaborative Filtering (CF)

Collaborative filtering (CF) [12] relies on user similarity to generate recommendations. It is widely used because it does not require explicit content descriptions and is able to handle complex objects such as songs and movies. The main goal is that users do not have to browse through too much information as the relevant content will have already been filtered and selected for them. For that, the RS needs to collect data information for every user; this step can be done explicitly or implicitly (2.2.1). Moreover, there are two types of algorithms within collaborative filtering: *memory-based* and *model-based*, which are addressed in sections 2.2.2 and 2.2.3 respectively.

2.2.1 Data collection

To generate relevant and personalized recommendations, a recommender system must first gather data to build a user profile. This process typically involves two main types of data collection: explicit (also known as active filtering) and implicit (or passive filtering).

a) Explicit (active filtering) data collection
In this case, the user gives explicit information to the system. Commonly, the user is asked to rate an item on a scale of 1 to 5 but explicit data also takes the form of adding to favourites, commenting or liking. The advantage of this method of data collection is that the historical behaviour of a user can easily be traced and reconstructed, without having information of other users interfering with it. It is important to keep in mind that users do necessarily know how to rate an item correctly and the way they attribute a rating may change, without their actual taste changing. This is called declaration bias and RS algorithms need to take it into account to provide adequate suggestions.

b) Implicit (passive filtering) data collection
On the opposite side, implicit data collection does not require to ask anything from the user. Instead, it is based on the observation and analysis of user behaviour. For example, the RS observe the items recently purchased, the songs listened to recently, the movies consulted, etc... The frequency and duration are also informative. All this information is collected automatically without requesting anything from the user which prevails from the previous declaration bias. Nevertheless, a user may not enjoy the things he has recently consulted with create another bias.

2.2.2 Memory-based CF

Memory-based CF [13] relies directly on the user-item rating matrix to identify and compute similarities either between users or between items. These similarities help in predicting future ratings. Memory-based methods are typically neighbourhood-based approaches, meaning the recommendations are generated by examining the preferences or characteristics of close items or users. The term "nearest neighbour" is attributed between two items or two users that have high similarity scores.

There are two main types of memory-based CF that exist: user-based and item-based. It simply depends on whether the similarity is computed between users or items. Both methods follow a similar approach which typically consists of four main steps:

1. Compute similarity (between users or items)
2. Find the nearest neighbours
3. Predict missing ratings
4. Recommend items (based on those predictions)

a) User-based

User-based algorithms operate on the assumption that if users have a similar historical rating, they have similar interest and will continue to have similar preferences in the future. Simply put, if two users have both rated multiple romantic comedies highly, the system better recommend romantic movies liked by one user to the other, if they have not seen it yet.

As mentioned above, the first step in memory-based method is to compute the similarity scores. The similarity between users is based on their *rating vectors*. For an active user u that has rated N items, the rating vector R_u contains the N ratings of that user.

$$R_u = \{r_{ui}\}_{i=1,\dots,N} = \{r_{u1}, \dots, r_{uN}\}$$

The similarity score of user u with a user v is computed with a similarity measure. While other exist, only the *cosine similarity* and the *Pearson correlation coefficient (PCC)* similarity measures are presented here as they are both largely used.

$$\text{sim}_{cos}(u,v) = cos(R_u, R_v) = \frac{R_u \cdot R_v}{||R_u||||R_v||} \tag{1}$$

$$\text{sim}_{\text{PCC}}(u,v) = cos(R_u - \bar{R}_u, R_v - \bar{R}_v) = \frac{(R_u - \bar{R}_u) \cdot (R_v - \bar{R}_v)}{\|R_u - \bar{R}_u\| \, \|R_v - \bar{R}_v\|} \tag{2}$$

Once the similarities have been computed using one of the metrics above, the nearest neighbours of the active user u can be identified. This can be done using a fixed number K and selecting the K users with whom u has the highest similarity scores (top-K nearest neighbours), or by selecting all users with a similarity above a predefined threshold δ.

Let N_u represent the set of selected nearest neighbours. The final step is predicting the ratings; in other words, how would user u rate the item i which they have not interacted with yet. Typically, a weighted average of ratings from the nearest neighbours is done, which brings the following basic prediction formula:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_u} \text{sim}(u,v) \cdot r_{vi}}{\sum_{v \in N_u} |\text{sim}(u,v)|} \tag{3}$$

b) Item-based

Item-based algorithms work similarly as the user-based ones, except the similarity is computed between items instead of users. The underlying consideration is that if a user liked a particular item, they are likely to enjoy other similar items.

Cosine similarity or PCC can be used once again to identify the most similar items to a target item i. The system can estimate how much a user u might like the item i based on how u rated similar items. The prediction formula is analogous to the user-based one, but the roles of users and items are swapped:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_i} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N_i} |\text{sim}(i,j)|} \tag{4}$$

While memory-based CF is intuitive and easy to implement, it shows several limitations when applied to real-world scenarios. Indeed, memory-based CF is heavily dependent on the volume and quality of the rating data and often sparsity. This is a common issue where users have rated only a small subset of available items which lead to a sparse rating matrix and unreliable similarity scores. Additionally, memory-based methods scale poorly with large datasets because computing similarities between all user or item pairs is computationally expensive. In real life, datasets usually consist of millions of samples which is why memory-based may not be the most ideal method.

2.2.3 Model-based CF

Goals and motivation

Model-based CF offers a more scalable and generalizable alternative to memory-based CF. At runtime, memory-based CF needs to compute direct comparisons while model-based techniques use a model that was created beforehand. Indeed, they learn a model from historical data which captures latent structures and patterns in the interactions between users and items. It is first trained on existing data, after which the system can generate predictions and produce recommendations on new data.

Thanks to their efficiency and robustness, model-based approaches have become more popular in recent years. Having a pretrained model enables a quick generation of predictions which is particularly valuable in commercial environments that require real-time recommendations. Additionally, these models are better with sparse data since global trends have been extracted during the training. They are also better at handling large-scale datasets than memory-based techniques which is extremely valuable for modern RS.

Overview of key approaches

Several families of machine learning algorithms are commonly used in model-based CF. A more theoretical and technical discussion is presented in Section 2.4; here, the main categories are briefly introduced.

Matrix factorization: These methods assume that the rating matrix can be approximated by the product of two matrices of lower dimensions. The two matrices represent users and items features in a shared latent space. Matrix factorization is particularly effective for capturing hidden features that can explain observed ratings. This technique was popularized by the Netflix Prize competition. [14]

Clustering-based methods: These group users or items into clusters based on similar characteristics. Once users are assigned to clusters, recommendations are generated within those clusters, which significantly reduces the search space and improves computational performance on large datasets. [15]

Hybrid models: These combine multiple recommendation strategies to achieve better performance. For example, matrix factorization might be used to reduce dimensionality, followed by clustering to improve prediction. Hybrids can also integrate content-based or memory-based techniques to leverage the strengths of each method. [11]

Trade-offs and challenges

Despite their many advantages, model-based approaches are not without drawbacks. Indeed, while they offer good scalability at runtime, the initial training phase of the models can be quite computationally intensive and time-consuming, especially on large datasets. Furthermore, the performance of these models often hinges on the choice of hyperparameters values - hyperparameter tuning is a sensitive phase that requires careful experimentation.

2.3 Basics of Machine Learning (ML)

2.3.1 Definitions

Machine learning (ML) is a subset of artificial intelligence that enables systems to make predictions and decisions without being explicitly programmed for each task [16] . It relies on building a model that learns patterns from data and that can generalize it after on unseen data. Three main categories of ML algorithms can be outlined, depending on the nature of the task and type of data available: supervised, unsupervised and semi-supervised learning [17]. Supervised learning involves training a model on a labelled dataset where each input is paired with a known target output. Predicting a user's rating for a movie based on historical ratings is typically classified as supervised learning. In contrast, unsupervised algorithms deal with unlabelled data and focus on identifying underlying patterns within the data; clustering users with similar preferences is one example as there are no predefined clusters given to the model. Finally, semi-supervised learning works with both labelled and unlabelled data. Usually, it uses a small amount of labelled data and a large amount of unlabelled data to improve the efficiency of the learning.

Another distinction made between ML models is in whether they are generative or discriminative [18]. Generative models aim to learn how the given data is generated to be able to generate new data points, such as matrix factorization where the goal is predicting missing ratings. Discriminative models focus on learning decision boundaries such as clustering-based methods. They are not trying to generate missing data, but rather analyse the existing data to learn how to distinguish different classes. For RS, knowing these distinctions is helpful when choosing which algorithms would be best suited for a particular recommendation task.

2.3.2 ML in practice

In practice, building a correct ML model comes with following several key implementation concepts including data splitting, evaluation, and tuning.

Typically, datasets are split into a training set and a test set – usually in 80%-20% or 75%-25% ratios. The training set is used to train the model and learn the data, while the test set is kept for the evaluation part: "how well does the model perform on unseen data?". Splitting the dataset into distinct subsets is essential to assess the generalization of the model. This splitting is usually improved with cross-validation techniques [19] to ensure even more reliability and robustness in the model. The most common technique is the K-fold cross-validation [20] where the dataset is split into k subsets; the model is trained k times, each time using a different subset as the test set. Cross-validation helps minimizing the risk of overfitting which refers to a model that performs well on the training data but poorly on the test data. This occurs when the model too specialized to the training data, it was overly trained and captured the noise present in the train set, rather than meaningful patterns. A major challenge in ML is finding the balance between overfitting and underfitting. The latter refers to a model that is too simple and fails to capture underlying patterns.

Finally, hyperparameter tuning is a critical step in the ML workflow, where the goal is to identify the most effective set of hyperparameters to instantiate a model [21]. There is a subtle distinction to be made between parameters, which are internal variables learned during training (e.g. cluster centroids in KMeans), and hyperparameters, which are set before the training and control the learning process (e.g. number of clusters in KMeans). Choosing appropriate hyperparameters is essential to achieve optimal performances, but it can be complicated due to the lack of explicit guidelines to find the right values. Tools like grid search and random search are commonly used to help in this task; given a range of hyperparameters value, they evaluate different combinations to output the best-performing one.

2.4 Theoretical grounds of the ML techniques in model-based CF

This section explores different machine learning techniques used in CF-RS to improve the performance of the system. Existing methods have been reviewed, compared and categorized in prior studies. For example, Su et al. [13] investigated various models and grouped them depending on the algorithm type, including Bayesian methods, clustering, regression-based models, Markov decision process (MDP), latent semantic CF models and others. In [22], Lü et al. discussed methods such as dimensionality reduction, diffusion-based, social filtering, and meta approaches. More recently, Shafiee reviewed the latest machine learning trends in RS and outlined research perspectives and opportunities across multiple domains such as education, health, e-commerce and digital journalism [23].

This current paper does not intend to provide an exhaustive review of all ML techniques used in model-based CF. Instead, it focuses on a set of common techniques that are part of these categories: dimensionality reduction approaches, clustering-based approaches, and hybrid models.

2.4.1 Dimensionality reduction approaches

Dimensionality reduction techniques are used to improve the scalability of recommender systems. In their case study on the MovieLens dataset, Sarwar et al. [24] demonstrated how the complexity of large-scale data can be reduced by applying such techniques. The user-item interaction space in collaborative filtering is typically high-dimensional, and the objective of dimensionality reduction is to transform this space into a lower-dimensional latent space that still captures the key structure and relationships within the data. Since the rating matrix is often extremely sparse, generating a compact representation helps manage sparsity and reduce noise, which improves the model's ability to generalize.

This section introduces three techniques: Matrix Factorization (MF), Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF). While SVD and NMF use a form of matrix factorization, they are presented on their own as they are two popular methods with distinct characteristics and optimization strategies.

a) Matrix Factorization (MF)

Matrix Factorization (MF) is a foundational dimensionality reduction technique used in model-based CF. In their well-known paper, Koren et al. [14] provide a detailed exploration of MF techniques in RS and emphasize their effectiveness, notably demonstrated during the Netflix Prize competition in 2006. In [25], Mehta et al. give a review of different matrix factorization techniques used in RS, while Xue et al. [26] propose a novel MF model based on neural network architecture. The remainder of this section presents the principle of MF, followed by a description of two commonly used optimization algorithms: Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS).

MF assumes that there exists a set of latent features that can adequately describe both users and items. The rating matrix – typically large and sparse – is factorized into two low-ranked matrices that each hold information about the users or the items. Let $R \in \mathbb{R}^{m \times n}$ be the user-item rating matrix, where m is the number of users and n the number of items. MF approximates this matrix as:

$$R \sim \hat{R} = PQ^T \tag{5}$$

Where:

- $P \in \mathbb{R}^{m \times k}$ is the user-feature matrix
- $Q \in \mathbb{R}^{n \times k}$ is the item-feature matrix
- $k << \min(m, n)$ is the number of latent features

The predicted rating for user u and item i is given by the dot product of the two feature vectors 6.

$$\hat{r}_{ui} = p_u^T q_i = \sum_{f=1}^{k} p_{uf} q_{if} \tag{6}$$

The model is trained by minimizing the reconstruction error, meaning minimizing the difference between R and $\hat{R}$. This error between the observed ratings r_{ui} and their predictions $\hat{r}_{ui}$ is typically obtained by computing the squared loss over known ratings:

$$\min_{P,Q} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - p_u^T q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2) \tag{7}$$

Where $\mathcal{K}$ is the set of known (non-missing) ratings, and $\lambda > 0$ is a regularization parameter to prevent overfitting.

