Information Fusion 117 (2025) 102905

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

A survey of latent factor models in recommender systems

Hind I. Alshbanat, Hafida Benhidour *, Said Kerrache

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh, 11543, Saudi Arabia

ARTICLE INFO

Keywords:
Personalized recommendations
Implicit feedback
Trust data
Nonlinear models
Deep neural networks
Self-supervised learning
Transfer learning
Optimization techniques
Stochastic gradient descent
Data sparsity
Scalability

ABSTRACT

Recommender systems are essential tools in the digital era, providing personalized content to users across various domains, including e-commerce, entertainment, and social media. Among the numerous approaches developed for these systems, latent factor models have proven to be particularly effective. This survey systematically reviews latent factor models in recommender systems, highlighting their core principles, methodologies, and recent advancements. The literature is analyzed through a structured framework that organizes prior work into a well-defined taxonomy based on four key aspects where advancements in latent factor models have occurred: learning data, model architecture, learning strategies, and optimization techniques. The analysis includes a taxonomy of contributions and detailed discussions on the types of learning data used, such as implicit feedback, trust data, and content data. Additionally, it explores various models, including probabilistic, nonlinear, and neural models, and examines diverse learning strategies like online learning, transfer learning, and active learning. Furthermore, the survey addresses the optimization strategies employed to train latent factor models, which enhance their performance and scalability. By identifying trends, gaps, and potential research directions, this survey aims to provide valuable insights for researchers and practitioners seeking to advance the field of recommender systems.

1. Introduction

Recommender systems are indispensable in the digital age. They play a crucial role in filtering large amounts of data to provide personalized content to users. These systems assist users in making informed choices across various domains, such as product selection, movie preferences, and music discovery. Recommender systems boost user satisfaction and engagement by forecasting user preferences using historical data and contextual information.

Latent factor models have emerged as a highly effective framework for constructing recommender systems, offering a versatile approach to capturing user–item interactions. Early examples include methods based on matrix factorization, such as Singular Value Decomposition (SVD), which laid the foundation for this class of models. Over time, latent factor models have evolved beyond basic matrix factorization techniques, incorporating advanced methodologies to address key challenges such as data sparsity and scalability. By representing users and items within a shared latent space, these models effectively uncover underlying patterns in recommendation tasks.

To better understand the inner workings of latent factor models in recommender systems, we present a general structure in Fig. 1. This structure highlights several key components that are essential for the effective operation of these systems:

- **Learning Data**: This includes the input data for training the model. Learning data forms the foundation of the recommender system and typically consists of user–item interaction records. These records can include explicit feedback, such as ratings, or implicit feedback, such as clicks, views, and purchase history. The quality and quantity of this data significantly influence the performance of the recommender system.
- **Model**: The structure that encodes user–item interactions. The model captures the relationships between users and items, translating the complex patterns in the interaction data into a mathematical representation. This representation allows the system to predict user preferences for unseen items.
- **Learning Strategy**: The approach used to train the model. The learning strategy defines how the model is exposed to the data, how it learns from it, and how it iteratively improves its performance. It includes decisions on the type of learning (e.g., batch, online) and specific techniques that guide the model training process to ensure effective learning from the data.
- **Optimization**: The process of tuning the model parameters to minimize prediction error. Optimization involves adjusting the model's parameters to improve its accuracy in predicting user preferences. This process uses various algorithms to find the

* Corresponding author.
E-mail address: hbenhidour@ksu.edu.sa (H. Benhidour).

https://doi.org/10.1016/j.inffus.2024.102905
Received 19 July 2024; Received in revised form 21 December 2024; Accepted 24 December 2024
Available online 31 December 2024

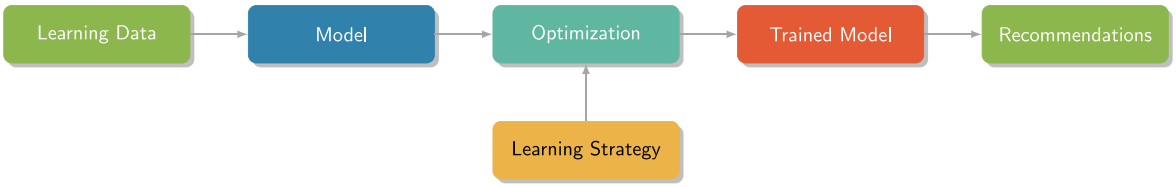

Fig. 1. The structure of a latent factor model-based recommender system.

Table 1
Comparison of recent surveys on recommender systems.

Ref.	Year	Title	Difference with the current survey
[3]	2023	Recent developments in recommender systems: A survey	Covers recent developments across various recommender system models, providing a broad perspective compared to the present survey, which offers a focused and extensive exploration of latent factor models and their advancements.
[4]	2023	Multimodal recommender systems: A survey	Concentrates on multimodal data in recommender systems, whereas the current survey focuses on latent factor models.
[5]	2023	Context-aware recommender systems and cultural heritage: A survey	Focuses on context-aware systems and cultural heritage applications, different from the latent factor model focus.
[6]	2022	Graph Neural Networks in Recommender Systems: A Survey	Surveys graph neural networks in recommender systems, whereas the current survey is about latent factor models.
[7]	2023	Self-Supervised Learning for Recommender Systems: A Survey	Focuses on self-supervised learning techniques in recommender systems, unlike the current survey on latent factor models.

best parameter values that minimize the difference between the predicted and actual user interactions.

- **Trained Model**: The output of the optimization process, which can generate recommendations. Once the model parameters are optimized, the trained model can produce accurate and personalized recommendations based on the learned patterns in the data.
- **Recommendations**: The trained model generates personalized recommendations tailored to the user's preferences and needs

Given the complexity and rapid advancements in recommender system technologies, a comprehensive survey is essential to summarize current knowledge. While prior surveys have explored areas like integrating deep learning with latent factor models [1] or deep versions of latent factor models [2], this survey provides a holistic view of the role of latent factor models in recommender systems (see Table 1).

This survey examines the literature through a structured framework considering four key axes of advancement: learning data, model architecture, learning strategies, and optimization techniques. It addresses the need for a unified methodology for categorizing contributions across different applications and datasets, offering a taxonomy that organizes contributions into four axes, highlighting individual papers' strengths and identifying trends and research gaps. This taxonomy of contributions is not mutually exclusive, and some papers may offer advancements in multiple categories. For readability, however, we present each paper in the category with the most substantial contribution.

The main contributions of this survey are as follows:

1. **Systematic Review**: It provides a systematic review of latent factor models by organizing prior work into a well-defined taxonomy based on four key aspects where advancements in latent factor models have been made.
2. **Comprehensive Analysis**: It highlights strengths, identifies trends, and points out gaps in current research. Additionally, we surveyed a large number of papers, focusing not only on the models but also on other key aspects of latent factor models.
3. **Research Trends and Future Directions**: It addresses potential research trends and future directions related to latent factor models.

As shown in Fig. 2, the rest of the paper is divided into two parts. Sections 2 and 3 cover background material on recommender systems and latent factor models. Section 2 covers the fundamentals of recommender systems, providing an overview of collaborative filtering, content-based filtering, and hybrid systems. It discusses the strengths and weaknesses of each approach and their application scenarios. Section 3 introduces latent factor models, explaining their basic principles, introducing mathematical notation, and presenting the fundamental algorithms used to develop and optimize these models for recommender systems.

The second part of the paper, from Sections 4 to 7, contains the surveyed methods. Section 4 discusses the various types of learning data used to improve the quality of recommendations, including implicit feedback, trust, and content data. Section 5 discusses various models used in latent factor approaches, including probabilistic, weighted, kernelized, and nonlinear models. It examines how these models enhance the capability and accuracy of recommender systems. Section 6 reviews learning strategies like online, transfer, and active learning, highlighting their applications and benefits in improving the adaptability and effectiveness of recommender systems. Section 7 focuses on optimization strategies for latent factor models. It covers general-purpose algorithms such as stochastic gradient descent, its various variants, and more specialized algorithms dedicated to latent factor models in recommender systems.

Each survey section concludes with an in-depth discussion of existing contributions and highlights interesting future research directions. Section 8 consolidates the findings and future research directions, including a functional comparison of the surveyed methods. By examining state-of-the-art approaches and identifying gaps in current research, we aim to provide insights into potential areas for future work and guide researchers in advancing the field of recommender systems. Finally, Section 9 concludes the survey.

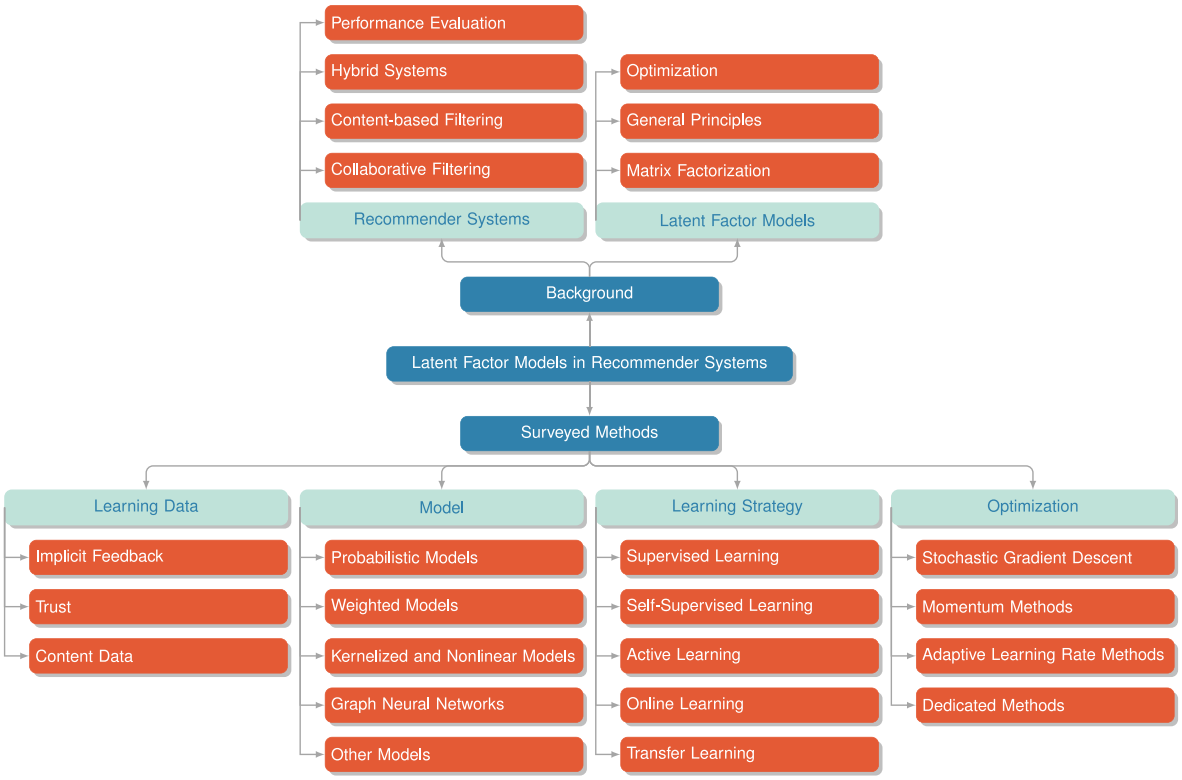

Fig. 2. Manuscript organization.