Optimization Algorithms

To find the optimal matrices P and Q that minimize the reconstruction error, two popular optimization strategies are commonly used: stochastic gradient descent (SGD) and alternating least squares (ALS). The general framework of these two optimization algorithms is as follow:

1. Randomly initialise the features matrices $P \in \mathbb{R}^{m \times k}$ and $Q \in \mathbb{R}^{n \times k}$
2. For each known rating r_{ui}, compute the prediction error $e_{ui} = r_{ui} \breve{} p_u^T q_i$
3. Update the latent vectors $\{p_u\}$ and $\{q_i\}$ using a rule specific to the algorithm (see below)
4. Repeat [2-3] until convergence or until the maximum number of iterations is reached.

Stochastic Gradient Descent (SGD)

SGD updates the user and item vectors incrementally for each observed pair.

$$\begin{cases} p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u) \\ q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i) \end{cases} \tag{8}$$

Where γ is the learning rate and λ the regularization coefficient.

Alternating Least Squares (ALS)

ALS fixes one matrix and solves for the other using regularized least squares. The algorithm keeps alternating the fixed matrix until convergence.

$$\begin{cases} P \leftarrow \arg\min_P ||R - PQ^T||_F^2 + \lambda||P||_F^2 \\ Q \leftarrow \arg\min_Q ||R - PQ^T||_F^2 + \lambda||Q||_F^2 \end{cases} \tag{9}$$

b) Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a matrix factorization technique in linear algebra that is widely applied in collaborative filtering. It helps uncover the latent relationships between users and items. SVD decomposes the rating matrix $R \in \mathbb{R}^{m \times n}$ into three matrices:

$$R = USV^T \tag{10}$$

The columns of the matrix $U \in \mathbb{R}^{m \times r}$ are called the "left singular vectors" and are the eigenvectors of RR^T. Similarly, $V \in \mathbb{R}^{n \times r}$ contains the "right singular vectors" which are the eigenvectors of $R^T R$. $S \in \mathbb{R}^{r \times r}$ is a diagonal matrix of size r, with r being the rank of the original matrix R. The diagonal entries of S are the singular values σ_i of R, they are ordered with decreasing magnitude and are all non-negative. Additionally, both U and V are orthogonal matrices.

$$\begin{cases} \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r, \quad \text{with } r = \text{rank}(R) \\ UU^T = U^T U = I_n \\ VV^T = V^T V = I_m \end{cases} \tag{11}$$

The first columns of the matrix can be interpreted as the key features that are the most important to keep. Precisely, the stake of SVD is to produce a low-rank approximation of the original matrix. Therefore, by retaining only the top k singular values (with $k < r$), the best rank-k approximation of R in terms of the Frobenius norm (the matrix equivalent of the Euclidean norm) is obtained:

$$R_k = U_k S_k V_k^T \tag{12}$$

Where U_k, S_k, and V_k are the truncated matrices of U, S and V, where only the first k columns were kept. This approximation means that out of all matrices of rank k, R_k is the matrix that minimizes the Frobenius norm $||R - R_k||_F$. In other words:

$$\min_{\text{rank}(X)=k} \|R - X\|_F = \|R - R_k\|_F \tag{13}$$

Overall, SVD allows to capture the (k) most influential latent features driving user preferences and item characteristics. Each row of U_k can be interpreted as a user embedding, and each row of V_k as an item embedding, in a shared latent feature space of dimension k.

While the scalability of SVD makes it a good option for large datasets, the major limitation of applying classical SVD in recommender systems is the sparsity of rating matrices. As SVD requires a full matrix, early implementations such as [24] relied on filling the missing ratings with zeros or with user/items averages. The risk of such method is to distort the data and underlying patterns and introduce bias in the latent features. To address this, researchers have developed incremental SVD techniques that can dynamically update the decomposition as new ratings arrive, without needing to recompute the entire factorization. Such incremental algorithms were proposed by Sarwar et al. [27] and Brand [28]. More recently, Zhou et al. proposed a combination of incremental and approximating SVD ("Incremental ApproSVD") [29].

In practice, libraries like Surprise in Python implement "Funk-SVD" which is a version of SVD popularized during the Netflix Prize by Simon Funk. Unlike classical SVD, this method does not perform an explicit matrix decomposition. Instead, it learns the latent user and item vectors directly from the observed ratings through stochastic gradient descent. This technique allows the application of an SVD-style factorization without needing to impute missing values which makes it particularly useful for sparse datasets.

c) Non-negative Matrix Factorization (NMF)

Non-negative matrix factorization approximates the rating matrix R like traditional matrix factorization 5, but requires P and Q to be non-negative: $P >= 0$ and $Q > 0$.

Adding this constraint makes NMF more interpretable than other methods, especially in domains where negative interactions are not meaningful. Each user and item are represented by a non-negative combination of latent factors, which can be viewed as additive contributions. This method has proven effective as illustrated by the early work of Lee and Seung [30], and further explored in the collaborative filtering context by Luo et al. [31]. A potential downside of NMF, however, is its reduced flexibility compared to unconstrained models. Indeed, the strict non-negativity can sometimes limit the predictive accuracy of the model.

2.4.2 Clustering-based approaches

To improve the scalability and interpretability of recommender systems, clustering-based approaches group similar users and items into clusters [32], [33], [15], [34]. Instead of comparing every user together, these methods first identify communities that have similar behaviours. The recommendations are then generated within the clusters which reduces the size of the problem while still capturing the main information. Two commonly used clustering techniques in RS are K-means and hierarchical clustering.

a) K-Means clustering
K-means is a popular unsupervised learning algorithm that partitions data into k disjoint clusters based on similarity. In collaborative filtering, a common user-based approach is to cluster users based on their rating patterns. Each user is represented by a vector which contains their ratings, with any missing rating filled with the user's mean rating. These vectors reflect each user's preference profile and are used by K-means to group similar users together. The algorithm tries to minimize the intra-cluster variance by solving the following optimization problem:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} ||x_i - \mu_j||^2 \tag{14}$$

Here, $x_i \in \mathbb{R}^n$ is the user vector, C_j is the set of users assigned to cluster j, and μ_j is the centroid of that cluster. The K-means algorithm proceeeds iteratively:

1. Randomly initialize k centroids (centres of the clusters)
2. Compute the distance of every user to each centroid (Euclidean distance)
3. Assign each user to the nearest centroid
4. Recompute centroids as the mean of all users assigned to each cluster
5. Repeat steps 2-4 until convergence (the clusters do not change) or until a maximum number of iterations is reached.

Once clustering is complete, recommendations can be made within the clusters. The general idea is to leverage the preferences of other users in the same cluster to make recommendations for a target user. For example, to recommend new items to a user u who belongs to cluster C_j, the recommendation process can be outlined as follows. First, the items that user u has not rated yet need to be identified. If item i is one of these unrated items, its estimated rating is computed as the average rating across other users of cluster C_j who have rated i.

$$\hat{r}_{ui} = \frac{1}{|C'_j| \sum_{v \in C'_j, v \neq u} R_{vi}} \tag{15}$$

Where $C'_j \subseteq C_j$ includes all users in the cluster who have rated item i.

Then, all the unrated items are sorted by descending order of their estimated ratings $\hat{r}_{ui}$. The N items with the highest estimated ratings constitute the top-N recommendation list for user u. To improve the algorithm, it is also possible to weight the users in the cluster based on their similarity to user u instead of treating them equally.

K-means is attractive because it is scalable, relatively simple and provides interpretable groupings. The choice of k, the number of clusters, has a direct impact on the results and must be chosen carefully using heuristics or validation metrics. Moreover, the algorithm is sensitive to the initial centroids – which are selected randomly – and can sometimes converge to a local minimum. Various strategies have been developed to improve this issue, such as GA K-means, which uses genetic algorithms to optimize the initial centroid placement [35].

b) Hierarchical clustering

Hierarchical clustering is another unsupervised learning method used to group users or items together. The main difference with K-means is that hierarchical clustering does not require a predefined number of clusters. Indeed, this algorithm builds a "nested hierarchy" of clusters that is usually represented with a tree structure called a "dendrogram" [36]. There are two main strategies to build this tree: agglomerative (from the bottom up) and divisive (from the top down). Recommender systems more commonly use the agglomerative method; it starts with each user as an individual cluster and iteratively merges clusters together. More precisely, using a distance metric (e.g., Euclidean distance) and a linkage criterion (such as single, complete, or average linkage), the two most similar clusters are selected to be merged. This continues until all users are grouped into a single hierarchical cluster.

To make recommendations with hierarchical clustering, a cut is made in the dendrogram to form a fixed number of clusters. Then, the process is similar than the K-means one: recommendations are generated using the average ratings of users within the same cluster. The main advantage of hierarchical clustering is that the number of clusters does not need to be specified in advance. It also provides a more visual representation of user similarity which can be useful. However, this method tends to be more computationally expensive and less scalable than K-means, which limits its practicality for large datasets. In [15], Kohrs et al. present an algorithm based on hierarchical clustering.

2.4.3 Hybrid approaches

Many recommender systems use hybrid approaches rather than one specific technique to improve their performance. The combination of multiple methods aims to leverage the strengths of each and minimize their weaknesses. For instance, the various techniques explored in the two previous sections are powerful on their own but can be applied together to create more robust models. Applying the dimensionality reduction techniques before running the clustering algorithms is a common strategy. Matrix factorization pre-processes the user-item rating matrix before the clustering; applying SVD or NMF to the high-dimensional and sparse matrix creates a more compact representation. The clustering process in this denser space of lower dimension makes the groupings of users and items more meaningful, which improves the quality of the recommendations.

For example, Li and Kim [34] proposed a hybrid clustering approach that improved prediction accuracy, particularly for the cold start problem. In [37], Koren et al. introduce a system that combines matrix factorization with neighbourhood-based methods. This hybrid model was used successfully in the Netflix Prize competition and became a cornerstone in the field of RS [14].

Beyond these examples, there are many other ways to combine recommendation techniques. Burke offers a well-known survey of hybrid recommender systems [11] that outlines different design approaches such as weighted, switching, or cascade models.

2.5 Evaluation metrics

RS is used in many different industries and can have different purposes depending on the types of information and items it needs to analyse. Some algorithms are therefore more effective than others in certain domains or perform differently according to the input (for example, the percentage of users over items). Choosing one algorithm instead of another is a crucial part of the recommendation system; to make the right decision, it is necessary to use evaluation measures that are coherent with the said algorithm. Researchers have classified evaluation measures into various categories. The algorithms can be evaluated at any point in time, meaning it is conceivable to evaluate intermediate steps such as the rapidity of the nearest neighbours classification, but since the most important part of a RS is the output, researchers mostly focus on measures evaluating the results of the algorithm rather than intermediate steps. When RS first appeared, most of the research was focused on improving the accuracy of the model. As the field evolved, many more evaluation measures have been introduced to consider more diverse aspects of the quality of a RS. This section presents the most common evaluation measures starting with accuracy metrics and then continuing with new kind of considerations.

2.5.1 Accuracy metrics

Accuracy metrics measure how close the prediction value is to the truth value. They are the most prominent type of metrics in RS and have been used since the beginning of the field and in many other subsets of machine learning. Evaluating an algorithm by computing the accuracy of the result is a straightforward approach and is easy to understand and to implement. Accuracy metrics can also be divided into three categories depending on the purpose of the RS: the predictive accuracy metrics, the classification accuracy metrics and the ranking accuracy metrics.

a) Predictions

Predictive accuracy metrics are relevant for systems that aim to display the predicted score to the user. For example, if a movie streaming platform gives a score to a movie so that the user knows how likely they are to enjoy it. These metrics measure the prediction error, in other words, the difference between the predicted ratings and the actual user ratings. The standardized metric used to evaluate the predicted score accuracy is the Mean Average Error (MAE). This metric measures the average absolute deviation between the true rating r_{ui} and its prediction $\hat{r}_{ui}$.

$$MAE = \frac{1}{N} \sum_{u,i} |r_{u,i} - \hat{r}_{u,i}| \tag{16}$$

Many other metrics derive from the MAE, such as root mean squared error (RMSE) and normalized mean absolute error (NMAE) [38]. RMSE is useful to give more importance to large errors because it computes the square of the error before the summation; while a 1-point error would increase the total sum of one point, a 2-point error increases it by four.

$$RMSE = \sqrt{\frac{1}{N} \sum_{u,i} (r_{u,i} - \hat{r}_{u,i})^2} \tag{17}$$

$$NMAE = \frac{MAE}{r_{\max} - r_{\min}} \tag{18}$$

b) Classification (or top-N recommendations)

For some systems, the exact score of an item is not relevant. Sometimes, users do not expect the RS to predict what score they would rate a certain item, and they would rather be simply presented a selection of items they will enjoy. Users will trust a RS the more they agree with the reduced set of items selected for them. In that case, knowing if some items have a bad score is not of interest as the system should simply not display them to the user. Researchers have thus focused on improving the top-N recommendations.

To evaluate the accuracy of these selected recommendations, the most widely used evaluation metrics are precision, recall and F1-score [39]. Those metrics typically evaluate the relevance or non-relevance of the items and were coined within the discipline of information retrieval. It is common to use a confusion matrix to display the four possibilities of any retrieval decision (Figure 2). Any item of the system can be categorized as "relevant" or "non-relevant" and can either be "selected" or "not selected" for recommendation. In the field of machine learning, we use the terms "True Positive (TP)", "True Negative (TN)", "False Positive (FP)", "False Negative (FN)".

	Relevant	**Non-relevant**
Selected	TP	FP
Not selected	FN	TN

Table 2: Confusion Matrix Representation

With these notations, we can define the recommendation metrics as:

$$\text{precision} = \frac{TP}{TP + FP} \tag{19}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{20}$$

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{21}$$

Precision measures the proportion of relevant items among those selected for recommendation, it shows the ability of the RS to display only useful information. On the other hand, *recall* measures how many items were selected among the relevant ones. It depicts how well the system can find all the useful items. The F1 score is the harmonic mean of precision and recall, it encapsulates the behaviour of both metrics

To study the capacity of the system to distinguish between relevant an irrelevant item, another option is to analyse the ROC (Receiver Operating Characteristics) curve that represents the recall against the fallout, which formula is given below (22). The curve is usually interpreted by calculating the Area Under the Curve (AUC).

$$\text{fallout} = \frac{FP}{FP + TN} \tag{22}$$

c) Ranked recommendations

Some systems require the selected items to be displayed in an ordered list, the items ranked the highest are predicted to be liked the most by the active user. Some of the most common ranking accuracy metrics are the NDCG (Normalized Discounted Cumulative Gain) and the half-life utility [12].

The half-life evaluates how satisfied a user is with a ranked list of recommendations by giving more weight to items near the top. The score of each item decreases exponentially the further it appears down the list. NDCG measures the ranking quality by rewarding relevant items found at higher positions. It applies a logarithmic discount based on position, meaning top-ranked items contribute more. The normalization ensures scores are comparable across users.

2.5.2 Other metrics

While most research focus on improving the accuracy of the predictions or of the recommendations, it is also important to have other goals in mind.

a) Novelty and diversity

Researchers have been trying to train algorithms that would give more diverse and innovative recommendations. In fact, since algorithms tend to make profit of the similarity among users or items, the recommendations can easily resemble each other too much, to the point where diversity would be clearly lacking. To prevent these "recommendation bubbles" from happening and enable a user to discover new content, different metrics have been proposed to evaluate the diversity of the results of the RS and bring attention to this issue. The most successful metrics are named *novelty* and *diversity*.

The novelty measure compares the list of items recommended to the user and the list of items known to the user and evaluate how different they are from each other. The diversity measure only looks at the list of recommended items and indicates the degree of variation within those recommendations. These diversity metrics are not standardized, therefore their definition may vary across papers; the formulas presented below can be found in [40]. The diversity of a recommendation list R of length N can be given by Equation 23. Equation 24 gives the novelty of an item $i \in L$, where $L \subseteq R$ is the set of items in R (items recommended) that the user likes; items in L can be known or unknown to the user.

$$\text{diversity}(R) = \frac{1}{N(N-1)} \sum_{i \in R} \sum_{j \in R, j \neq i} |1 - sim(i,j)| \tag{23}$$

$$\text{novelty}_L(i) = \frac{1}{N-1} \sum_{j \in L} |1 - sim(i,j)| \tag{24}$$

b) Learning rate

The learning rate metric evaluates how much time it takes for a recommender system before its suggestions are relevant and personalized to a new user [6]. It is typically computed by tracking how many ratings a user must provide before the system can generate useful recommendations. This is an important quality for a RS since new users are often not willing to spend much time on a platform if it does not satisfy them. Therefore, the RS needs to become effective within a short period of time to avoid losing the user.

c) Confidence of the system

The confidence of a system refers to its ability to assess the quality of its predictions. An item is usually recommended if it has a high predicted rating, but if the system is also capable of computing how confident it is with the rating, then it can choose to recommend only the items it is most certain about. Aiming for a confident system means prioritizing reliable suggestions and reducing the number of false positives. However, it introduces a trade-off as high confidence can potentially reduce coverage or novelty. [6]

2.6 Limitations

2.6.1 Data sparsity

The good functioning of RS algorithms relies heavily on the quality and density of the collected data. As mentioned before, a key challenge in CF is the high sparsity of the user-item ratings matrix due to users rating only a very small fraction of all the available items on a platform. This lack of information makes it difficult for the RS to produce reliable and personalized recommendations. Because many algorithms require a full matrix, the sparse ratings matrix often needs to be artificially filled using default values such as zeros, user or item means, or more advanced imputation techniques. However, these filling strategies can introduce noise and bias into the data, potentially leading to inaccurate recommendations.

Sparsity also contributes to *reduced coverage*, meaning that even if a user interacts with the system, many potentially relevant items cannot be recommended because they lack sufficient data to be properly evaluated. Another related issue is *neighbour transitivity*, where the system is not aware of the similarity between two users because they have not rated any of the same items. [13]

2.6.2 Cold-start problem

A specific problem arising from data sparsity is the phenomenon of *cold start* [7], [41]. It refers to the incapacity of the RS to produce relevant recommendations for new users or new items who have just entered the system and lack prior interaction data.

In the case of the introduction of a new item into the system, the problem is that it has not been rated by any user yet. As a result, it is unlikely that the item will be recommended because it cannot match any user or any other group of items. But if users do not get this item recommended, they will not know about its existence and will have no opportunity to rate it leading the item to go completely unnoticed. It can happen that a set of items is left out of the RS because of this cold-start issue. This problem does not have the same negative impact on every type of RS; in fact, an RS dealing with items that do not solely exist on this platform will not be impacted as much. The release of a new movie or a new song can be announced on another platform, and those items can be searched for and rated without needing to be recommended first. On the other hand, for e-commerce platforms which sell exclusive products, new items are more likely to go unnoticed [10].

In the case of a new user, the problem is similar but represents a bigger challenge. New users in an RS have not provided any rating, therefore they cannot get personalized recommendations. While new users are likely aware that the performance of the RS will improve over time, early poor suggestions can lead to dissatisfaction and disengagement and send the user to a competitor. To tackle the new user problem, hybrid filtering approaches are often used by combining CF with content-based or demographic methods to generate more reliable suggestions early on.

2.6.3 Data scalability

Finally, the issue of data scalability is becoming more prominent in the era of big data. As the number of users and items continues to grow, traditional CF algorithms can struggle to handle this amount of data efficiently. The computational complexity of traditional methods can become so high that it is no longer acceptable in practice, especially for systems that must respond in real-time and provide immediate recommendations. For instance, Spotify currently manages a catalogue of over 100 million tracks and has approximately 675 million active users. Similarly, Netflix must support personalized recommendations for around 300 million subscribers across a vast library of shows and movies. Therefore, ensuring that models can maintain their performance and responsiveness for large-scale datasets is a central challenge.

3 Methodology

Some of the collaborative filtering techniques reviewed before in the paper were applied on a real-world dataset. The goal of this experiment was to get in touch with the algorithms through their implementations, as well as evaluating and comparing them together. Adding this practical part in the research was a way to better understand the behaviour and functioning of recommender systems. In addition, it explicitly demonstrated the challenges often encountered when dealing with large-scale datasets. The following of this section provide a description of the dataset used, as well as the framework of the experiment.

The code of this experiment is available on GitHub at: github.com/diane-ch/ResearchPaper2025_CF-RS

3.1 Dataset description

This research is conducted using the MovieLens 1M dataset [42][8], a public and widely used benchmark dataset published by GroupLens. GroupLens is a research group from the University of Minnesota that has a long history of research on recommender systems, notably through their early work on USENET and collaborative filtering [43]. The MovieLens website is an online platform where registered users can rate and discover movies. Over the years, GroupLens has released several versions of the dataset, containing anonymous user ratings on a 1–5 scale. The 1M version includes 1,000,209 ratings from 6,040 users on 3,883 movies, and each user has rated at least 20 movies.

The dataset is composed of three interlinked tables (Table 3).

Table	**Shape**	**Format**
`ratings.dat`	(1000209, 4)	`UserID::MovieID::Rating::Timestamp`
`movies.dat`	(3883, 3)	`MovieID::Title::Genres`
`users.dat`	(6040, 5)	`UserID::Gender::Age::Occupation::Zip-code`

Table 3: Description of the three tables of MovieLens 1M dataset

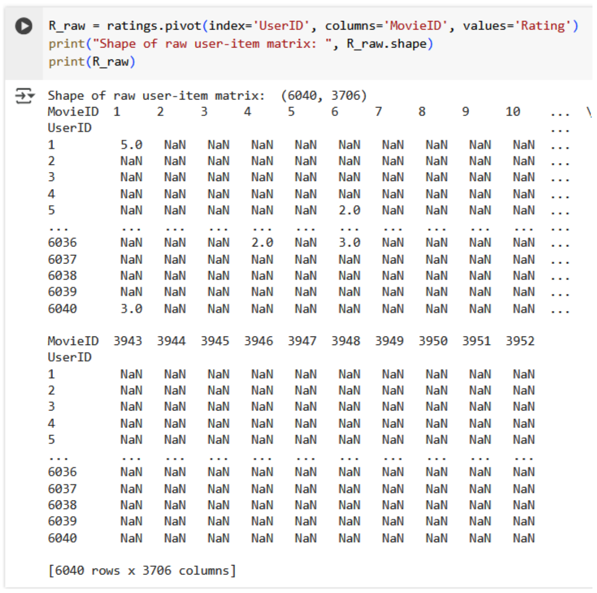
The core input for collaborative filtering algorithms is the user-item rating matrix derived from this data. This matrix is highly sparse as most users rate only a small subset of the available movies (Figure 1). Notably, 3131 users – more than half of the total– have rated 100 movies or fewer, and only a small minority have rated more than 500 as illustrated by the user activity distribution graph (Figure 2).

3.2 Algorithms implemented

Three different model-based CF techniques were explored. First, the two dimensionality reduction approaches SVD and NMF were implemented and compared, with a particular attention to the choice of the hyperparameters. Then, a clustering-based approach using K-means was applied to the dataset; users were assigned into clusters and the recommendations were made by averaging the ratings within those clusters for each item. The influence the data structure has on the model's perfomance was analyzed by varying the size and density of the training user-item rating matrix.

3.3 Programming language

The project was implemented in Python and used the following key libraries. First, *Surprise* was very useful as it is a specialized library for recommender systems; it was used for the implementation of the matrix factorization techniques SVD and NMF. Since it does not incorporate KMeans, the clustering-based algorithm was implemented with *Scikit-learn*. Then, *Pandas* was also employed for several steps in pre-processing such as loading and preparing the dataset. All the visualization graphs were made using the usual *Matplotlib* library.



```
R_raw = ratings.pivot(index='UserID', columns='MovieID', values='Rating')
print("Shape of raw user-item matrix: ", R_raw.shape)
print(R_raw)
```

```
Shape of raw user-item matrix:  (6040, 3706)
MovieID  1     2     3     4     5     6     7     8     9     10    ...  \
UserID                                                               ...
1         5.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
2         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
3         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
4         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
5         NaN   NaN   NaN   NaN   NaN   2.0   NaN   NaN   NaN   NaN  ...
...       ...   ...   ...   ...   ...   ...   ...   ...   ...   ...  ...
6036      NaN   NaN   NaN   2.0   NaN   3.0   NaN   NaN   NaN   NaN  ...
6037      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
6038      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
6039      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
6040      3.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...

MovieID  3943  3944  3945  3946  3947  3948  3949  3950  3951  3952
UserID
1         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
2         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
3         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
4         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
5         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
...       ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
6036      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
6037      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
6038      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
6039      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
6040      NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

[6040 rows x 3706 columns]
```

Figure 1: Sparse rating matrix (NaNs are empty values)

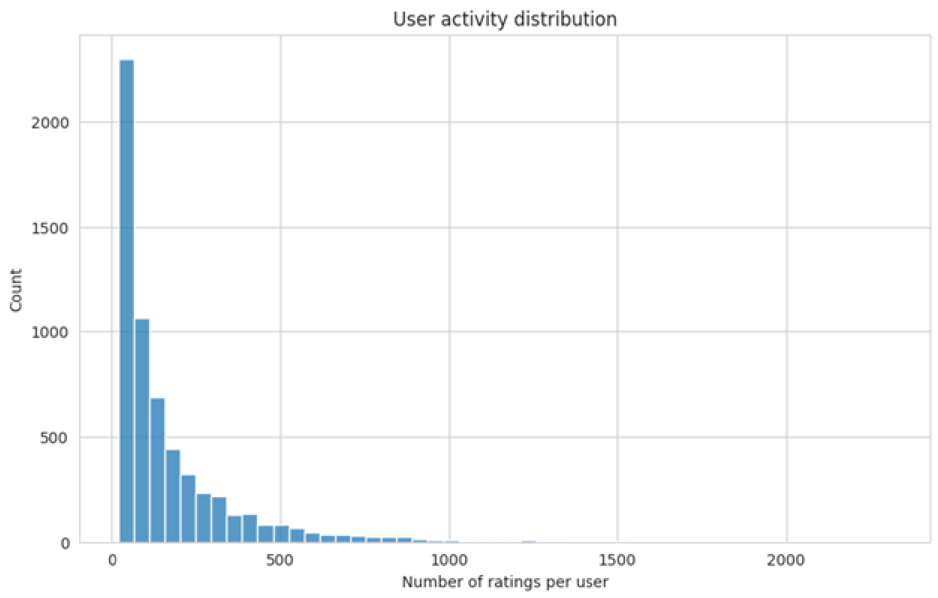


Figure 2: User activity distribution

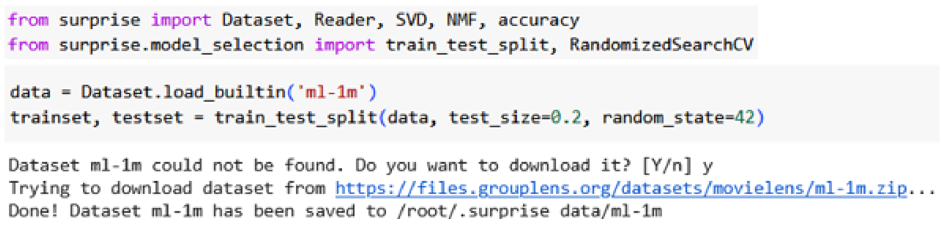
3.4 Evaluation metrics used

Two types of evaluation were conducted depending on the algorithms. First, SVD and NMF were compared on their rating prediction accuracy using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), as they are two standard metrics for this task. Second, the KMeans algorithm was evaluated on the quality of the recommendations made by the system. Precision@N, Recall@K and F1-score were computed to evaluate the relevance of the top-N recommendation list generated for each user.