2. Fundamentals of recommender systems

Recommender systems filter data to provide relevant information to users [8]. Despite the large amount of online information, there are typically few rated items due to users being unwilling to provide ratings. As a result, recommendation data are typically highly sparse. The recommender systems' task is to predict unrated items based on a small portion of rated items. The system takes as input a user model (such as ratings, preferences, situational context, demographics) and items with or without a description and finds the corresponding relevance scores [9]. Finally, it returns a list of recommended items relevant to the target user.

Several approaches have been proposed to solve the recommendation problem. These approaches are mainly organized into three classes [9,10]:

- Collaborative filtering recommender systems are based on user profiles and ratings of similar users and recommend items to users based on what other users with similar tastes have previously preferred.
- Content-based recommender systems use user profiles and item descriptions to recommend items similar to those the users liked in the past.
- Hybrid recommender systems combine collaborative filtering and content-based approaches to improve recommendation accuracy and diversity.

The rest of this section will present each category, detailing the most common and basic algorithms associated with each and their respective advantages and disadvantages.

2.1. Collaborative filtering

Collaborative filtering recommender systems are the most familiar and widely used personalized recommendation algorithms. It is based on the premise that users with similar tastes in the past tend to like similar items in the future [11]. These systems first detect user similarities based on historical data and then use these similarities to recommend new items. Collaborative filtering is primarily based on ratings, whether explicit (e.g., numeric, thumbs up or down) or implicit (e.g., clicking, watching). Collaborative filtering is divided into two categories: memory-based and model-based collaborative filtering [12]. Memory-based collaborative filtering requires a complete user–item database in memory for computing recommendations. In contrast, model-based collaborative filtering requires only an abstract representation of such a database.

2.1.1. Memory-based collaborative filtering

Memory-based collaborative filtering uses similarity measures and various prediction computation techniques to estimate unknown ratings. In general, memory-based collaborative filtering algorithms work as follows:

1. Build a user–item rating matrix where rows correspond to users and columns correspond to items. An entry r_{ui} in this matrix contains the rating assigned to item i by user u.
2. Calculate the similarity row-wise between users or column-wise between items using a prescribed similarity measure.
3. Select the k users or items most similar to the target user or item.
4. Predict unknown ratings using a specified prediction calculation method.

Memory-based collaborative filtering is divided into user-based and item-based collaborative filtering [13,14] depending on whether the similarity computation is conducted on users or items. The idea of user-based collaborative filtering algorithms is that users who have rated the same items in the past have similar interests. Thus, the system infers a target user's future rating based on ratings from similar users. Several similarity measures can be used to compute the similarity between the users, including cosine similarity [15–18], Pearson's Correlation

Coefficient (PCC) [19,20], Spearman's correlation coefficient [21], Adjusted cosine similarity [22], and the Jaccard coefficient [23]. Within this plethora of measures, cosine similarity and the Pearson correlation coefficient are the most widely used in practice. Cosine similarity considers the user's ratings as an n-dimensional vector and computes user similarity through the angles between vectors:

$$sim_{uv} = \cos(r_u, r_v) = \frac{r_u \cdot r_v}{\|r_u\|_2 \|r_v\|_2} = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2} \sqrt{\sum_{i \in I_v} r_{vi}^2}}, \quad (1)$$

where sim_{uv} represents the similarity between user u and user v, r_u and r_v represent the rating vectors of users u and v, $\|\cdot\|_2$ denotes the L_2 norm, r_{ui} and r_{vi} represent ratings of users u and v on item i, I_u and I_v are the set of items already rated by user u and user v, respectively, and I_{uv} stands for the shared items rated by both users.

Pearson's correlation coefficient, on the other hand, measures the similarity between users based on the linear correlation between their shared ratings:

$$sim_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)}}, \quad (2)$$

where $\bar{r}_u$ and $\bar{r}_v$ denote the average ratings user u and v respectively.

Once the k nearest neighbor or the highest similar users to the target user have been selected, user-based collaborative filtering then predicts unknown ratings using the following formula [24]:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_u} sim_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in N_u} |sim_{uv}|}, \quad (3)$$

where N_u is the set of nearest neighbors of user u. Eq. (3) states that the predicted difference between the rating given to item i and the average rating $\bar{r}_u$ of user u is the weighted sum of the deviations of the ratings given to the same item by the k nearest neighbors of u. The absolute value is necessary here since some similarity measures may take a negative sign.

Item-based collaborative filtering computes the similarity between items rather than users. For example, Eq. (4) shows how to calculate the similarity between items using the Pearson correlation coefficient [25]:

$$sim_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}}, \quad (4)$$

where sim_{ij} is the similarity of item i and item j, U_i and U_j represent the set of users who rated item i and item j, respectively, U_{ij} is the set of users who rated both items, and $\bar{r}_u$ and $\bar{r}_j$ are the average ratings of item i and item j, respectively.

2.1.2. Model-based collaborative filtering

Memory-based recommender systems are easy to understand and implement but unsuitable for large datasets due to their large storage requirements and poor computational efficiency and scalability. Model-based collaborative filtering avoids this drawback by building and learning a model based on the user–item rating matrix and then predicting unknown ratings [26,27]. Model-based collaborative filtering approaches can be classified into latent factor models [28–32], clustering models [33], Bayesian classifiers [34], and various probabilistic relational models [35]. In this survey, we will focus on latent factor models because of their remarkable success in solving the recommendation problem, particularly their ability to handle sparsity and capture underlying patterns in the rating data.

2.2. Content-based filtering

Content-based recommender systems provide recommendations to target users by comparing the representation of the description of items with the representation of the content of interest to the target user [36,37]. For example, if the user prefers action movies, the system learns to recommend other movies in the action genre. Content-based recommender systems can also extract information from the user's profile, such as gender, age, nationality, and demographic information, to improve recommendations [38,39]. This information can prove valuable to the recommendation algorithm as it helps build a model of user preferences for their profile data.

A content-based system analyzes item descriptions and user profiles using text mining and natural language processing to extract meaningful features. A model is trained to capture similarity between items or between items and user preferences. The system ranks items by relevance, calculated by the learned model, and suggests top recommendations.

Content-based recommender systems can effectively recommend items that users will like, even if those items are new or less popular. However, they tend to recommend items similar to those the user has already interacted with, which can result in a lack of serendipity in the recommendations. Their performance also depends heavily on the availability of informative, accurate item descriptions. However, these descriptions are often prone to human error, imperfections, or missing information, which can profoundly impact the quality of the recommendations.

2.3. Hybrid systems

To overcome the shortcomings of a single recommendation model, collaborative filtering and content-based algorithms can be combined in hybrid systems to improve the prediction accuracy of the recommender system [30,40–42]. Combining collaborative and content-based algorithms into a hybrid recommender system can be achieved in various ways. One approach involves implementing each model separately and then aggregating their predictions to leverage the strengths of the base models. Another strategy incorporates characteristics of content-based recommendations into collaborative filtering or vice versa, resulting in a system that benefits from the unique advantages of both methods. A more integrated approach also involves building a fused model that inherently includes collaborative filtering and content-based characteristics.

2.4. Performance evaluation

The type of performance measures used to evaluate a recommender system depends on the query the system is designed to answer. The most common use case requires the system to predict a numerical rating given by a user to a specific item. The task is then considered a regression task, and usual regression performance measures are used. These typically include the Mean Absolute Error (MAE) [12] and the Root Mean Square Error (RMSE) [39]. MAE computes the average absolute difference between predicted and actual ratings:

$$MAE = \frac{\sum_{r_{ui} \in \mathcal{T}} |r_{ui} - \hat{r}_{ui}|}{|\mathcal{T}|}, \quad (5)$$

where $\mathcal{T}$ is the test set, $|\mathcal{T}|$ represents the total number of ratings in the test set, r_{ui} and $\hat{r}_{ui}$ represent the actual and predicted ratings for user u and item i, respectively. RMSE computes the square root of the average of the squared differences between predicted and actual ratings:

$$RMSE = \sqrt{\frac{\sum_{r_{ui} \in \mathcal{T}} (r_{ui} - \hat{r}_{ui})^2}{|\mathcal{T}|}}. \quad (6)$$

Lower values of MAE and RMSE indicate better performance by the model. These two measures are generally positively correlated, but MAE has the property of treating all error magnitudes as equal. In contrast, RMSE, on the other hand, penalizes large errors more than small ones. For example, with MAE, an error of 2 in a single example is equivalent to two errors of 1 in two different examples, whereas with RMSE, an error of 2 in a single example is considered more severe than two errors of 1 in two different examples.

A recommender system can also be used to discriminate between items relevant to a given user and those irrelevant rather than producing a numerical rating. This scenario is a classification task that can be evaluated using binary classification performance measures such as accuracy, recall, precision, and the F1 measure.

In various practical scenarios, such as online shopping and reservation websites, recommender systems sort the available items in descending order of relevance and present them to the user. Having the most relevant items at the top of the list is desirable. Ranking metrics are used to evaluate the performance of this type of system [43], which include Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Discounted Cumulative Gain (DCG), Normalized Discounted Cumulative Gain (NDCG), and Hit Ratio (HR). MAP is commonly used to evaluate such recommender systems. MAP is found by taking the mean of all users' Average Precision (AP). The AP for a single user is calculated as the average precision at k for all k corresponding to relevant items within the recommendation list. This process captures both the precision of the recommendations and their ranking quality. MRR focuses on the rank of the most relevant item. The higher its rank, the closer the MRR is to its maximum of 1. DCG measures the overall quality of the recommendation list by giving a higher weight to relevant items placed at the top. In contrast, NDCG normalizes DCG against the ideal ranking order, providing a scale from 0 to 1, where 1 represents the perfect order. HR is usually reported at a specific cut-off point, HR@k, indicating if the relevant item is within the top k recommendations.