4 Experiment: implementations, results and analysis

This section presents the experiment conducted on the MovieLens 1M dataset [8]. The detailed process of the implementations is provided, with insights on why certain choices were made (computational cost, time limits, etc...). The results are analyzed and the limitations of the experiment are addressed at the end of the section.

The *Surprise* library enables the download of the MovieLens dataset directly through a built-in function `load_builtin('ml-1m')`. The dataset is then saved in the root folder of the notebook and available for manipulation. *Surprise* also provides a function to split the data into a trainset and a subset. The two subsets were therefore created at the beginning and are used throughout the various algorithms (`trainset` and `testset`); the same testing set is used for every technique which allows a fair comparison. With the common specification of `test_size=0.2`, the training set contains approximately 800,000 ratings and the testing set approximately 200,000. Furthermore, adding `random_state=42` is to enable reproduciblity - any value would work, but the number 42 is commonly used in the field. Figure 3 shows these initial steps.



```
from surprise import Dataset, Reader, SVD, NMF, accuracy
from surprise.model_selection import train_test_split, RandomizedSearchCV

data = Dataset.load_builtin('ml-1m')
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

Dataset ml-1m could not be found. Do you want to download it? [Y/n] y
Trying to download dataset from https://files.grouplens.org/datasets/movielens/ml-1m.zip...
Done! Dataset ml-1m has been saved to /root/.surprise data/ml-1m
```

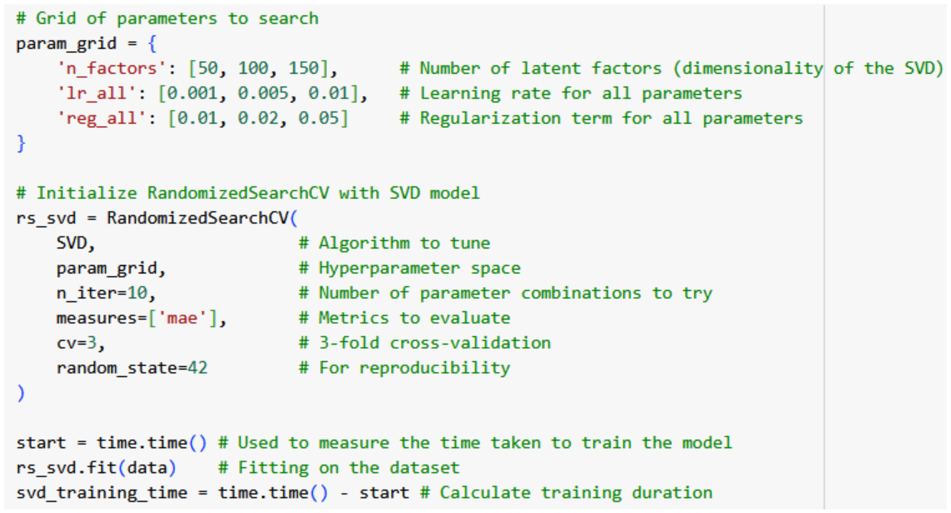
Figure 3: Dataset loaded and split into train/test subsets

4.1 Dimensionality Reduction approaches

The first part of the experiment consisted of exploring two dimensionality reduction techniques used in recommender systems: singular value decomposition (SVD) and non-negative matrix factorization (NMF). *Surprise* directly provides an implementation for both of these algorithms, which is very useful.

4.1.1 Singular Value Decomposition (SVD)

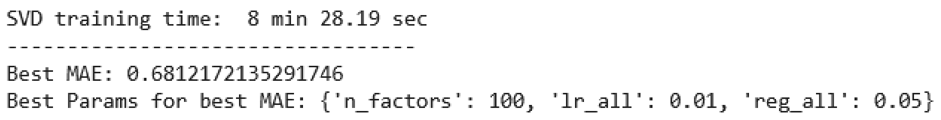
The function `surprise.SVD()` requires a set of hyperparameters to build the model: `n_factors` the number of latent features, `lr_all` the learning rate, and `reg_all` the regularization parameter. The value of each of these parameters play a significant role in the creation of the model and therefore needs to be chosen carefully. The `RandomizedSearchCV` method was used to find relevant parameters: given a grid of hyperparameters, this method randomly picks various combinations, builds a model using those hyperparameters and outputs the best combination. The value of `n_iter` indicates how many combinations are tested. The values of `param_grid` - seen on Figure 4 - were chosen by picking a range of common values for each of the parameters. Also, cross-validation is directly implemented into `RandomizedSearchCV` through the argument `cv=3`; this means that for each combination, the data is split into three folds and the model is trained three times using a different fold as the test set each time. This helps preventing overfitting and makes the model more robust.



```
# Grid of parameters to search
param_grid = {
    'n_factors': [50, 100, 150],      # Number of latent factors (dimensionality of the SVD)
    'lr_all': [0.001, 0.005, 0.01],   # Learning rate for all parameters
    'reg_all': [0.01, 0.02, 0.05]     # Regularization term for all parameters
}

# Initialize RandomizedSearchCV with SVD model
rs_svd = RandomizedSearchCV(
    SVD,                        # Algorithm to tune
    param_grid,                 # Hyperparameter space
    n_iter=10,                  # Number of parameter combinations to try
    measures=['mae'],           # Metrics to evaluate
    cv=3,                       # 3-fold cross-validation
    random_state=42             # For reproducibility
)

start = time.time() # Used to measure the time taken to train the model
rs_svd.fit(data)    # Fitting on the dataset
svd_training_time = time.time() - start # Calculate training duration
```

Figure 4: Training of the SVD model and use of `RandomizedSearchCV` to find optimal parameters

Once the hyperparameters that contributed to the best MAE score are found (Figure 5), they are used to create the "optimized" SVD model. While these hyperparameters constitute the best found combination, it does not necessarily mean that no better combination exists. Indeed, the randomized search technique only tests values that are given in `param_grid`, so it is always possible that a better score would be given by other values. In addition, it only tests `n_iter` combinations randomly, which can be less than the total number of possible combinations in the grid. The other method `SearchGridCV` tests every combination possible of the given hyperparameter grid but this technique takes more time to run, which is why the randomized version was chosen.



```
SVD training time:  8 min 28.19 sec
---------------------------------
Best MAE: 0.6812172135291746
Best Params for best MAE: {'n_factors': 100, 'lr_all': 0.01, 'reg_all': 0.05}
```

Figure 5: Optimal hyperparameters found through `RandomizedSearchCV` (SVD model)

Once this optimized model is fitted on the initial `trainset`, it is ready to be used for comparison with NMF. Below are the formulas that `surprise.SVD()` uses for the prediction $\hat{r}_{ui}$ (25), the regularized squared error to minimize (26), and the SGD performed for that minimization (27). The bias parameters (`b_u,b_i`) are there to account for systematic tendencies of users to give high or low ratings (and of items to receive higher or lower ratings).

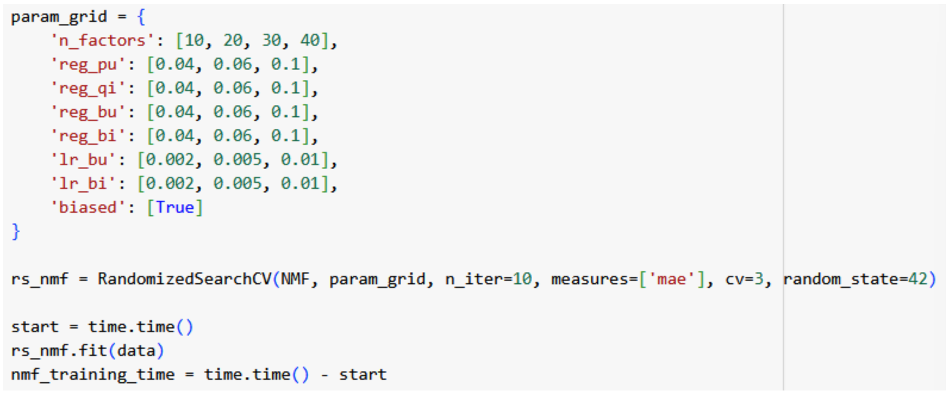
$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_i \tag{25}$$

$$\sum_{r_{ui}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{26}$$

$$\begin{cases} b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u) \\ q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i) \end{cases} \tag{27}$$

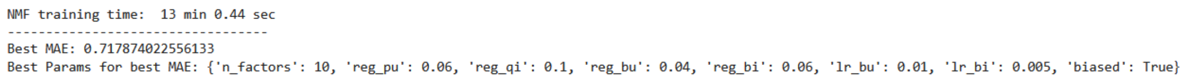
4.1.2 Non-negative matrix factorization (NMF)

The framework followed for the implementation of the NMF model is similar to the SVD one: a randomized search is used to find the best hyperparameters of the model and the resulting optimized model is fitted on the `trainset`. The main difference is that the common range of values for the hyperparameters is slightly different than for SVD. Furthermore, the function `surprise.NMF()` does not allow to set one regularization value and one learning rate value for all parameters so there are more hyperparameters in the grid, as shown by Figure 6. While the same range of values is given for the four regularization parameters (`reg_pu, reg_qi, reg_bu, reg_bi`) and the two learning rate parameters (`lr_bu, lr_bi`), the `RandomizedSearchCV` can - rightly so - choose different values for each as the best combination so it differs a bit from the SVD hyperparameters tuning. Finally, `surprise.NMF()` does not compute the "biased" version of the algorithm by default, which is why the parameters `biased=True` needs to be specified. That way, the rating prediction is computed in the same way as for SVD: $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$. Figure 7 shows the hyperparameters that were picked after the randomized search; this search took 13 minutes which is more time than `RandomizedSearchCV` took for SVD (8 minutes) even though both tested `n_iter=10` combinations. This suggests that the computation of NMF is more costly.



```
param_grid = {
    'n_factors': [10, 20, 30, 40],
    'reg_pu': [0.04, 0.06, 0.1],
    'reg_qi': [0.04, 0.06, 0.1],
    'reg_bu': [0.04, 0.06, 0.1],
    'reg_bi': [0.04, 0.06, 0.1],
    'lr_bu': [0.002, 0.005, 0.01],
    'lr_bi': [0.002, 0.005, 0.01],
    'biased': [True]
}

rs_nmf = RandomizedSearchCV(NMF, param_grid, n_iter=10, measures=['mae'], cv=3, random_state=42)

start = time.time()
rs_nmf.fit(data)
nmf_training_time = time.time() - start
```

Figure 6: Training of the NMF model and use of `RandomizedSearchCV` to find optimal parameters



```
NMF training time:  13 min 0.44 sec
-----------------------------------
Best MAE: 0.717874022556133
Best Params for best MAE: {'n_factors': 10, 'reg_pu': 0.06, 'reg_qi': 0.1, 'reg_bu': 0.04, 'reg_bi': 0.06, 'lr_bu': 0.01, 'lr_bi': 0.005, 'biased': True}
```

Figure 7: Optimal hyperparameters found through `RandomizedSearchCV` (NMF model)

4.1.3 Comparison

The two optimized model of SVD and NMF were fitted on `trainset` and, as suggested by the duration of the randomized search, the fitting of NMF is a bit longer (Figure 8). At this scale, the time difference is negligible but it becomes already more significant during the hyperparameters tuning phase and would increase with larger datasets.

The two models are now evaluated and compared for their prediction accuracy using the same testing set for both, prepared from the beginning. The predicted ratings for each user-item pair in `testset` are computed using the `test()` function. For each model, a list of predictions is generated with each prediction including the user ID, item ID, true rating, and estimated rating as shown on Figure 9.

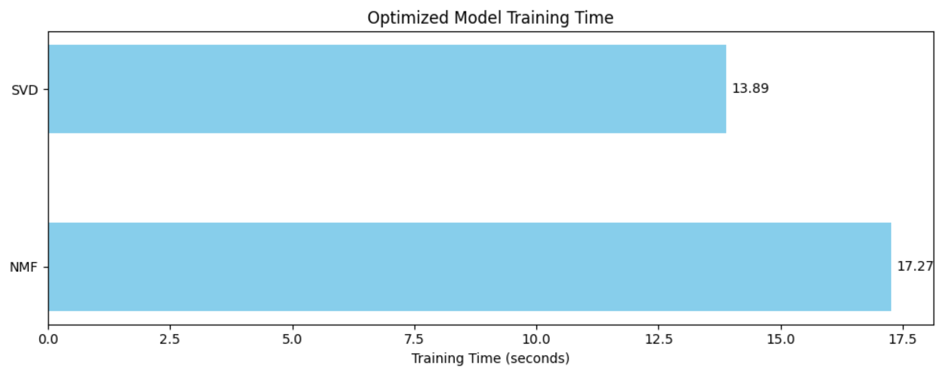
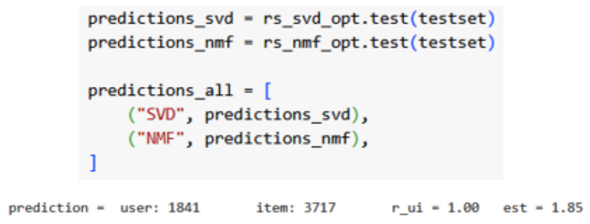


Figure 8: Comparison of SVD and NMF fitting time on `trainset`



```
predictions_svd = rs_svd_opt.test(testset)
predictions_nmf = rs_nmf_opt.test(testset)

predictions_all = [
    ("SVD", predictions_svd),
    ("NMF", predictions_nmf),
]
```

```
prediction =  user: 1841       item: 3717       r_ui = 1.00   est = 1.85
```

Figure 9: Computation of the rating predictions using both SVD and NMF models

The two models are compared using MAE and RMSE as they are both standard evaluation metrics for prediction accuracy. Figure 10 shows that the SVD model obtains a MAE-score of 0.672 and RMSE-score of 0.854, while NMF has a MAE-score of 0.717 and RMSE of 0.906. The lower these error metrics, the better the model performs. Therefore, the SVD model has provided better predictions that NMF - even though the scores between the two models are relatively close.

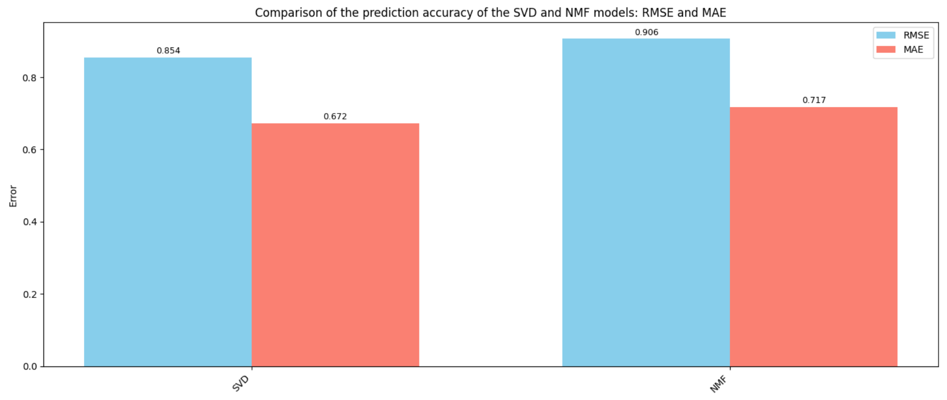


Figure 10: Comparison of the MAE and RMSE of SVD and NMF models

The two graphs on Figure 11 offer a more detailed representation of the prediction performance of both models. They represent how frequently a certain marge of error was made. The decreasing curve indicates there is a majority of small errors rather than large errors. Since the MovieLens dataset works with a 1-5 scale of integer numbers, a prediction is incorrect if its absolute difference with the true rating is more than 0.5. This is what the bar chart aims to represent; it shows that out of the approximate 200,000 test ratings, around half of them were correct and half of them were 1-point away from being correct. Considering the basic implementations of these models, these results are relatively satisfying.

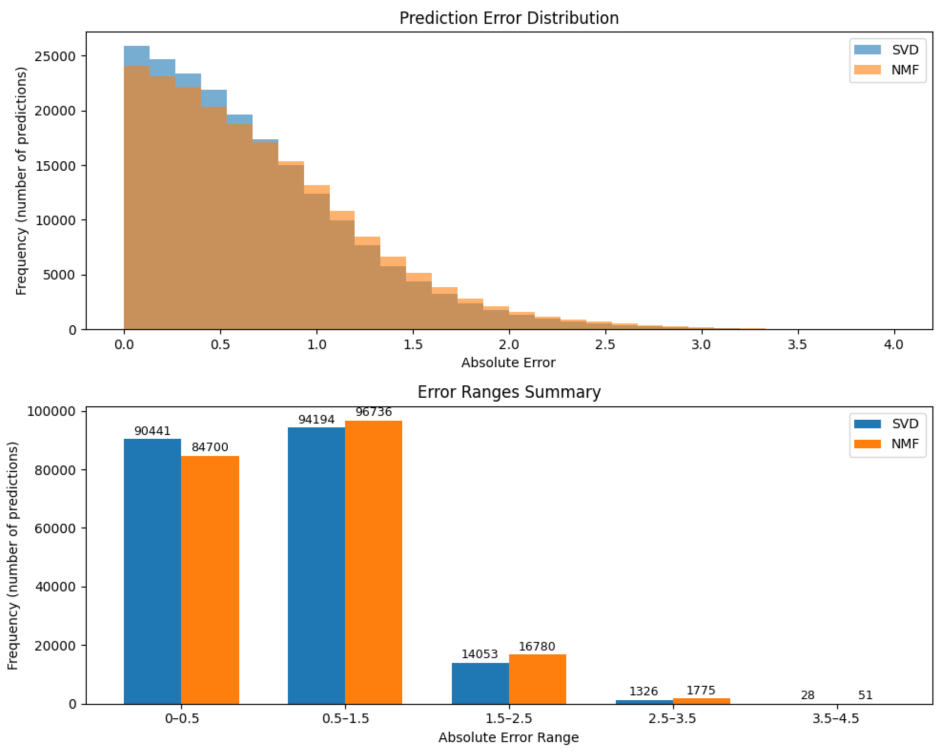


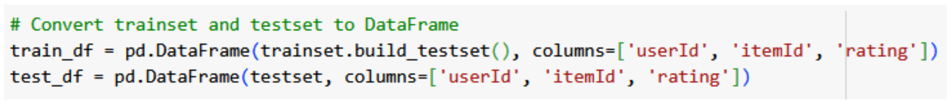
Figure 11: Graphs comparing the absolute errors of SVD and NMF

4.2 Clustering-based approach: K-Means

As a second step, a clustering-based approach for recommending items is explored through the K-means algorithm. The *Surprise* library does not contain an already-implemented K-means algorithm, so the *Scikit-learn* library was used this time.

4.2.1 Framework of the algorithm

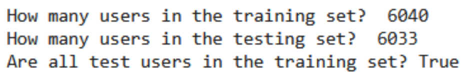
1) Train/test split
The **trainset** and **testset** created at the beginning of the experiment are used to create two user-item ratings matrix. They have to be converted to a Pandas dataframe as the *Surprise* library creates "trainset" object. The training matrix will be used for clustering users, computing the average rating within the clusters and generating the recommendations. The test set is only use to evaluate the said recommendations.



```
# Convert trainset and testset to DataFrame
train_df = pd.DataFrame(trainset.build_testset(), columns=['userId', 'itemId', 'rating'])
test_df = pd.DataFrame(testset, columns=['userId', 'itemId', 'rating'])
```

Figure 12: Conversion of `trainset` et `testset` to Pandas dataframe

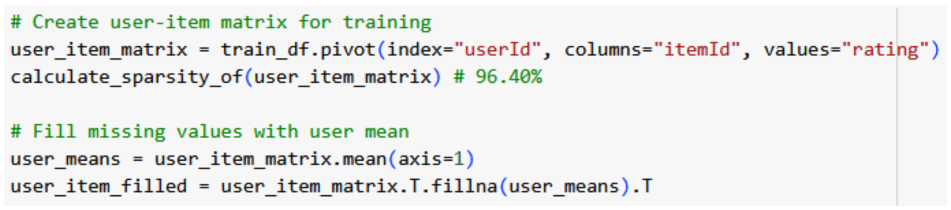
2) Check if all test users are part of the training
In practice, for each user, a portion of their ratings is hidden from the recommender system and will be used as ground truth to evaluate the recommendation list. However, to be able to use the test set as a verification, it is necessary that all the users present in the test set were also part of the training - just not with the same ratings. A user that was not part of the training would not be assigned to any cluster which would make it impossible to generate recommendations using the clustering approach. In other words, the system must train and test on the same users, but with different items. A simple verification showed that 6,033 users were in the test set and they were all part of the training set - the latter actually contains ratings from every user of the dataset (6,040 users in total) (Figure 13).



```
How many users in the training set?  6040
How many users in the testing set?  6033
Are all test users in the training set? True
```

Figure 13: Verification that every test users had some of their ratings in the training set

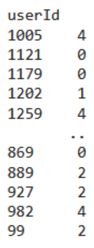
3) Prepare the training rating matrix
The dataframe `train_df` is converted into a matrix with users as rows and items as columns. The `user_item_matrix` has many missing values because users do not rate many items; precisely, it is 96.40% sparse. Since, the K-means algorithm requires a full matrix, the missing values for each user are replaced with the mean rating of that user: Figure 14.



```
# Create user-item matrix for training
user_item_matrix = train_df.pivot(index="userId", columns="itemId", values="rating")
calculate_sparsity_of(user_item_matrix) # 96.40%

# Fill missing values with user mean
user_means = user_item_matrix.mean(axis=1)
user_item_filled = user_item_matrix.T.fillna(user_means).T
```

Figure 14: Fill training rating matrix with user means

4) Clustering process
Now that all the pre-processing is complete, the clustering process can begin. Only the number of desired clusters needs to be specified to `sklearn.KMeans()`; here, `k_clusters = 5` was chosen to be the best value which the following section explains in more details. Once the K-means model is fitted on the training matrix `user_item_filled`, every user has been assigned to one of the five created clusters. Figure 15 illustrates how every user is now part of a cluster that is designated by a number between 0 and 4.



```
userId
1005    4
1121    0
1179    0
1202    1
1259    4
       ..
869     0
889     2
927     2
982     4
99      2
```

Figure 15: Users (left) and their assigned clusters (right)

5) Generating recommendations
To generate recommendations for a user u, the first step is to identify which cluster they belong to. Then, identify all the items in the training set that the user has not rated. For each of these unrated

items, an average rating is computed using all the other users that are part of the cluster and who have rated the said item. It is possible that some items are not rated by any user of a cluster, in that case those items will simply not be recommended to user u. The items are ordered according to the average rating that was computed, and the N first items with the highest ratings constitute the top-N recommendation list of user u.

6) Evaluating recommendation quality
The final step is to evaluate the recommendation list; in other words: "Would user u actually enjoy the recommended items?". This question can be answered by looking at the actual ratings of user u that are in the `testset`. An item i is considered to have been enjoyed if it has a rating of at least 4, meaning user u rated the movie 4 or 5 stars. All the items that u has enjoyed are the "relevant items". The quality of the recommendations is evaluated using the metrics of precision (*how many of the suggestions are relevant?*), recall (*how many of the user's relevant items were successfully suggest?*), and F1-score, which combines the two previous ones (Figure 28).

$$\begin{cases} \text{Precision} = \dfrac{|\text{Relevant items} \cap \text{Recommended items}|}{|\text{Recommended items}|} \\ \text{Recall} = \dfrac{|\text{Relevant items} \cap \text{Recommended items}|}{|\text{Relevant items}|} \\ \text{F1-score} = \dfrac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{cases} \tag{28}$$

4.2.2 Choosing the number of clusters K

To pick in how many clusters the data should be partitioned, two classic methods were used: the elbow method and silhouette analysis.

The **elbow method** consists of plotting the *inertia* as a function of K. The inertia is the sum of squared distances from each point to its assigned cluster centre. Increasing the number of clusters K results in a decreasing inertia, but the goal is to identify a point where the rate of decrease suddenly slows down. The value of the "elbow" constitute the optimal trade-off between the model simplicity and the compactness of the clusters.

The **silhouette score** corresponds to how well each data point fits within its cluster compared to other clusters. The score can vary from -1 to 1, with higher values representing better defined clusters.

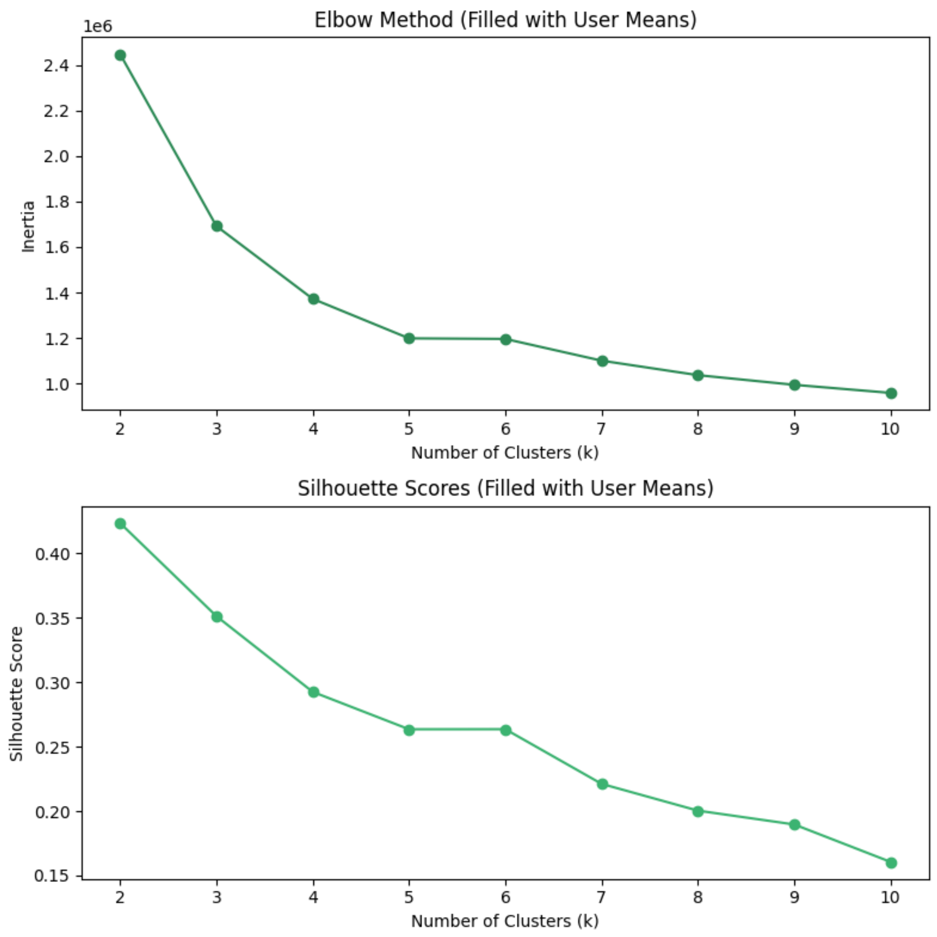


Figure 16: Choosing a number of clusters k with the elbow method and silhouette analysis

The quality of the clustering was tested for values of K ranging from 2 to 10. As shown on Figure 16, none of the two graphs show a clear optimal value but considering the plateau appearing around five and six clusters, the value $k = 5$ can be considered as the best tradeoff. Therefore, the users were grouped into five clusters.