3. Fundamentals of latent factor models

Latent factor models are among the most successful model-based techniques for recommender systems [44]. These models often focus on factorizing the user–item rating matrix into two lower-dimensional latent spaces. The latent factors are derived from the observed ratings using well-established linear algebra or optimization methods. In some models, a reconstruction function, such as the inner product of the user and item latent vectors, is then used to predict unknown ratings. For other models, such as those based on neural networks, the reconstruction function is learned from data. This section covers the fundamentals of latent factor models. We begin by discussing the precursors to most modern models and then provide a general formulation of them. We conclude by highlighting the key optimization techniques used to fit these models to data.

3.1. Matrix factorization: A basic latent factor model

Singular Value Decomposition (SVD) [45,46] is one of the earliest latent factor approaches for collaborative filtering. SVD is a classical linear algebra technique that generalizes eigenvalue decomposition to non-square matrices. More precisely, any $m \times n$ matrix R can be decomposed as:

$$R = U \Sigma V^T, \tag{7}$$

where U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ rectangular diagonal matrix (with all non-diagonal entries set to zero), and V^T is the transpose of an $n \times n$ orthogonal matrix. The diagonal entries $\sigma_i = \Sigma_{ii}$ are known as the *singular values* of R and are guaranteed to be non-negative for a real matrix R. In collaborative filtering, R represents the rating matrix, U encapsulates user latent vectors, and V contains item latent vectors. The magnitude of the singular values indicates each factor's contribution strength in explaining the observed ratings. SVD requires fully known matrices and, therefore, cannot be directly applied to rating matrices, as they typically contain missing values. This issue can be addressed through matrix imputation by filling in missing values with the mean rating of either the user or item (mean imputation) or setting them to zero (zero imputation). Nonetheless, applying SVD directly to an imputed matrix is not very useful, as the predicted rating will exactly equal the imputed values, causing the model to fail in generalization. Instead, we adopt the *low-rank assumption*, which posits that the full high-dimensional matrix R can be approximated by using a small number of factors that capture the data's essential structure and main patterns. More specifically, we retain the factors corresponding to the k largest singular values and discard the rest, leading to the following approximation:

$$R \approx \tilde{R} = \tilde{U} \tilde{\Sigma} \tilde{V}^T, \tag{8}$$

where $\tilde{U}$ is $m \times k$, $\tilde{\Sigma}$ is $k \times k$, and $\tilde{V}^T$ is $k \times n$. Fig. 3 illustrates the SVD procedure for rating prediction.

This approach to dimensionality reduction helps mitigate overfitting, enabling SVD to predict unseen ratings using a small number of factors. However, SVD faces several challenges. Primarily, rating matrices are typically sparse, and imputation introduces biases. For example, mean imputation can significantly reduce the ratings' variance, while zero imputation shifts the average rating towards lower values. Excessive imputation in highly sparse matrices also obscures existing patterns, resulting in poor generalization. Additionally, SVD incurs a high computational overhead from factoring the often high-dimensional rating matrix. These issues are addressed by modern latent factor models using various regularization and optimization techniques.

In response to SVD's limitations, Matrix Factorization (MF) [28] is a refined approach, optimizing the decomposition process for sparsity and computational efficiency. MF is a latent factor model that consists of factorizing the original user–item rating matrix R into two lower-dimensional matrices P and Q:

$$R \approx \tilde{R} = PQ^T, \tag{9}$$

where P is an $n \times k$ matrix representing the latent factors of n users, Q is an $m \times k$ matrix representing the latent factors of m items, and k is the dimensionality of the latent factor space. Each row vector p_u of the matrix P constitutes the latent factor representation of user u, whereas each row vector q_i of Q represents item i. The unknown ratings are predicted by computing the dot product of the user and item latent vectors:

$$\hat{r}_{ui} = \sum_{j=1}^{k} P_{uj} Q_{ji} = p_u^T q_i, \tag{10}$$

where $\hat{r}_{ui}$ refers to the predicted rating given by user u to item i.

Note that when considered individually, p_u and q_i are treated as column vectors. To find p_u and q_i, the model is trained using a gradient descent algorithm, which minimizes the sum of squared errors (SSE) iteratively between predicted and observed ratings using the following function:

$$SSE = \frac{1}{2} \sum_{r_{ui} \in \mathcal{R}} e_{ui}^2 = \frac{1}{2} \sum_{r_{ui} \in \mathcal{R}} \left(\hat{r}_{ui} - r_{ui}\right)^2 = \frac{1}{2} \sum_{r_{ui} \in \mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2, \tag{11}$$

where $\mathcal{R}$ is the set of known ratings, and e_{ui} is the error between predicted and known ratings. At each iteration, the user and item vectors p_u and q_i are updated as follows:

$$p_u = p_u + \eta \sum_{r_{ui} \in \mathcal{R}_u} e_{ui} q_i, \tag{12}$$

$$q_i = q_i + \eta \sum_{r_{ui} \in \mathcal{R}_i} e_{ui} p_u, \tag{13}$$

where $\mathcal{R}_u$ is the set of all ratings made by user u, $\mathcal{R}_i$ is the set of all ratings of item i, and η, the learning rate, is an algorithm parameter determined empirically.

For sparse data, training the model using the loss defined in Eq. (11) may result in overfitting. Hence, the need to use L_2-norm regularization to reduce the model complexity by penalizing the norm of its parameters [47]. The regularized objective function takes the form:

$$J_{MF} = \frac{1}{2} \sum_{r_{ui} \in \mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2 + \lambda \left(\sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2 \right), \tag{14}$$

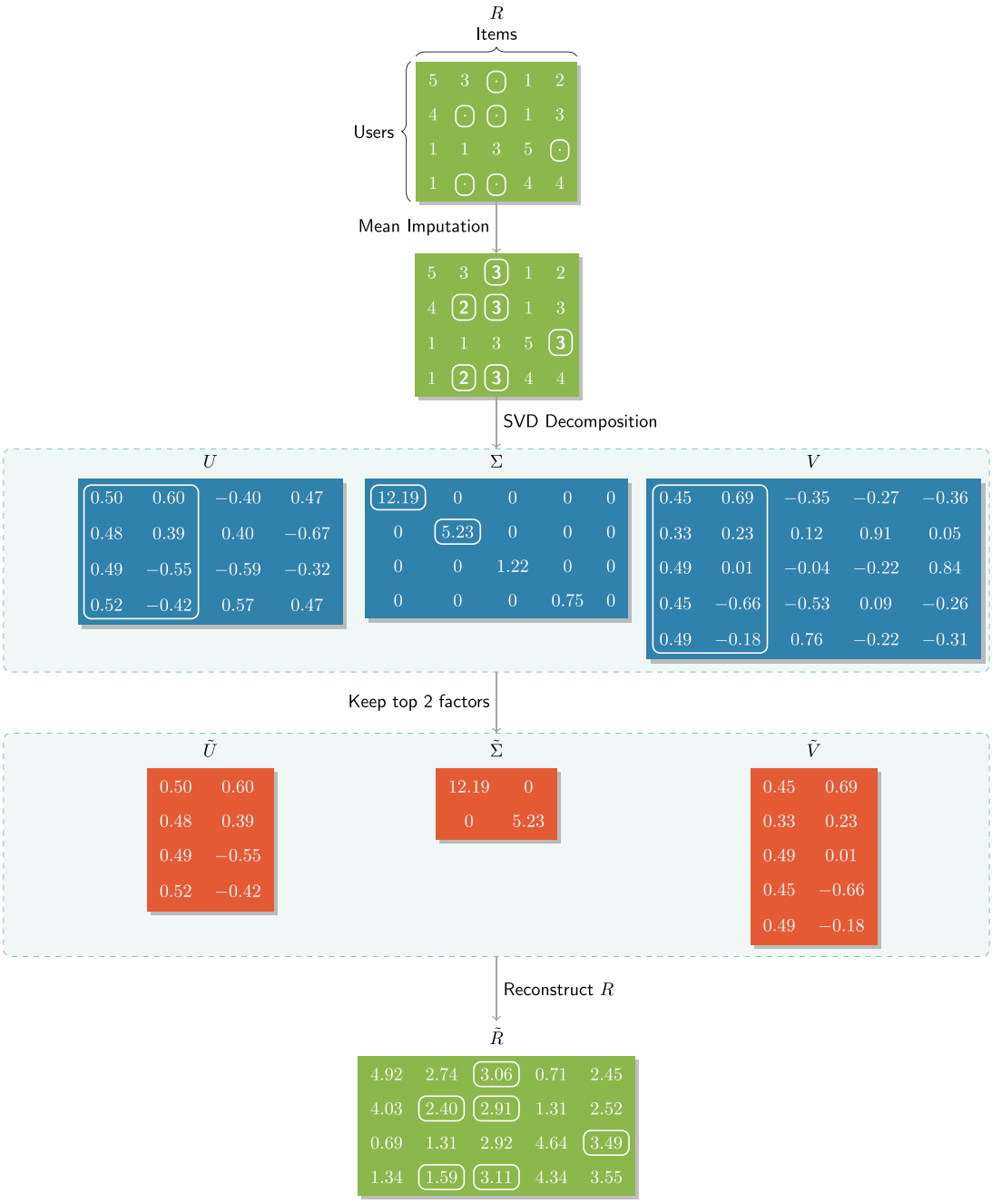

Fig. 3. An example of using SVD to predict ratings. Initially, the rating matrix R undergoes imputation, where missing ratings are replaced with the average rating of each item. Subsequently, R is decomposed using SVD. The two factors corresponding to the largest singular values are retained to reconstruct an approximation $\tilde{R}$ of R.

where λ is the regularization coefficient, a hyper-parameter typically set by model selection techniques, $\|p_u\|_2^2 = \sum_{j=1}^{k} p_{uj}^2$, and $\|q_i\|_2^2 = \sum_{j=1}^{k} q_{ij}^2$.

Matrix factorization forms the basis for various latent factor approaches in recommendation systems. These methods extend the base model by incorporating additional information, such as implicit feedback [48], trust data [49,50], or user and item weights [51]. These extensions enhance the ability to capture complex relationships and improve the accuracy of the recommendation. The details of these advanced approaches will be discussed in subsequent sections.