These clusters can be represented more visually by plotting the average item ratings per cluster. The graph on Figure 17 reveals an evident trend of rating for each cluster. For instance, users grouped in cluster 1 tend to give low ratings as their average rating is around 2.7. By contrast, cluster 2 regroups users who tend to rate highly, as the average rating of that cluster is approximately 4.4.

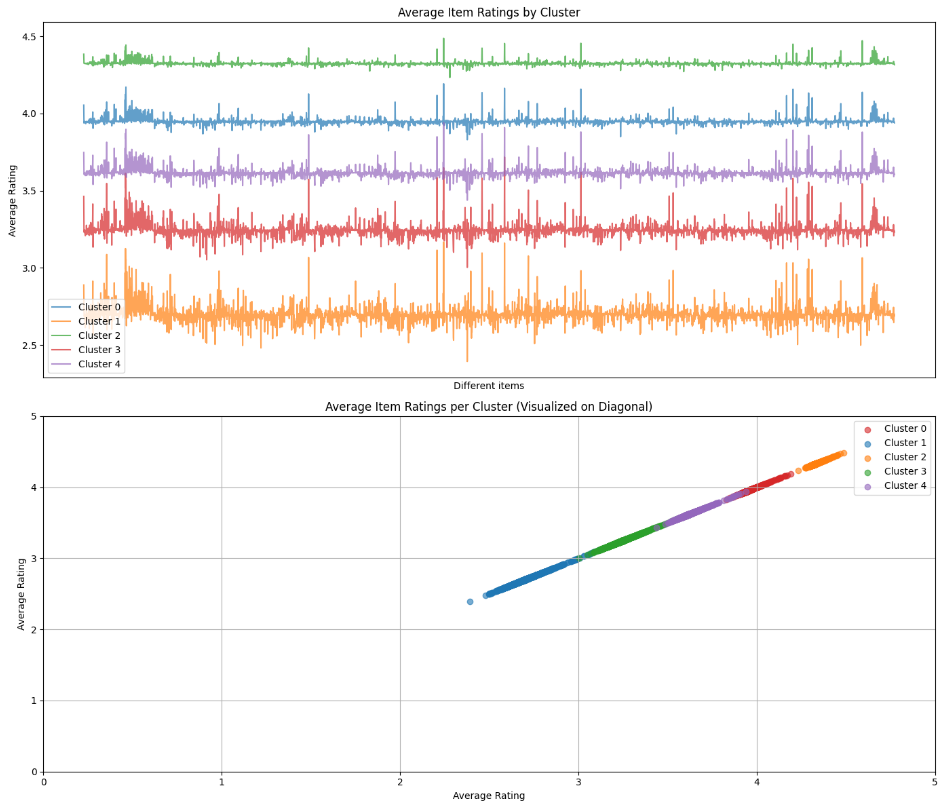


Figure 17: Visualization of the $k = 5$ clusters

4.2.3 Evaluating the recommendations for one specific user u

A first approach to evaluating the model was to evaluate the recommendations made for one specific user u. Considering the large size of the dataset and computational time it takes to compute the recommendations for every user, this acts as a quick check and offers preliminary insights. The different steps outlined previously in 4.2.1 for generating the recommendations and evaluating the list were implemented into the function `evaluate_top_n_recommendations()` presented on Figure 19.

First, the top-10 recommendation list of user 12 was computed and evaluated; the model did very bad as every metric scored 0 (Figure 18). Following these poor results, the length of the recommention list was increased to check whether the scores of zero were due to an implementation error or were the actual results. So, the top-100 recommendation list of user 12 was computed and this time the evaluation metrics were not null, but low nonetheless (Figure 20).

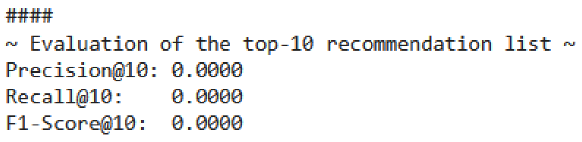
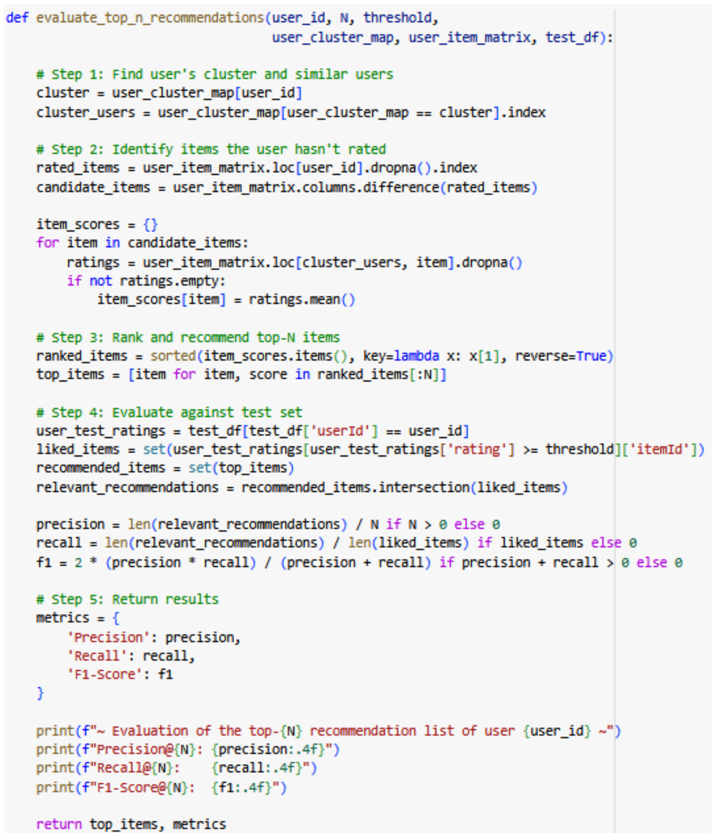
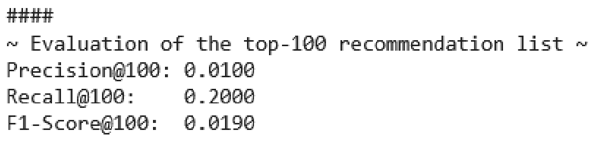

```
####
~ Evaluation of the top-10 recommendation list ~
Precision@10: 0.0000
Recall@10:    0.0000
F1-Score@10:  0.0000
```

Figure 18: Evaluation metrics for the top-10 of user 12



```python
def evaluate_top_n_recommendations(user_id, N, threshold,
                                   user_cluster_map, user_item_matrix, test_df):

    # Step 1: Find user's cluster and similar users
    cluster = user_cluster_map[user_id]
    cluster_users = user_cluster_map[user_cluster_map == cluster].index

    # Step 2: Identify items the user hasn't rated
    rated_items = user_item_matrix.loc[user_id].dropna().index
    candidate_items = user_item_matrix.columns.difference(rated_items)

    item_scores = {}
    for item in candidate_items:
        ratings = user_item_matrix.loc[cluster_users, item].dropna()
        if not ratings.empty:
            item_scores[item] = ratings.mean()

    # Step 3: Rank and recommend top-N items
    ranked_items = sorted(item_scores.items(), key=lambda x: x[1], reverse=True)
    top_items = [item for item, score in ranked_items[:N]]

    # Step 4: Evaluate against test set
    user_test_ratings = test_df[test_df['userId'] == user_id]
    liked_items = set(user_test_ratings[user_test_ratings['rating'] >= threshold]['itemId'])
    recommended_items = set(top_items)
    relevant_recommendations = recommended_items.intersection(liked_items)

    precision = len(relevant_recommendations) / N if N > 0 else 0
    recall = len(relevant_recommendations) / len(liked_items) if liked_items else 0
    f1 = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0

    # Step 5: Return results
    metrics = {
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    }

    print(f"~ Evaluation of the top-{N} recommendation list of user {user_id} ~")
    print(f"Precision@{N}: {precision:.4f}")
    print(f"Recall@{N}:    {recall:.4f}")
    print(f"F1-Score@{N}:  {f1:.4f}")

    return top_items, metrics
```

Figure 19: Evaluating the recommendation list for one user u



```
####
~ Evaluation of the top-100 recommendation list ~
Precision@100: 0.0100
Recall@100:    0.2000
F1-Score@100:  0.0190
```

Figure 20: Evaluation metrics for the top-100 of user 12

To better observe the influence of the number of recommended items N, multiple recommendation list of varying length were computed for user 12. Figure 21 shows the evolution of precision, recall and F1 for N ranging from 5 to 500; the main observation is the presence of distinct plateaus in the evolution of recall. This behaviour is explained by the fact that user 12 had only 5 relevant items (rating ≥ 4) in the test set. Therefore, recall is at $1/5$ when one relevant is in the recommendation list, then, when increasing the length of the recommendation other relevant items are found gradually: recall only varies from $1/5$ to $2/5$, etc... and will reach 1 once the five items are found. On the other hand, precision drops as N increases because more items are being recommended but only five can possibly match.

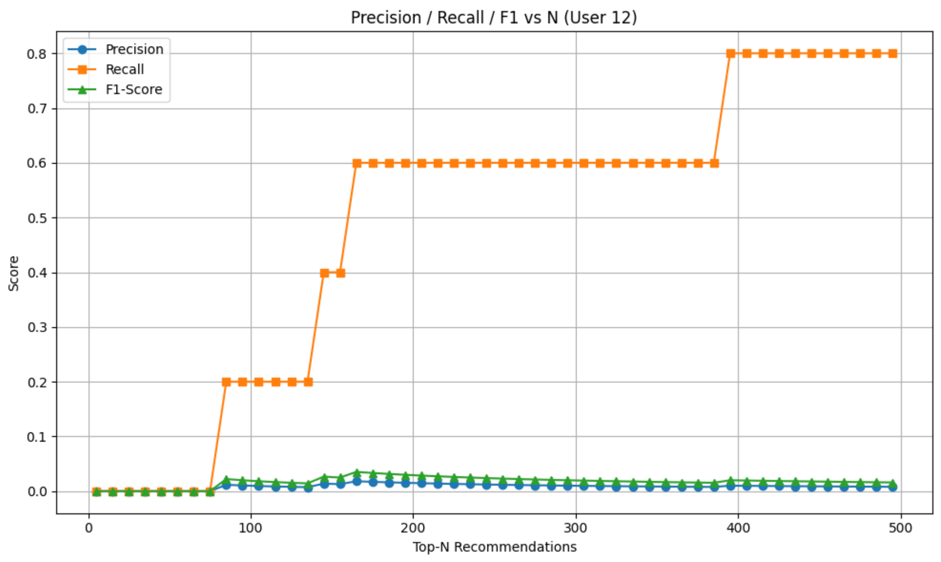


Figure 21: Evolution of the metrics when the length N of the recommendation list increases (user 12)

This behaviour highlights that evaluation metrics cannot be interpreted without context. For users with very few relevant items, the benefits of increasing N are limited. To analyze the model on a different type of user, the same process was applied to a user 4277 - it is the user who has the most relevant items in the test set (312 items with a rating above 4: Figure 22). As shown on Figure 24, the precision, recall and F1-score of user 4277 have a more gradual and continuous increase, without the plateaus observed for user 12. In this K-means experience, the metrics vary more meaningfully when the user has a higher number of relevant items. However, even with more chances to suggest relevant items, the model fails to retrieve one in the top-10 recommendations (Figure 23).

This suggests that the sparsity of the rating matrix actively hinders the model's performance: the computed average scores to fill the matrix are based on too few ratings. In addition, these basic averages cannot capture individual preferences, so the more these approximations are done (i.e. the sparser the matrix), the more the model ignores nuance in user profiles.