3.2. General principles

Matrix Factorization constitutes a foundational framework applicable to many latent factor models. Generally, a latent factor model assumes that a set of latent factors can represent user preferences and item attributes. These factors enable the explanation of existing ratings and the prediction of future ones. Specifically, a latent factor model in the context of a recommender system with n users and m items can be defined as a tuple (P, Q, w, f), where:

- $P = \{p_1, \ldots, p_n\}$ where $p_u \in \mathbb{R}^{k_u}$ represents the latent vector associated with user i, and k_u is the dimension of the latent user space.
- $Q = \{q_1, \ldots, q_m\}$ where $q_i \in \mathbb{R}^{k_i}$ represents the latent vector associated with item i, and k_i is the dimension of the latent item space.
- $w \in \mathbb{R}^d$ represents the model parameters shared among all users and items.

- f is the reconstruction function used to predict ratings:

$$\hat{r}_{ui} = f(p_u, q_i, w). \tag{15}$$

The reconstruction function f can be as simple as a dot product, as in MF, or a complex nonlinear function represented by a deep neural network.

For instance, in MF, the dimensions of the latent user space and the latent item space are equal ($k_u = k_i = k$); the model has no shared parameters, and the reconstruction function is given by Eq. (10).

The model parameters, P, Q, and w, are determined by minimizing a loss function $\ell(\hat{r}_{ui}, r_{ui})$ that captures the error between the predicted rating $\hat{r}_{ui}$ and the actual rating r_{ui}. A popular choice for ℓ with numerical ratings is the squared error due to its nice analytical and numerical properties:

$$\ell(\hat{r}_{ui}, r_{ui}) = \frac{1}{2}\left(\hat{r}_{ui} - r_{ui}\right)^2. \tag{16}$$

For binary ratings, binary cross-entropy is customarily used:

$$\ell(\hat{r}_{ui}, r_{ui}) = -r_{ui}\log(\hat{r}_{ui}) - (1 - r_{ui})\log(1 - \hat{r}_{ui}). \tag{17}$$

The total loss over the training set is obtained by summing the losses of all ratings:

$$L(P, Q, w) = \sum_{r_{ui} \in \mathcal{R}} \ell(\hat{r}_{ui}, r_{ui}), \tag{18}$$

where $\mathcal{R}$ is the set of all available ratings. Minimizing L alone often leads to overfitting, especially in highly sparse datasets. Introducing a regularization term Ω biases the model towards simplicity, enhancing generalization. The regularized objective function is thus:

$$J(P, Q, w) = L(P, Q, w) + \lambda\Omega(P, Q, w), \tag{19}$$

where λ is the regularization coefficient. Common regularization terms Ω include:

- L_1 Regularization (Lasso Regularization): penalizes the loss function with the L_1 norm of the model parameter vectors, encouraging sparsity:

$$\Omega(P, Q, w) = \sum_{u=1}^{n} \|p_u\|_1 + \sum_{i=1}^{m} \|q_i\|_1 + \|w\|_1. \tag{20}$$

- L_2 Regularization (Ridge Regularization): penalizes the loss function with the square of the L_2 norm of the model parameter vectors, leading to smoother solutions:

$$\Omega(P, Q, w) = \sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2 + \|w\|_2^2. \tag{21}$$

This form of regularization is used in the original MF algorithm and is by far the most widely used due to its differentiability and convexity.

- Elastic Net Regularization: combines both L_1 and L_2 regularization to benefit from the properties of both, encouraging sparsity while also ensuring smoother solutions:

$$\begin{aligned}\Omega(P, Q, w) = &\sum_{u=1}^{n} \|p_u\|_1 + \sum_{i=1}^{m} \|q_i\|_1 + \|w\|_1 \\ &+ \sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2 + \|w\|_2^2.\end{aligned} \tag{22}$$

In addition, methods like early stopping and dropout [52] also help combat overfitting. Early stopping halts training once validation set performance worsens. Dropout, used in neural networks, randomly zeros a fraction of outputs in a layer during training, introducing noise that helps the model generalize better.

3.3. Optimization

Most latent factor approaches rely on continuous optimization algorithms to fit the model to training data. The fundamental algorithm used to find the optimum of a continuously differentiable function is gradient descent. To simplify notation, let us define the vector θ by concatenating all model parameters:

$$\theta = \text{concat}\left(p_1, \ldots, p_n, q_1, \ldots, q_m, w\right). \tag{23}$$

We combine all parameters into one vector because the optimization algorithms do not distinguish between the various model parameters. Gradient descent iteratively updates the parameter values, starting from an initial guess, using the gradient of the objective function:

$$\theta = \theta - \eta\nabla_\theta J(\theta), \tag{24}$$

where η is a parameter known as the *step size* or *learning rate*. Note that the gradient is calculated with respect to the parameters θ, not the rating data r_{ui}. A good choice of η is essential for the algorithm's convergence. Small values can result in slow convergence, while large values can cause oscillations or even divergence. The value of η is often considered a hyperparameter tuned using a validation set, although methods for automatically adjusting the learning rate during training also exist. It is important to remember that gradient descent is a local optimization algorithm, so the solution it finds may not be the global optimum. Good initialization strategies can help find better solutions. For example, restarting optimization from multiple starting points can mitigate local convergence issues.

In the experiment illustrated in Fig. 4, we use a small rating matrix consisting of three users and three items, with two missing ratings: $r_{1,3}$ and $r_{2,2}$. The goal is to predict these missing ratings by fitting a matrix factorization model using gradient descent. The process begins with randomly initializing the user and item factor vectors. The gradient descent algorithm, with learning rate $\eta = 0.01$, then iteratively updates these factors to minimize the objective function. Our objective function does not include a regularization term and consists only of the sum of squared errors (SSE) between the predicted and actual ratings. The objective value and the gradient norm are recorded throughout the iterations to monitor the algorithm's progress. Upon convergence, the obtained factors are used to predict the missing ratings, which gives $\hat{r}_{1,3} = 3.83$ and $\hat{r}_{2,2} = 1.51$.

Gradient descent requires computing the gradient using the entire training set, which can be computationally expensive for large datasets. To address this, Stochastic Gradient Descent (SGD) [53] approximates the gradient using a mini-batch of the training set, which reduces computational costs. Several variants of SGD, including SGD with Nesterov Momentum [54], AdaGrad [55], RMSProp [56], and Adam [57], are widely used in modern machine learning and deep learning models. These and other advanced optimization methods are discussed in detail in Section 7.

4. Learning data

Collaborative filtering typically uses rating data as its primary input, which reflects user preferences through explicit feedback mechanisms. However, integrating contextual information can significantly improve the accuracy and personalization of recommendations. In this section, we explore three key types of contextual data crucial for improving recommendation systems: implicit feedback, trust, and content data. Implicit feedback is indirectly gathered from user activities, such as browsing history and purchase records, and provides insights into user preferences without explicit ratings. Trust data, derived from social interactions and user endorsements, incorporates the relational trust dynamics into the recommendation process. Content data, including item descriptions and user profiles, offers detailed insights into the characteristics of items and users. Systems that combine content data with collaborative filtering techniques are known as hybrid models, combining the strengths of content-based and collaborative filtering approaches to improve recommendation accuracy and personalization.

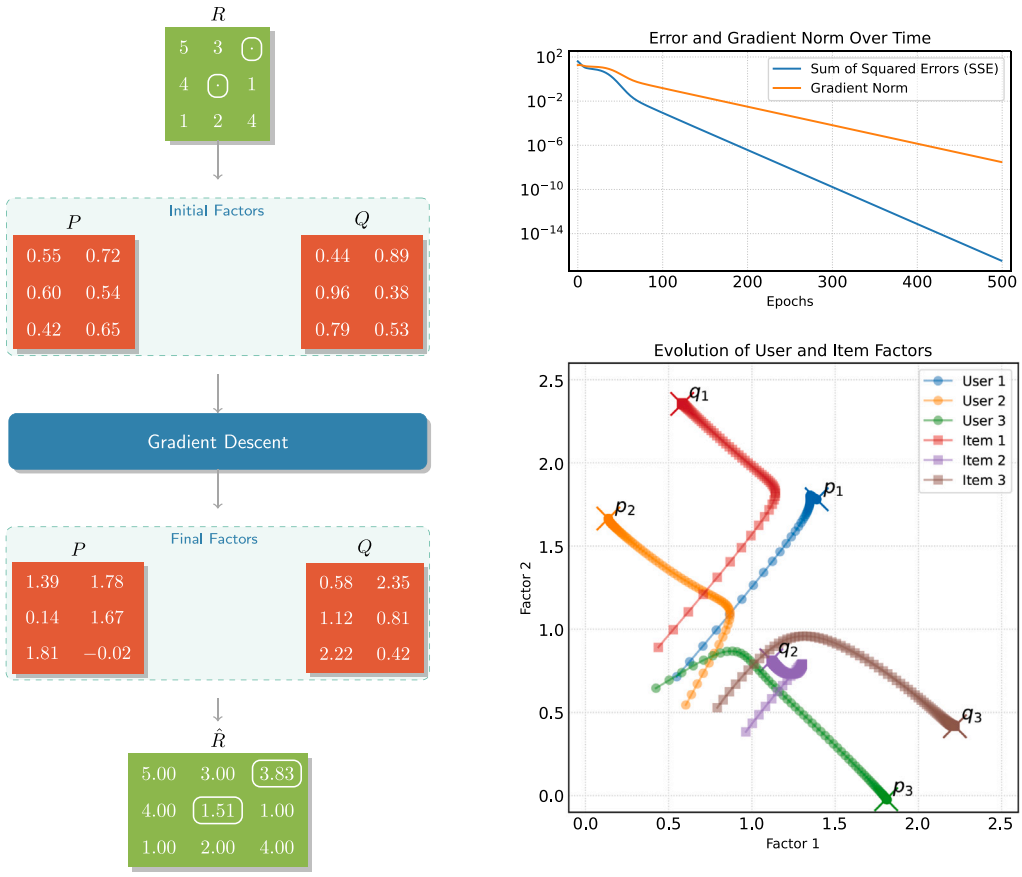

Fig. 4. An example of using gradient descent to fit the matrix factorization model without regularization to data. Left: Starting from randomly generated values, gradient descent iteratively updates user and item factors until convergence. The final factors are used to predict the missing ratings. Right: The upper plot shows the evolution of the objective function (SSE) and the norm of the gradient throughout the optimization procedure. The bottom plot shows the evolution of user and item factors from their random initial positions to their final ones.