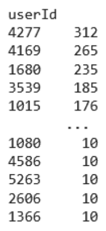

```
userId
4277    312
4169    265
1680    235
3539    185
1015    176
        ...
1080     10
4586     10
5263     10
2606     10
1366     10
```

Figure 22: Number of "relevant" items in the test set for each user

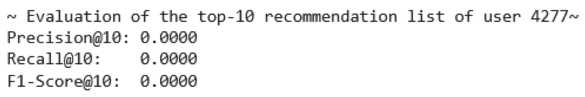

```
~ Evaluation of the top-10 recommendation list of user 4277~
Precision@10: 0.0000
Recall@10:    0.0000
F1-Score@10:  0.0000
```

Figure 23: Evaluation metrics for the top-10 of user 4277

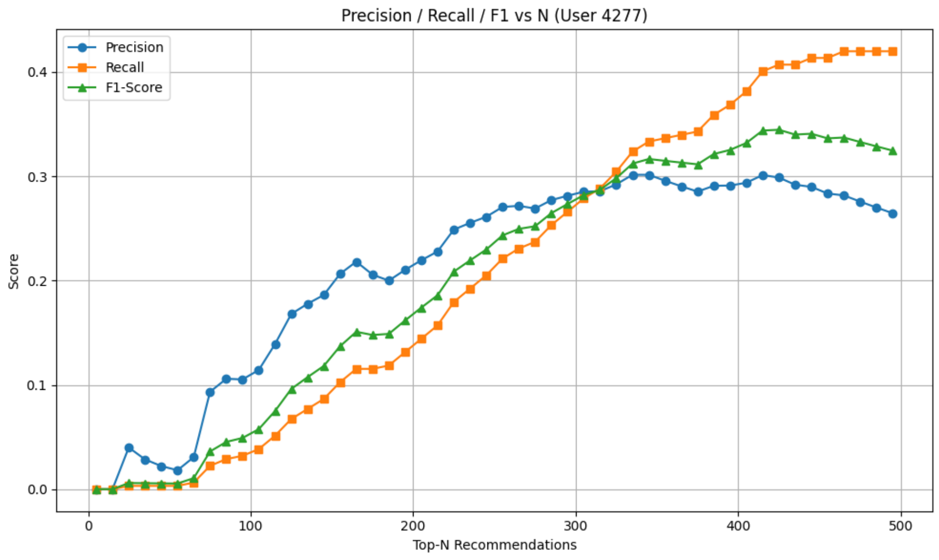


Figure 24: Evolution of the metrics when the length N of the recommendation list increases (user 4277)

4.2.4 Influence of the size and density of the rating matrix

To evaluate the average performance of the model on multiple users, several subsets of varying size were extracted from the dataset. Due to computational cost, this method was chosen instead of using the entire dataset; computing the recommendations for the 6,040 users that were part of the training set took five hours, the need for reproduciblity and handling the implementation errors made that not an option. So, subsets were used instead, which was enough to find interesting findings and coherent results.

Initially, the model was evaluated using square subsets, meaning that the numbers of users and items were identical. Figure 25 shows the variation of the evaluation metrics *Precision@10, Recall@10* and *f1@10* when the number of user-item ranges from 20 to 200. The graph illustrates that the recommendation quality declines as subsets become larger.

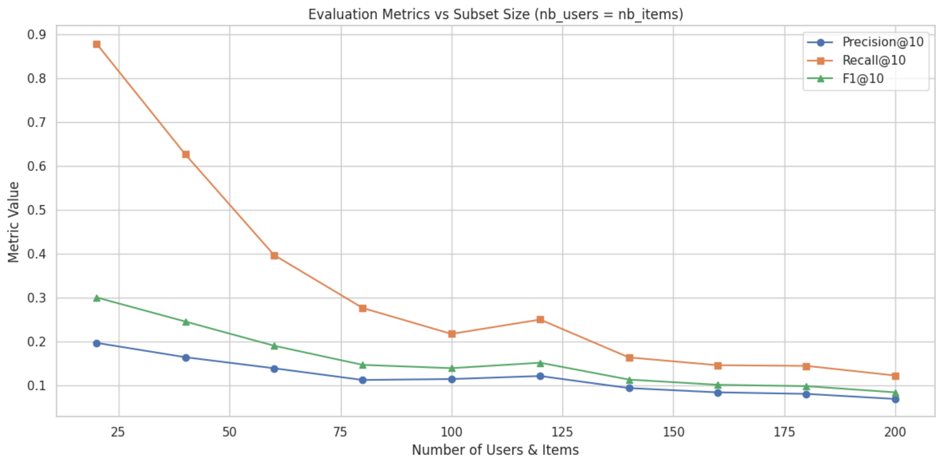


Figure 25: Evolution of the evaluation metrics with the subset size

Figure 26 offers another type of representation of the influence of the subset size, with a variation of 20 to 100 users/items. All three performance metrics, as well as the sparsity of the training matrix, are visualized as heatmaps - the number of items increases from left to right, while the number of users increases from bottom to top. The gradient in performance is predominantly vertical which indicates that the number of items has a greater influence on recommendation quality than the number of users. In addition, the heatmap representing matrix sparsity (bottom-right corner) shows that sparsity increases with the number of users.

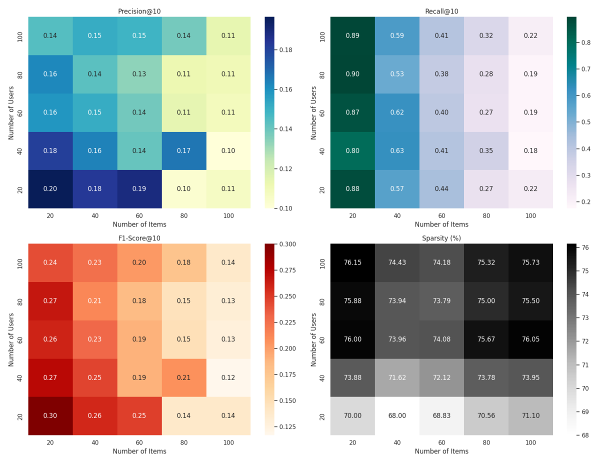


Figure 26: Heatmaps showing the influence of the number of users and items

To confirm the effects of each dimension, asymetric subsets were tested: the range of items stayed at [25, 50, 75, 100] while the range of users was enlarged to [250, 500, 750, 1000]. This confirmed that a large number of users did not decrease the performance of the model as much as a larger number of items. Therefore, the dominant factor affecting performance in this basic recommendation model is the number of items.

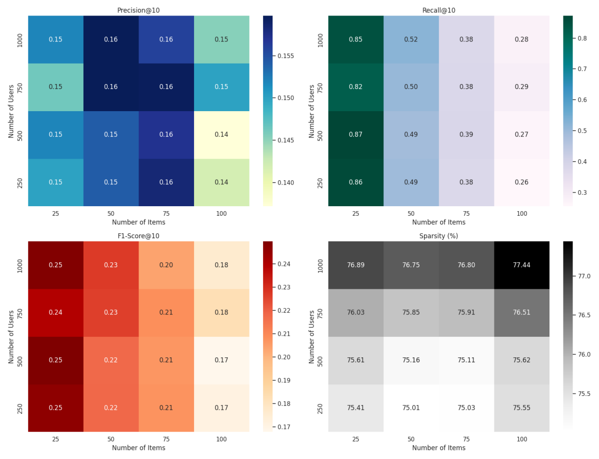


Figure 27: Heatmaps showing the influence of the number of users and items (with a larger variation for users)

4.3 Concluding remarks

This experiment provided insights into several common algorithms used in recommender systems. The comparison between SVD and NMF models showed that both methods offer effective predictions. The SVD model outperformed NMF in the accuracy metrics MAE and RMSE, and was also faster to compute. This suggests that, for this specific dataset and configuration, SVD is more efficient in generalizng user preferences. The K-Means section illustrated clustering approaches where users with similar rating patterns are put into the same cluster. This algorithm was significantly affected by the sparsity of the user-item matrix (96.4%); it showed that computing simple average ratings to fill the matrix is not enough to achieve good recommendations, especially for a limited list of items such as in the top-N context where precision matters greatly. Additionally, these experiments highlighted the need for scalable algorithms as a simple K-Means algorithm already takes too much time to run on the entire dataset.

Overall, these experimentations particularly demonstrated the need for dimensionality reduction techniques to densify the data into key features. It was initially intended to conclude this experiment with the implementation of a hybrid approach - applying the SVD algorithm before doing K-means. The goal was to observe whether this combination could improve the precision in recommendations compared to a singular K-means model. However, due to time constraints, the hybrid model could not be implemented.

5 Further discussions and broader perspectives

While this paper focused on basic implementations of matrix factorization and clustering, the field of recommender systems evolves constantly and is much broader. This final section briefly outlines some other key areas of RS that go beyond the scope of this study.

First, Bayesian models are common approaches in recommendation tasks that make use of probabilistic theories. Indeed, unlike standard methods where parameters are fixed values, Bayesian models represent unknown parameters as probability distributions. By doing so, these approaches introduce the notion of uncertainty into models which makes them more robust and better at handling noisy and sparse data, typical of real-world situations. In recommender systems, Bayesian models are often used in combination with matrix factorization. For example, Salakhutdinow and Mnih [44] presented a Probabilistic Matrix Factorization (PMF) model which performs well on large and sparse datasets. A year later, they proposed a Bayesian version (BPMF) that uses Markov chain Monte Carlo (MCMC) methods, and which achieved a higher prediction accuracy than a regular PMF model. The Bayesian Personalized Ranking (BPR) in [45] focused on the optimization of rankings over ratings.

Another active area of research is the use of reinforcement learning (RL) for recommender systems. RL models aim to maximize long-term engagement over improving immediate prediction accuracy. By modelling interactions over time, they adopt strategies that consider how current recommendations would influence user behaviour in the future. In other words, RL systems can learn to make better decisions based on how users react to previous recommendations. Zou et al. [46] propose a framework where recommendations are made based on user trajectories. More recently, Giannikis et al. [47] explored how RL could help solving the cold-start problem with models that adapt quickly to new users' preferences based on exploration strategies.

In recent years, deep learning (DL) methods for recommender systems have also been intensively studied and reviewed [48], [49], [50]. Neural networks are DL techniques that can model nonlinear and complex user-item interactions better than traditional techniques. Deep matrix factorization combines latent factor models with deep neural architectures which allows for richer representations. For example, Singh et al. [51] demonstrate how deep matrix factorization and regression techniques can be used to incorporate multiple factors, such as quality or price, into the recommendation process. Deep reinforcement learning – a combination of DL and RL – is also being explored to create more adaptative and intelligent systems [52].

Other research areas include social and knowledge-based recommendations, which take into account users' social networks or semantic information [53]. These approaches can provide better personalization in cases where ratings alone are limited. Finally, to deal with privacy and ethical concerns, Aïmeur et al. proposed a privacy-preserving framework for e-commerce platforms [54]. As recommender systems are increasingly used in sensitive domains such as health, education or professional environments, there is a growing need for fairness, transparency and privacy.

6 Conclusion

Recommender systems (RS) have become essential in today's digital landscape. They are used in various domains such as streaming platforms or e-commerce, and provide personalized suggestions to users. In the era of big data, RS offer a practical way to tailor content to each user profile and prevent wasting time looking for relevant content. Recommendation techniques have evolved over the years, leading to a variety of algorithms with different strengths and limitations. Challenges such as data sparsity or the cold-start problem continue to ask questions within the field, and hybrid approaches are developed to improve the overall performance and robustness of a system. Machine learning (ML) has played a key role in the progress of RS algorithms by providing methods that enhance the quality of predictions. Today, more advanced techniques such as deep learning (DL) are employed and developed to improve the field even more.

The first part of this paper provided a literature review of collaborative filtering recommender systems (CF-RS). While not exhaustive, the review offered a useful overview for readers new in the field. The core ideas behind CF-RS were explained, with a focus on a few key ML techniques such as matrix factorization and clustering. The second part of this paper presented the experiment that was conducted on the MovieLens 1M dataset. Three techniques were implemented in Python: Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF) and KMeans. The goal of these experiments was to go beyond theory by engaging with real-world data and observing how the algorithms performed on it. Implementing a basic version of an RS enabled to see typical issues that come with large and sparse datasets and that were mentioned in the literature review. The dimensionality reduction techniques (SVD, NMF) produced reasonably good results with tuned hyperparameters. Furthermore, the KMeans experimentation showed a limited clustering performance on sparse data and also highlighted the time and scalability challenges that come with already large enough datasets such as MovieLens 1M.

Overall, this study has provided both a conceptual and practical understanding of some core methods used in recommender systems, showing the role of machine learning in shaping this evolving field.

References

[1] M. F. Aljunid, M. D.H., M. K. Hooshmand, W. A. Ali, A. M. Shetty, and S. Q. Alzoubah, "A collaborative filtering recommender systems: Survey," *Neurocomputing*, vol. 617, p. 128 718, Feb. 2025, ISSN: 09252312. DOI: 10.1016/j.neucom.2024.128718. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231224014899 (visited on 03/09/2025).

[2] G. Linden, B. Smith, and J. York, "Linden g, smith b and york j: 'amazon.com recommendations: Item-to-item collaborative filtering', internet comput. IEEE, , 7," *Internet Computing, IEEE*, vol. 7, pp. 76–80, Feb. 1, 2003. DOI: 10.1109/MIC.2003.1167344.

[3] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, 13:1–13:19, Dec. 28, 2016, ISSN: 2158-656X. DOI: 10.1145/2843948. [Online]. Available: https://dl.acm.org/doi/10.1145/2843948 (visited on 03/09/2025).

[4] D. Holtz, B. Carterette, P. Chandar, Z. Nazari, H. Cramer, and S. Aral, "The engagement-diversity connection: Evidence from a field experiment on spotify,"

[5] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, ser. UAI'98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jul. 24, 1998, pp. 43–52, ISBN: 978-1-55860-555-8. (visited on 05/04/2025).