4.1. Implicit feedback

Implicit feedback is information collected indirectly from user interactions with items. This type of data includes browsing history, purchase records, or watch time, and it does not require explicit user actions such as ratings. Unlike explicit feedback, implicit feedback provides insights into user preferences through their behavior. It is especially valuable in systems where explicit ratings are sparse or absent, and it can enhance the quality of recommendations by providing additional insights into user preferences for items that were neither purchased nor explicitly rated. Existing approaches can be categorized into linear and nonlinear depending on how the implicit feedback component relates to the predicted ratings.

4.1.1. Linear approaches

In linear approaches, the effect of implicit feedback is captured through latent factors that affect the predicted rating linearly. These often appear as additive terms that adjust the basic matrix factorization prediction. SVD++ [58] is such a model that integrates neighborhood models with matrix factorization to enhance recommendation systems by utilizing both explicit and implicit feedback. This approach merges the strengths of latent factor models and neighborhood-based methods within a unified framework, aiming to improve the accuracy of user preference predictions. The estimated rating given by user u to item i by SVD++ is expressed as:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i + q_i^T \left(p_u + \frac{|N_u|^{-\frac{1}{2}}}{2} \sum_{j \in N_u} y_j \right) + \frac{|\mathcal{R}_{ui}^k|^{-\frac{1}{2}}}{2} \sum_{j \in \mathcal{R}_{ui}^k} w_{ij}(r_{uj} - b_{uj}) + \frac{|N_{ui}^k|^{-\frac{1}{2}}}{2} \sum_{j \in N_{ui}^k} c_{ij}, \tag{25}$$

where $\bar{r}$ is the global average rating across all users and items, b_u is the bias term associated with user u, b_i is the bias term associated with item i, N_u is the set of items for which user u showed an implicit preference, y_j is the implicit feedback factor vector for item j, $\mathcal{R}_{ui}^k = \mathcal{R}_u \cap S_i^k$, where $\mathcal{R}_u$ is the set of items rated by user u and S_i^k the set of k items most similar to i, b_{uj} is the bias term for the interaction of user u with item j, w_{ij} is the weight of the influence of item j on the rating of item i by user u, $N_{ui}^k = N_k \cap S_i^k$, and c_{ij} is a weight factor representing the relationship strength between items i and j for user u.

Shi et al. [59] proposed User Embedding for rating prediction in the SVD++ model [28] (UE-SVD++). The UE-SVD++ model leverages user correlations from the user–item matrix and constructs a user embedding matrix to enhance prediction accuracy. Initially, the most favored users for each item are identified, specifically those whose ratings exceed 70% of the highest rating. This restricted set of ratings is denoted by $\tilde{\mathcal{R}}$. The list of users for each item is termed a context. Using $\tilde{\mathcal{R}}$, User-wise Mutual Information (UMI) values are calculated as follows:

$$UMI(u, v) = \log \frac{p(u, v)}{p(u)p(v)}, \tag{26}$$

where $p(u, v)$ is the joint probability of users u and v appearing together in a context, and $p(u)$ and $p(v)$ are the probabilities of u and v appearing in any context, respectively. These probabilities are computed from the frequency of occurrences in $\tilde{\mathcal{R}}$. A threshold k is then applied to the UMI

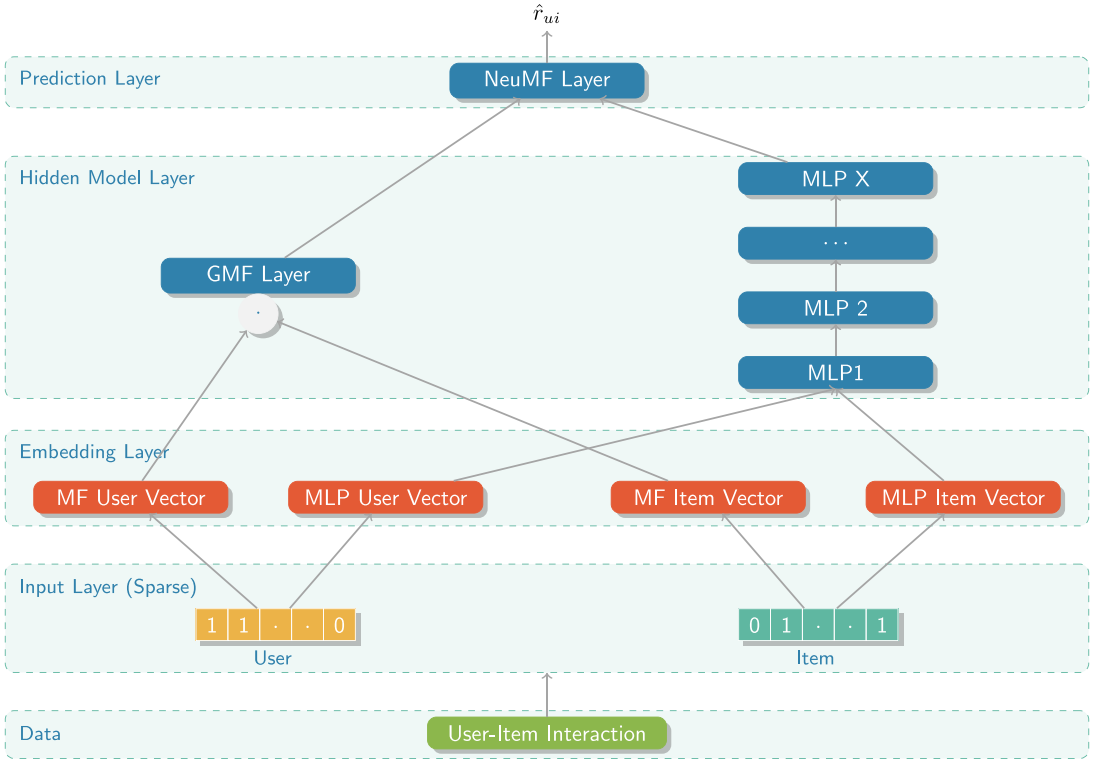

Fig. 5. The architecture of the NCF model [60].

matrix to enhance its reliability:

$$KMUI(u,v) = \begin{cases} UMI(u,v) & \text{if } UMI(u,v) \geq \log(k) \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

The embedding matrix D is constructed based on KMUI as follows:

$$D_{uv} = \begin{cases} 1 & \text{if } KMUI(u,v) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

Define the set D_u as the set of users v such that $D_{uv} = 1$. The rating predicted by UE-SVD++ is an adaptation of the SVD++ rating [58] and is calculated as follows:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i + q_i^T \left(p_u + |N_u|^{-\frac{1}{2}} \sum_{j \in N_u} y_j + |D_u|^{-\frac{1}{2}} \sum_{v \in D_u} z_v \right), \tag{29}$$

where $\bar{r}$ is the global average rating across all users and items, b_u is the bias term associated with user u, b_i is the bias term associated with item i, N_u is the set of items for which user u showed an implicit preference, y_j is the implicit feedback factor vector for item j, and z_v is the factor vector for user-to-user dependency.

4.1.2. Nonlinear approaches

Nonlinear methods capture user–item interactions and the relationship between implicit feedback and ratings through complex models, typically utilizing neural networks. For instance, Zhang et al. [61] introduced Stacked Sparse Auto-encoder Recommender Systems (SSAERec). This model integrates a deep Stacked Sparse Autoencoder into matrix factorization, effectively learning the user–item matrix representation. It utilizes multiple layers of sparse autoencoders, a Sparse Autoencoders (SAE) variant, to enhance feature extraction. The middle-most layer is combined with Singular Value Decomposition++ (SVD++) [48] to incorporate implicit feedback into the model and predict unknown ratings. For optimization, they employed the Adam algorithm [57] to optimize the Stacked Sparse Auto-encoder model and Stochastic Gradient Descent (SGD) [62] to learn the rating prediction parameters.

He et al. [60] introduced the Neural Collaborative Filtering (NCF) model, which aims to enhance traditional collaborative filtering approaches by integrating neural networks. This model focuses on implicit feedback scenarios — such as clicks or views rather than explicit ratings — to learn user preferences more effectively. The NCF framework, shown in Fig. 5, encompasses multiple instantiations, including Generalized Matrix Factorization (GMF), Multi-Layer Perceptron (MLP), and a combined model called NeuMF, which merges GMF and MLP to leverage both linearity and non-linearity in user–item interactions. The core idea is to replace the traditional dot product used in matrix factorization with a neural architecture that can learn an arbitrary function from user–item interactions. This approach allows NCF not only to mimic but also to extend the capabilities of matrix factorization models by introducing non-linearities through deep learning layers. Extensive testing on datasets like MovieLens and Pinterest showed that NCF significantly outperforms classical methods, particularly in handling the complex and often sparse data typically found in collaborative filtering systems.