[6] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The Adaptive Web: Methods and Strategies of Web Personalization*, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds., Berlin, Heidelberg: Springer, 2007, pp. 291–324, ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_9. [Online]. Available: https://doi.org/10.1007/978-3-540-72079-9_9 (visited on 03/09/2025).

[7] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, ser. ICUIMC '08, New York, NY, USA: Association for Computing Machinery, Jan. 31, 2008, pp. 208–211, ISBN: 978-1-59593-993-7. DOI: 10.1145/1352793.1352837. [Online]. Available: https://doi.org/10.1145/1352793.1352837 (visited on 04/01/2025).

[8] "MovieLens 1m dataset," GroupLens. (Sep. 23, 2015), [Online]. Available: https://grouplens.org/datasets/movielens/1m/ (visited on 05/05/2025).

[9] R. Chen, Q. Hua, and Y.-S. Chang. "A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks | IEEE journals & magazine | IEEE xplore." (2018), [Online]. Available: https://ieeexplore.ieee.org/document/8506344 (visited on 03/10/2025).

[10] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, Jul. 2013, ISSN: 09507051. DOI: 10.1016/j.knosys.2013.03.012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950705113001044 (visited on 03/09/2025).

[11] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, Nov. 1, 2002, ISSN: 1573-1391. DOI: 10.1023/A:1021240730564. [Online]. Available: https://doi.org/10.1023/A:1021240730564 (visited on 03/09/2025).

[12] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, Jan. 1, 2004, ISSN: 1046-8188. DOI: 10.1145/963770.963772. [Online]. Available: https://doi.org/10.1145/963770.963772 (visited on 03/09/2025).

[13] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, no. 1, p. 421 425, 2009, ISSN: 1687-7489. DOI: 10.1155/2009/421425. (visited on 03/10/2025).

[14] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009, Conference Name: Computer, ISSN: 1558-0814. DOI: 10.1109/MC.2009.263. [Online]. Available: https://ieeexplore.ieee.org/document/5197422 (visited on 03/09/2025).

[15] A. Kohrs and B. Merialdo. "Clustering for collaborative filtering applications | EURECOM." (), [Online]. Available: https://www.eurecom.fr/en/publication/419 (visited on 04/03/2025).

[16] T. Mitchell, *Machine learning*, New York: McGraw-hill., 9 vols. 1997, vol. 1.

[17] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, M. Cord and P. Cunningham, Eds., Berlin, Heidelberg: Springer, 2008, pp. 21–49, ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_2. [Online]. Available: https://doi.org/10.1007/978-3-540-75171-7_2 (visited on 05/08/2025).

[18] R. Raina, Y. Shen, A. McCallum, and A. Ng, "Classification with hybrid generative/discriminative models," in *Advances in Neural Information Processing Systems*, vol. 16, MIT Press, 2003. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2003/hash/b53477c2821c1bf0da5d40e57b870d35-Abstract.html (visited on 05/08/2025).

[19] D. Berrar, "Cross-validation," in Jan. 1, 2018, ISBN: 978-0-12-809633-8. DOI: 10.1016/B978-0-12-809633-8.20349-X.

[20] I. Nti, O. Nyarko-Boateng, and J. Aning, "Performance of machine learning algorithms with different k values in k-fold cross-validation," *International Journal of Information Technology and Computer Science*, vol. 6, pp. 61–71, Dec. 12, 2021. DOI: 10.5815/ijitcs.2021.06.05.

[21] E. Elgeldawi, A. Sayed, A. R. Galal, and A. M. Zaki, "Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis," *Informatics*, vol. 8, no. 4, p. 79, Dec. 2021, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2227-9709. DOI: 10.3390/informatics8040079. [Online]. Available: https://www.mdpi.com/2227-9709/8/4/79 (visited on 05/08/2025).

[22] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems," *Physics Reports*, Recommender Systems, vol. 519, no. 1, pp. 1–49, Oct. 1, 2012, ISSN: 0370-1573. DOI: 10.1016/j.physrep.2012.02.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157312000828 (visited on 03/10/2025).

[23] S. Shafiee, "Unveiling the latest trends and advancements in machine learning algorithms for recommender systems: A literature review," *Procedia CIRP*, vol. 121, pp. 115–120, 2024, ISSN: 22128271. DOI: 10.1016/j.procir.2023.08.062. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2212827123009575 (visited on 03/09/2025).

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system - a case study:" Defense Technical Information Center, Fort Belvoir, VA, Jul. 14, 2000. DOI: 10.21236/ADA439541. [Online]. Available: https://apps.dtic.mil/sti/citations/tr/ADA439541 (visited on 03/10/2025).

[25] R. Mehta and K. Rana, "A review on matrix factorization techniques in recommender systems," in *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, Apr. 2017, pp. 269–274. DOI: 10.1109/CSCITA.2017.8066567. [Online]. Available: https://ieeexplore.ieee.org/document/8066567 (visited on 04/03/2025).

[26] H.-J. Xue, X.-Y. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, ser. IJCAI'17, Melbourne, Australia: AAAI Press, Aug. 19, 2017, pp. 3203–3209, isbn: 978-0-9992411-0-3. (visited on 04/03/2025).

[27] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems,"

[28] M. Brand, "Incremental singular value decomposition of uncertain data with missing values," in *Computer Vision — ECCV 2002*, A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., Berlin, Heidelberg: Springer, 2002, pp. 707–720, isbn: 978-3-540-47969-7. doi: 10.1007/3-540-47969-4_47.

[29] X. Zhou, J. He, G. Huang, and Y. Zhang, "SVD-based incremental approaches for recommender systems," *Journal of Computer and System Sciences*, vol. 81, no. 4, pp. 717–733, Jun. 1, 2015, issn: 0022-0000. doi: 10.1016/j.jcss.2014.11.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000014001706 (visited on 04/03/2025).

[30] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999, Publisher: Nature Publishing Group, issn: 1476-4687. doi: 10.1038/44565. [Online]. Available: https://www.nature.com/articles/44565 (visited on 05/05/2025).

[31] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, May 2014, Conference Name: IEEE Transactions on Industrial Informatics, issn: 1941-0050. doi: 10.1109/TII.2014.2308433. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6748996?casa_token=9jN5Vt7NoZAAAAAA:-c4QXuOJh3twgU7Z_JYBJQvt9abPFyT6OuQmKvltwVtI-zBOc2iragwrtxD7b6htdQ8yZrxU_rJe (visited on 04/03/2025).

[32] G. Pitsilis, X. Zhang, and W. Wang, "Clustering recommenders in collaborative filtering using explicit trust information," in *Trust Management V*, I. Wakeman, E. Gudes, C. D. Jensen, and J. Crampton, Eds., Berlin, Heidelberg: Springer, 2011, pp. 82–97, isbn: 978-3-642-22200-9. doi: 10.1007/978-3-642-22200-9_9.

[33] Z. Wang, X. Yu, N. Feng, and Z. Wang, "An improved collaborative movie recommendation system using computational intelligence," *Journal of Visual Languages & Computing*, Distributed Multimedia Systems DMS2014 Part I, vol. 25, no. 6, pp. 667–675, Dec. 1, 2014, issn: 1045-926X. doi: 10.1016/j.jvlc.2014.09.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1045926X14000901 (visited on 04/02/2025).

[34] Q. Li and B. Kim, *Clustering approach for hybrid recommender system.* Nov. 13, 2003, 33 pp., Journal Abbreviation: Proceedings of the International Conference on Web Intelligence Pages: 38 Publication Title: Proceedings of the International Conference on Web Intelligence, isbn: 978-0-7695-1932-6. doi: 10.1109/WI.2003.1241167.

[35] K.-j. Kim and H. Ahn, "A recommender system using GA K-means clustering in an online shopping market," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1200–1209, Feb. 1, 2008, issn: 0957-4174. doi: 10.1016/j.eswa.2006.12.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417406004076 (visited on 04/03/2025).

[36] N. Boyko. "(PDF) hierarchical clustering algorithm for dendrogram construction and cluster counting." (2023), [Online]. Available: https://www.researchgate.net/publication/373146750_Hierarchical_clustering_algorithm_for_dendrogram_construction_and_cluster_counting (visited on 05/05/2025).

[37] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model,"

[38] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, Jul. 1, 2001, ISSN: 1573-7659. DOI: 10.1023/A:1011419012209. [Online]. Available: https://doi.org/10.1023/A:1011419012209 (visited on 04/02/2025).

[39] F. Hernández del Olmo and E. Gaudioso, "Evaluation of recommender systems: A new approach," *Expert Systems with Applications*, vol. 35, no. 3, pp. 790–804, Oct. 1, 2008, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2007.07.047. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417407002928 (visited on 03/26/2025).

[40] N. Hurley and M. Zhang, "Novelty and diversity in top-n recommendation – analysis and evaluation," *ACM Trans. Internet Techn.*, vol. 10, p. 14, Mar. 1, 2011. DOI: 10.1145/1944339.1944341.

[41] S.-T. Park and W. Chu, "Pairwise preference regression for cold-start recommendation," in *Proceedings of the third ACM conference on Recommender systems*, ser. RecSys '09, New York, NY, USA: Association for Computing Machinery, Oct. 23, 2009, pp. 21–28, ISBN: 978-1-60558-435-5. DOI: 10.1145/1639714.1639720. [Online]. Available: https://doi.org/10.1145/1639714.1639720 (visited on 04/01/2025).

[42] "MovieLens," GroupLens. (Sep. 6, 2013), [Online]. Available: https://grouplens.org/datasets/movielens/ (visited on 05/05/2025).

[43] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: Applying collaborative filtering to usenet news," *Commun. ACM*, vol. 40, no. 3, pp. 77–87, Mar. 1, 1997, ISSN: 0001-0782. DOI: 10.1145/245108.245126. [Online]. Available: https://dl.acm.org/doi/10.1145/245108.245126 (visited on 05/05/2025).

[44] R. Salakhutdinov and A. Mnih. "Bayesian probabilistic matrix factorization using markov chain monte carlo | proceedings of the 25th international conference on machine learning." (), [Online]. Available: https://dl.acm.org/doi/10.1145/1390156.1390267 (visited on 05/05/2025).

[45] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, *BPR: Bayesian personalized ranking from implicit feedback*, May 9, 2012. DOI: 10.48550/arXiv.1205.2618. arXiv: 1205.2618[cs]. [Online]. Available: http://arxiv.org/abs/1205.2618 (visited on 05/05/2025).

[46] L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, and D. Yin, "Reinforcement learning to optimize long-term user engagement in recommender systems," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage AK USA: ACM, Jul. 25, 2019, pp. 2810–2818, ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330668. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330668 (visited on 03/09/2025).

[47] S. Giannikis, F. Frasincar, and D. Boekestijn, "Reinforcement learning for addressing the cold-user problem in recommender systems," *Knowledge-Based Systems*, vol. 294, p. 111 752, Jun. 21, 2024, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2024.111752. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705124003873 (visited on 05/06/2025).

[48] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys*, vol. 52, no. 1, pp. 1–38, Jan. 31, 2020, ISSN: 0360-0300, 1557-7341. DOI: 10 . 1145 / 3285029. arXiv: 1707 . 07435[cs]. [Online]. Available: http://arxiv.org/abs/1707.07435 (visited on 05/07/2025).

[49] R. Mu, X. Zeng, and L. Han, "A survey of recommender systems based on deep learning," *IEEE Access*, vol. PP, pp. 1–1, Nov. 9, 2018. DOI: 10.1109/ACCESS.2018.2880197.

[50] S. Gheewala, S. Xu, and S. Yeom, "In-depth survey: Deep learning in recommender systems—exploring prediction and ranking models, datasets, feature analysis, and emerging trends," *Neural Computing and Applications*, Mar. 21, 2025, ISSN: 1433-3058. DOI: 10.1007/s00521-024-10866-z. [Online]. Available: https://doi.org/10.1007/s00521-024-10866-z (visited on 05/07/2025).

[51] R. Singh, P. Dwivedi, and P. Patidar. "Multi-criteria recommendation system based on deep matrix factorization and regression techniques | international journal of information technology." (2024), [Online]. Available: https://link.springer.com/article/10.1007/s41870-024-01780-7 (visited on 05/06/2025).

[52] X. Chen, L. Yao, J. McAuley, G. Zhou, and X. Wang, "Deep reinforcement learning in recommender systems: A survey and new perspectives," *Knowledge-Based Systems*, vol. 264, p. 110 335, Mar. 15, 2023, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2023.110335. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705123000850 (visited on 05/07/2025).

[53] W. Carrer-Neto, M. L. Hernández-Alcaraz, R. Valencia-García, and F. García-Sánchez, "Social knowledge-based recommender system. application to the movies domain," *Expert Systems with Applications*, vol. 39, no. 12, pp. 10 990–11 000, Sep. 15, 2012, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2012.03.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417412004952 (visited on 03/09/2025).

[54] E. Aïmeur, G. Brassard, J. M. Fernandez, and F. S. Mani Onana, "Alambic: A privacy-preserving recommender system for electronic commerce," *International Journal of Information Security*, vol. 7, no. 5, pp. 307–334, Oct. 1, 2008, ISSN: 1615-5270. DOI: 10.1007/s10207-007-0049-3. [Online]. Available: https://doi.org/10.1007/s10207-007-0049-3 (visited on 05/06/2025).