Liu et al. [63] proposed a neural matrix factorization algorithm based on explicit–implicit feedback (EINMF), which learns both linear and nonlinear explicit–implicit feedback features for the user–item matrix. As illustrated in Fig. 6, the explicit rating matrix and the implicit feedback matrix are used as input for the model. Embedding with normal stochasticity was applied to obtain user and item explicit–implicit latent vectors. These complementary latent feature vectors are input for the hybrid model layer. In the hybrid model layer, matrix factorization was used to extract the linear preference features with the dot product operation, and the Multilayer Perceptron (MLP) model was used to obtain the nonlinear preference features for users and items. They use matrix factorization to extract linear preference features via the dot product operation, and a Multilayer Perceptron (MLP) to capture nonlinear preference features for users and items. Subsequently, outputs from the hybrid model are concatenated to predict the degree of user preferences. This process leads to the prediction of unknown item ratings, providing users with a personalized top-N recommendation list. They introduced a new loss function tailored to explicit and

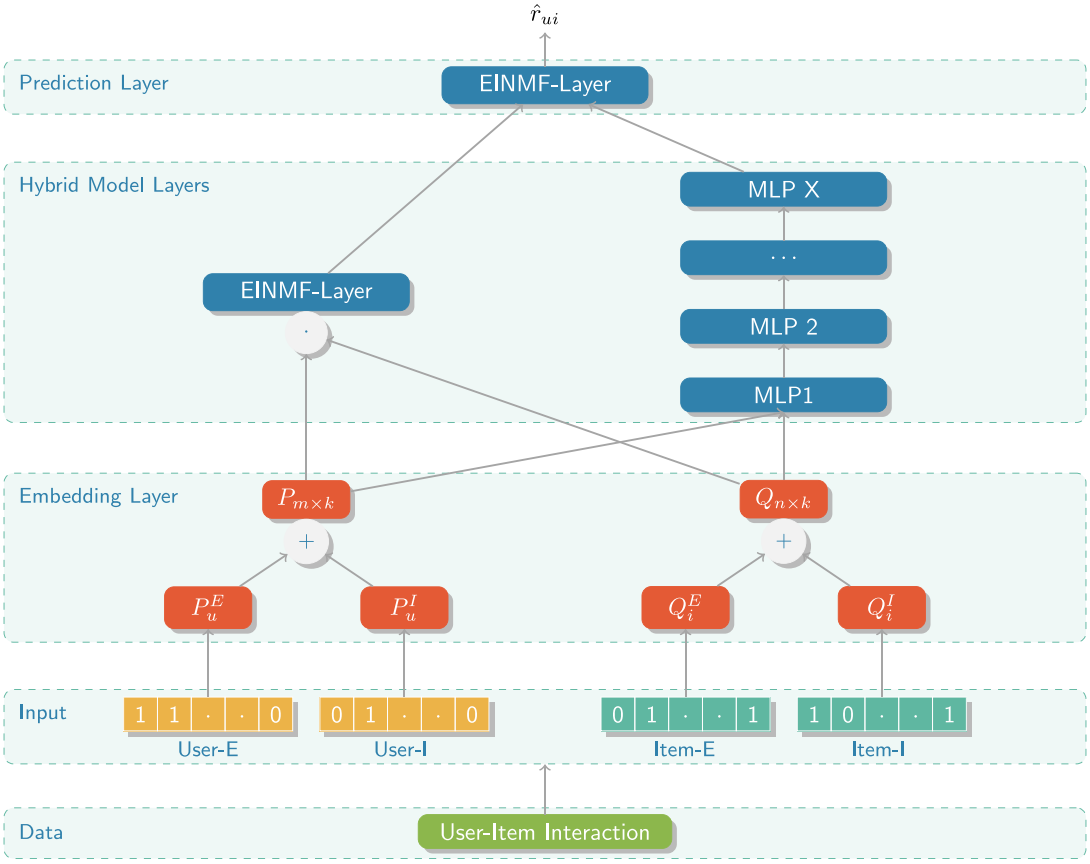

Fig. 6. The architecture of the EINMF model [63].

implicit feedback and optimized the EINMF model parameters using forward and backward propagation.

4.2. Trust

Collaborative filtering and recommendation systems benefit from incorporating trust data, which represents the level of confidence or reliance that a user places in the opinions or behaviors of others. This information is gathered from social interactions on platforms where users rate not just products or services, but also express their trust in other users' recommendations. Trust data is typically found in social networks, user reviews, and interactive platforms where users can follow others or endorse their reviews. By integrating trust metrics into recommendation algorithms, we can leverage the social context of user interactions, which provides an additional layer of data that significantly enhances the accuracy and personalization of recommendations. This is especially useful in addressing challenges such as data sparsity and the cold start problem, where new users or items have insufficient interactions.

TrustMF [50] incorporates social trust into the collaborative filtering process to address the limitations of data sparsity and the cold start problem in recommendations. It utilizes matrix factorization techniques to integrate user rating data with social trust networks, mapping users into two low-dimensional spaces reflecting their roles as trusters and trustees. This dual representation captures the directional nature of trust, where a user's rating behavior is influenced by those they trust and, in turn, influences others. Like traditional matrix factorization, the authors assume that ratings can be expressed as the product of user and item factor vectors $p_u^T q_i$. The trust network is represented as a non-symmetric matrix T where T_{uv} indicates the trust level from user u to user v, ranging from 0 (no trust) to 1 (complete trust). Each user u is associated with two distinct latent feature vectors: a truster-specific vector p_u, the same as the vector used to compute ratings, and a trustee-specific vector w_u. These vectors capture the behaviors of trusting others and being trusted, respectively. The trust value T_{uv} is modeled as the inner product $p_u^T w_v$, reflecting the directional nature of trust. The model fits the rating and trust matrices simultaneously by minimizing the following objective function:

$$J_{TrustMF} = \sum_{r_{ui}\in\mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2 + \lambda_T \sum_{(u,v)\in\Psi} \left(p_u^T w_v - T_{uv}\right)^2 + \lambda\left(\sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2 + \sum_{u=1}^{n} \|w_u\|_2^2\right), \tag{30}$$

where Ψ denotes the set of observed trust relations, λ_T is the weight given to the trust term, and λ is the regularization coefficient. This approach has shown enhanced performance on the Epinions dataset, outperforming traditional collaborative filtering and other trust-aware methods, significantly improving the quality of recommendations.

TrustSVD [49] is an extension of the SVD++ algorithm that takes into account both explicit and implicit influences from user–item ratings and user-user trust. The explicit influence consists of numerical ratings and trust scores, while the implicit influence involves observing which users rated which items and who trusts whom. More precisely, the rating model of TrustSVD is given by:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i + q_i^T \left(p_u + \frac{1}{|\mathcal{R}_u|}\sum_{j\in\mathcal{R}_u} y_j + \frac{1}{|T_u|}\sum_{v\in T_u} s_v\right), \tag{31}$$

where $\bar{r}$ is the global average rating, b_u is the bias of user u, b_i is the bias of item i, $\mathcal{R}_u$ is the set of items rated by user u, y_j is the implicit feedback for item j, T_u is the set of users trusted by user u, and s_v is the implicit trust feedback for user v. The authors analyzed the performance of their approach on several datasets and concluded

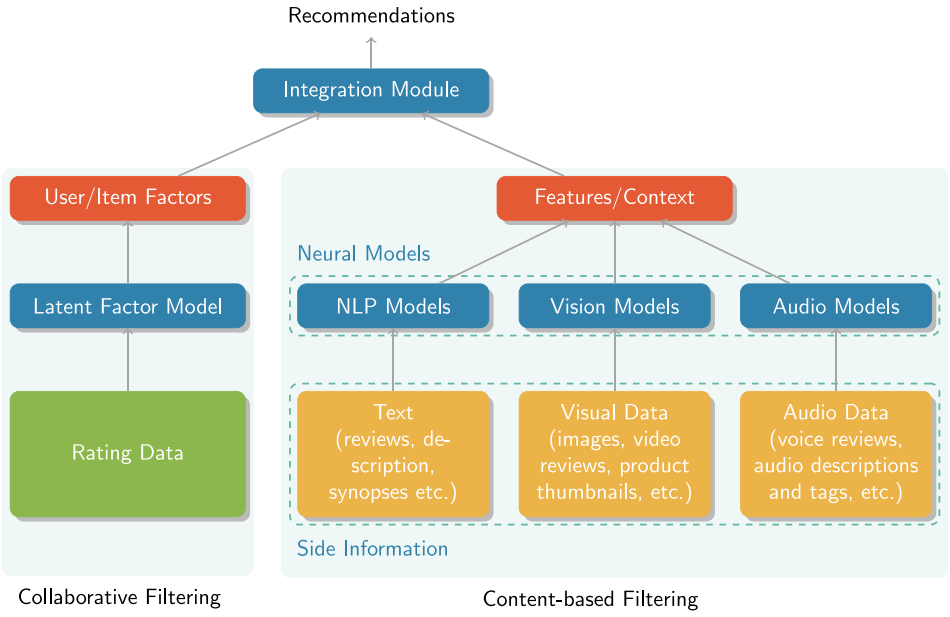

Fig. 7. A hybrid recommendation system combines collaborative and content-based filtering methods to improve prediction accuracy. It leverages diverse data types such as textual, visual, and audio through neural models to extract relevant features. The extracted features are then combined with user/item factors to produce the final recommendations.

that trust information is sparse but complements rating information, leading to improved recommendation accuracy.

Incorporating social trust information as an additional data source has proven effective in improving prediction accuracy while reducing computational costs and enhancing convergence speed. Parvin et al. [64] proposed a social regularization method called Trust Regularized Single-element based Non-Negative Matrix Factorization (TrustRSNMF). The TrustRSNMF model extends the Regularized Single-element based Non-Negative Matrix Factorization (RSNMF) [65] model by considering users' trust statements in the recommendation process. Users' trust statements were incorporated into the non-negative matrix factorization to improve prediction accuracy.

4.3. Content data

Incorporating content data, such as item descriptions and user profiles, into rating data gives rise to hybrid models. This integration leverages additional information such as item descriptions, user profiles, and textual content commonly found in e-commerce product reviews or movie review sites. For instance, item descriptions in online shopping platforms or movie synopses in streaming services provide rich, descriptive data. Additionally, the availability of textual and sometimes imagery data creates an ideal scenario for employing neural networks, particularly deep learning models, to extract pertinent features that enhance recommendation accuracy. As illustrated in Fig. 7, most hybrid models combine a neural model, which processes textual or visual content, with a collaborative filtering framework. The features extracted by the neural model are then integrated with the outputs from the collaborative filtering model to produce more precise and personalized recommendations.

Kim et al. [66] proposed a context-aware recommendation model called Convolutional Matrix Factorization (ConvMF), which integrates a Convolutional Neural Network (CNN) [67] into Probabilistic Matrix Factorization (PMF) [68]. Initially, a CNN analyzes documents' contextual information and generates latent document vectors. These vectors are then integrated with the PMF model to predict unknown ratings. Consequently, CNN provides a deep representation of documents, enhancing the accuracy of rating predictions.

They applied maximum a posteriori (MAP) [68] estimation to optimize the latent model variables for users and items, as well as the weights and biases of the CNN. Three datasets were used for testing: MovieLens-1M, MovieLens-10M, and Amazon, with RMSE as the evaluation metric. The ConvMF model was compared with Probabilistic Matrix Factorization (PMF), Collaborative Deep Learning (CDL) [69], and ConvMF with a pre-trained word embedding model (ConvMF+). The results demonstrated that both ConvMF and ConvMF+ significantly enhanced prediction accuracy compared to PMF across all datasets.

Mohd et al. [70] analyzed word documents using Long Short-Term Memory (LSTM) networks to transform product review documents into a 2D latent space vector. They then integrated this vector with Probabilistic Matrix Factorization (PMF) [68], known for effectively generating rating predictions in large datasets and robustly handling imbalanced data.

The first layer of LSTM-PMF is responsible for collecting the datasets. The second layer employs the Natural Language Toolkit (NLTK) for preprocessing and Global Vector (GloVe) for word embedding [71]. The third layer uses LSTM to transform the product review documents into a 2D vector space. PMF integrates the item and user latent vectors in the fourth layer to generate a rating prediction. The final layer utilizes the Root Mean Square Error (RMSE) [39] metric to evaluate prediction accuracy. Learning and weight variables are optimized using Maximum A Posteriori distribution (MAP) [68] and the back-propagation algorithm. In their experiments with the MovieLens-1M and MovieLens-10M datasets, the proposed LSTM-PMF model was compared with traditional PMF and a Convolutional Neural Network-based PMF (CNN-PMF) [66], using RMSE as the performance measure.

Sun et al. [72] introduced a probabilistic framework, Joint Matrix Factorization (JMF), incorporating user and item latent information alongside additional side information from product reviews. Differing from Mohd et al. [70], they employed a modified Long Short-Term Memory (LSTM) [73], precisely Bidirectional Long Short-Term Memory (BLSTM), to capture information from both directions of a sequence. The JMF model comprises three components:

1. A Multilayer-crossing Factorization Machine (MFM), a variation of the Factorization Machine (FM) [51,74] designed for improved computational efficiency and noise reduction. This component extracts latent user factors based on behavioral data.
2. The BLSTM, which processes item latent factors from document sequences.

3. These components are integrated with Probabilistic Matrix Factorization (PMF) to generate predictive ratings.

Five datasets were utilized for their experiments: MovieLens-100k, MovieLens-1M, MovieLens-10M, Amazon Music, and Amazon Baby. The Root Mean Square Error (RMSE) [39] was employed as the evaluation metric.

Ong et al. [75] explored linear and nonlinear user and item representations. They introduced the Neural Matrix Factorization++ (NeuMF++) model, an enhanced version of NeuMF [60]. NeuMF++ incorporates side information using Stacked Denoising Autoencoders (SDAE) [76], effectively capturing latent features.

The NeuMF++ model consists of two main components: feature extraction and neural collaborative filtering. Initially, SDAEs process each user and item feature to derive latent representations. Subsequently, high-level features are extracted from the middle-most layer. NeuMF++ integrates Generalized Matrix Factorization (GMF++) to address linearity and a Multilayer Perceptron (MLP++) to manage nonlinearity in the neural collaborative filtering stage. The features extracted from SDAEs are treated as embeddings, concatenated, and fed into both GMF++ and MLP++. The outputs from GMF++ and MLP++ are then concatenated into a single MLP layer to predict unknown ratings. The Adam optimizer [57] is employed to refine the NeuMF++ model parameters. In their experiments using the MovieLens-1M dataset, the NeuMF++ model was benchmarked against the original NeuMF model using RMSE.

Zhang et al. [77] introduced a new hybrid model that integrates a Contractive Auto-encoder (CAE) [78] with Biased Singular Value Decomposition (BSVD) [48], termed AutoSVD. The CAE extracts complex, nonlinear feature representations of item information, which are then integrated into SVD to enhance learning and prediction of unknown ratings. They extended this approach with AutoSVD++, incorporating implicit feedback, side information, and the user–item rating matrix to address data sparsity better. Experiments were conducted across three datasets: MovieLens-100k, MovieLens-1M, and MovieTweetings. AutoSVD and AutoSVD++ were compared with SVD++ [28] and Biased SVD [28] using RMSE.

Nassar et al. [79] suggested a multi-criteria collaborative filtering recommender system by integrating a Deep Neural Network (DNN) [80] and Matrix Factorization (MF) [28]. This integration aims to harness the non-linearity of DNN alongside the linearity of MF to improve rating predictions. The proposed model is an evolution of their previous work, the Deep Multi-Criteria Collaborative Filtering (DMCCF) [81] model, differing primarily in its use of MF in conjunction with a DNN to predict criteria-specific ratings, whereas the original DMCCF utilized only DNN.

The model operates in two phases. Initially, user and item features are input into a combined model of a DNN and MF to predict the criteria ratings. Subsequently, these criteria ratings are used as inputs for another DNN to predict the overall ratings. Both model components are optimized using the Adam optimizer [57] and the MAE loss function.

Experiments conducted with the TripAdvisor and Movies datasets compared the performance of the proposed model against the DMCCF. The evaluation metric used was the RMSE. The results indicate that the proposed model performs better on both datasets and significantly outperforms the DMCCF model.

4.4. Discussion

The section on learning data focuses on improving collaborative filtering by including contextual information. Three key types of contextual data are explored: implicit feedback, trust, and content data:

- Implicit feedback is gathered indirectly from user interactions like browsing history and purchase records, providing insights into user preferences without requiring explicit ratings. This data type is particularly valuable in scenarios where explicit ratings are sparse or unavailable, as it enhances the quality of recommendations by revealing user interests through their behavior.
- Trust data is derived from social interactions and user endorsements, integrating relational trust dynamics into the recommendation process. By incorporating trust data, recommendation systems can leverage the social context of user interactions, improving the accuracy and personalization of recommendations. This approach is especially useful for addressing data sparsity and the cold start problem.
- Content data includes information about items and users, such as item descriptions and user profiles. This data type enriches the recommendation process by providing additional context and characteristics of the items and users involved. Hybrid models, which combine content data with collaborative filtering techniques, leverage the strengths of both content-based and collaborative filtering approaches. This combination results in more precise and personalized recommendations, as neural networks and deep learning models can extract rich features from textual and imagery data.

Table 2 provides a summary of the main experimental results from surveyed papers using these three types of learning data. These research areas remain significant and continue to offer valuable insights and improvements for recommendation systems. There are also new research directions where integrating sophisticated data types to enhance the accuracy and personalization of recommendations promises to advance the field further.

Multimodal data integration, which combines text, images, and audio, leverages deep learning for effective processing [4,82]. This approach helps create richer and more contextually aware recommendations by utilizing diverse data types.

Context-aware recommendations use information like time and location to provide timely suggestions [5,83,84]. By adapting to the user's current situation, these systems can deliver more relevant and personalized content.

Physiological and behavioral signals, such as EEG and gaze tracking, capture deeper user insights for personalized recommendations [85–90]. These signals provide valuable information about users' cognitive and emotional states, allowing recommendation systems to tailor suggestions based on real-time user feedback and interactions.

5. Models

This section covers advanced modeling techniques that can significantly improve the performance of recommendation engines. We will discuss probabilistic models that use statistical methods to handle sparse and large-scale data, weighted models that assign varied importance to different factors, kernelized and nonlinear models that capture complex, nonlinear relationships within the data, and Graph Neural Networks (GNN), which leverage the relational structure of data to enhance the accuracy and personalization of recommendations. Each category represents an improvement over traditional matrix factorization methods, incorporating innovative strategies such as probabilistic inference, dynamic weighting, and high-dimensional data transformations to provide more accurate and personalized user–item recommendations.

5.1. Probabilistic models

Probabilistic Matrix Factorization (PMF) [68] tackles the challenges of handling large-scale, sparse, and imbalanced datasets like those encountered in the Netflix Prize competition. Traditional latent factor models often struggle with such datasets due to their inability to manage missing data effectively and the computational demands posed by large-scale data. PMF addresses these issues by scaling linearly with the number of observations and implementing probabilistic linearity to handle data sparsity efficiently. The model asserts that user preferences can be expressed as a product of two lower-rank matrices, one for users

Table 2
Summary of experimental results from surveyed papers using various additional learning data types.

Reference	Type of Additional Data	Datasets Used	Results
SVD++ [58]	Implicit feedback	Netflix	RMSE: 0.887
UE-SVD++ [59]	Implicit feedback	MovieLens-100k, Epinions, FilmTrust, EachMovie	MovieLens-100k: RMSE 0.942, Epinions: RMSE 1.058, FilmTrust: RMSE 0.802, EachMovie: RMSE 0.257
SSAERec [61]	Implicit feedback	Ciao, MovieLens-100k, MovieLens-1M	MovieLens-100k: RMSE 0.902, MovieLens-1M: RMSE 0.848
NCF [60]	Implicit feedback	MovieLens, Pinterest	MovieLens-1M: HR@10: 0.69, NDCG@10: 0.42 (8 factors) HR@10: 0.87, NDCG@10: 0.55 (8 factors)
EINMF [63]	Implicit feedback	MovieLens-100K, MovieLens-1M	MovieLens-100K: HR 0.698, NDCG 0.316; MovieLens-1M: HR 0.654, NDCG 0.283
TrustMF [50]	Trust	Epinions	Epinions: RMSE 1.059
TrustSVD [49]	Trust	Epinions, FilmTrust, Flixster, Ciao	Epinions: RMSE 1.044, FilmTrust: RMSE 0.787, Flixster: RMSE 0.950, Ciao: RMSE 0.956
TrustRSNMF [64]	Trust	Epinions, FilmTrust	FilmTrust: MAE 0.603, RMSE 0.753, Epinions: MAE 0.843, RMSE 1.052
Kim et al. [66]	Content data	MovieLens-1M, MovieLens-10M, Amazon	MovieLens-1M: RMSE 0.853, MovieLens-10M: RMSE 0.793, Amazon: RMSE 1.128
Mohd et al. [70]	Content data	MovieLens-1M, MovieLens-10M	MovieLens-1M: RMSE 0.841, MovieLens-10M: RMSE 0.790
Sun et al. [72]	Content data	MovieLens-100k, MovieLens-1M, MovieLens-10M, Amazon Music, Amazon Baby	MovieLens-10M: RMSE 0.782, MovieLens-1M: RMSE 0.841, MovieLens-100k: RMSE 0.902
Ong et al. [75]	Content data	MovieLens-1M	RMSE: 0.868
Zhang et al. [77]	Content data	MovieLens-100k, MovieLens-1M, MovieTweetings	MovieLens-100k: RMSE 0.901, AutoSVD++: RMSE 0.904, MovieLens-1M: RMSE 0.848 (AutoSVD++), 0.864 (AutoSVD)
Nassar et al. [79]	Content data	TripAdvisor	TripAdvisor: RMSE 0.743

and one for items, incorporating Gaussian noise to model the variability in user ratings. This approach not only provides more robust handling of sparse data but also improves prediction accuracy for users with few ratings by using a constrained version of PMF, which assumes users rating similar sets of movies have similar tastes.

PMF assumes a prior distribution on the latent factor vectors for both users and items to regularize these parameters:

$$p\left(P|\sigma_u^2\right) = \prod_{u=1}^{n} \mathcal{N}\left(p_u|0, \sigma_U^2\mathbf{I}\right), \quad p\left(Q|\sigma_i^2\right) = \prod_{i=1}^{m} \mathcal{N}\left(q_i|0, \sigma_I^2\mathbf{I}\right), \tag{32}$$

where $\mathcal{N}$ is the normal distribution, σ_U^2 and σ_I^2 are the variances of these priors, and $\mathbf{I}$ is the identity matrix. The probability of observing the set of ratings $\mathcal{R}$ given the latent factors P, Q and noise variance σ^2 is defined as:

$$p\left(\mathcal{R}|P, Q, \sigma^2\right) = \prod_{r_{ui}\in\mathcal{R}} \mathcal{N}\left(r_{ui}|p_u^T q_i, \sigma^2\right), \tag{33}$$

where p_u and q_i are the latent feature vectors for users and items. The optimization of this model involves minimizing a loss function that balances the reconstruction error and regularization terms to prevent overfitting, tailored through hyperparameters that control model complexity. By treating σ_U, σ_I and σ as hyperparameters, maximizing the log-likelihood of (33) amounts to minimizing:

$$J_{PMF} = \sum_{r_{ui}\in\mathcal{R}} \left(r_{ui} - p_u^T q_i\right)^2 + \lambda_U \sum_{u=1}^{n} \|p_u\|_2^2 + \lambda_I \sum_{i=1}^{m} \|q_i\|_2^2, \tag{34}$$

where $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_I = \sigma^2/\sigma_I^2$. The effectiveness of PMF and its extensions, such as constrained PMF and PMF with adaptive priors, was demonstrated through extensive experiments on the Netflix dataset. The combined use of PMF models and Restricted Boltzmann Machines led to a performance improvement of nearly 7% over Netflix's algorithm, achieving an error rate of 0.886.

5.2. Weighted models

Traditional latent factor models often assume that latent factors are weighted equally, which may not always be a reasonable assumption. This motivated Chen et al. [91] to develop the Weighted-Singular Value Decomposition (WSVD) model, which assigns different weight values to latent factors to explain their importance. They used the SVD model [32] to generate latent factors and then assigned a weight to each latent factor:

$$\hat{r}_{ui} = \bar{r} + b_u + b_i + (w \odot p_u)^T \cdot q_i, \tag{35}$$

where $\bar{r}$ refers to the average of all ratings, b_u and b_i are the user bias for user u and the item bias for item i, respectively. The vector w contains the weights of the latent factors, and p_u is the vector of user u's latent factors. The operator $\odot$ denotes the Hadamard product between vectors w and p_u, and q_i is the vector of item i's latent factors. Stochastic Gradient Descent (SGD) is used for optimization during the model's training.

In their experiments, they used five datasets: MovieLens-100k, MovieLens-1M, MovieLens-10M, FilmTrust, and Epinions. The WSVD model was compared with the SVD and SVD++ [28]. WSVD achieved an RMSE of 0.943 on MovieLens-100k, 0.992 on MovieLens-1M, 0.947 on MovieLens-10M, and 1.093 on FilmTrust, outperforming baseline methods for all datasets.

Gu et al. [92] also proposed a weighted SVD model (wSVD), which, unlike the model in [91], assigns weights to each rating in the rating matrix to reduce the effects of noise and unreliable ratings. This approach applies the SVD model to the original matrix to calculate an entrywise absolute error e_{ui} as follows:

$$e_{ui} = \left|r_{ui} - p_u^T q_i\right|, r_{ui} \in \mathcal{R}, \tag{36}$$

where r_{ui} is the rating given by user u on item i, p_u and q_i are the user vector and the item vector, respectively. A lower weight is then

assigned to entries with a high absolute error:

$$w_{ui} = \phi(e_{ui}), \tag{37}$$

where ϕ is a non-increasing mapping. After obtaining the weights, the model is obtained by minimizing the following objective function:

$$J_{wSVD} = \sum_{r_{ui}\in\mathcal{R}} w_{ui}\left(r_{ui} - \bar{r} - b_u - b_i - p_u^T q_i\right)^2 + \lambda\left(\sum_{u=1}^{n}\left(\|p_u\|_2^2 + b_u^2\right) + \sum_{i=1}^{m}\left(\|q_i\|_2^2 + b_i^2\right)\right), \tag{38}$$

where b_u and b_i are the user bias for user u and item bias for item i, respectively. Gu et al. [92] also suggested an initial guess to speed up the optimization process by employing a direct sparse SVD solver. They performed experiments on MovieLens-100k, MovieLens-1M, and MovieTweetings, using the Root Mean Square Error (RMSE) [39] metric to measure the model's performance. The proposed user-based weight (wSVDu) and user–item-based weight (wSVDui) were compared with Biased SVD (BSVD) [28] and SVD++. The best result was achieved by the user–item-based weight (wSVDui) model with RMSE 0.839 for MovieLens-100k, 0.801 for MovieLens-1M, and 1.317 for MovieTweetings. The results showed that the user–item-based weight (wSVDui) model outperformed other models in all cases.

5.3. Kernelized and nonlinear models

Matrix factorization typically assumes data are distributed on a linear hyperplane, which is not always true. Liu et al. [93] proposed Kernel Matrix Factorization (KMF), integrating matrix factorization with kernel methods [94]. KMF embeds the latent factor matrices in a higher-dimensional feature space, as shown in Fig. 8, to learn the non-linear correlations of the ratings in the original space.

Kernel methods implicitly embed data into a high-dimensional, possibly infinite-dimensional, space using a kernel. Let $\mathcal{X}$ denote the original data space, and $\mathcal{H}$ the high-dimensional feature space, which has the structure of a Hilbert space and is sometimes referred to as the kernel space. Denote by $\phi : \mathcal{X} \to \mathcal{H}$ the embedding map that associates each data point x with its embedding $\phi(x)$. The mapping ϕ is generally only implicitly defined and observed only through the inner product $\phi(x)^T\phi(x') \in \mathbb{R}$. A widely used choice of kernel is the Gaussian kernel:

$$K(x, x') = \phi(x)^T\phi(x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \tag{39}$$

where σ^2 is the bandwidth parameter. We assume that $\mathcal{X} = \mathbb{R}^d$, where d is a hyperparameter of the model. Each user u is represented by a vector $a_u \in \mathbb{R}^d$ and each item i by a vector $b_i \in \mathbb{R}^d$. Let k denote the number of factors, that is, the dimensionality of the latent factor space (k is generally different from d). Start by randomly selecting k vectors $d_1, \ldots, d_k \in \mathbb{R}^d$. We refer to these vectors as the dictionary vectors. We assume that we can write $\phi(a_u)$ as a linear combination of the images of the dictionary vectors:

$$\phi(a_u) = \sum_{j=1}^{k} p_{uj}\phi(d_j) = \Phi p_u, \tag{40}$$

where $\Phi = \left(\phi(d_1), \ldots, \phi(d_k)\right)$, and $p_u = \left(p_{u1}, \ldots, p_{uk}\right)$ is the latent factor vector associated with user u. The same applies to items:

$$\phi(b_i) = \sum_{j=1}^{k} q_{ij}\phi(d_j) = \Phi q_i. \tag{41}$$

The rating given by user u to item i is estimated as the inner product of $\phi(p_u)$ and $\phi(q_i)$ in the kernel space:

$$\hat{r}_{ui} = \phi(p_u)^T\phi(q_i) = (\Phi p_u)^T\Phi q_i = p_u^T(\Phi^T\Phi)q_i = p_u^T\mathbf{K}q_i. \tag{42}$$

The matrix $\mathbf{K}$ is the Gram matrix associated with the dictionary vectors $\{d_i\}$, obtained by applying the kernel K to each pair of dictionary vectors (d_i, d_j):

$$\mathbf{K}_{ij} = K(d_i, d_j). \tag{43}$$

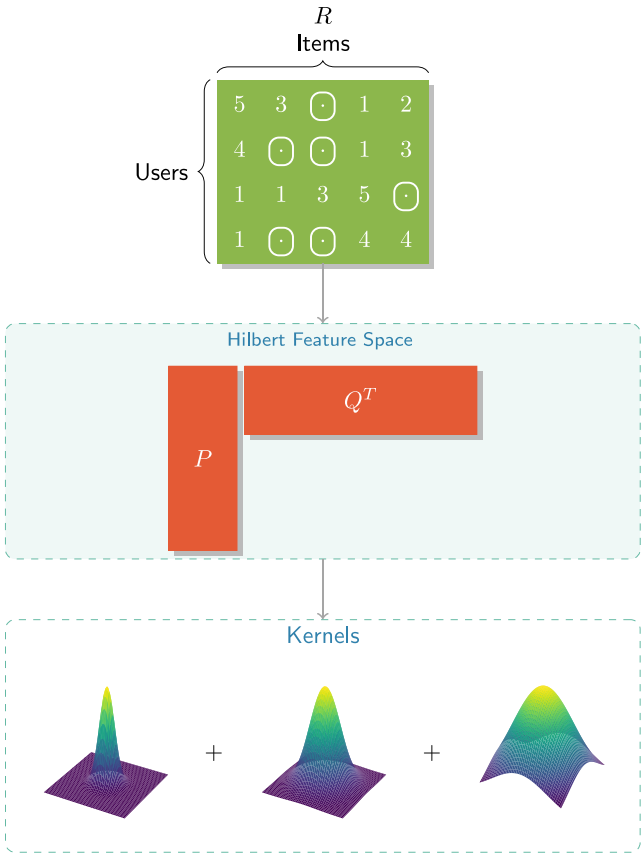

Fig. 8. Kernelized low-rank matrix factorization [93] enhances prediction accuracy by embedding latent factor matrices P and Q into a high-dimensional Hilbert feature space using kernel functions. This non-linear reconstruction of the rating matrix R in its original space is achieved through the product of these transformed matrices.

The matrices P and Q are obtained by minimizing the following objective:

$$J_{KFM} = \sum_{r_{ui}\in\mathcal{R}}\left(r_{ui} - p_u^T\mathbf{K}q_i\right)^2 + \lambda\left(\sum_{u=1}^{n}\|p_u\|_2^2 + \sum_{i=1}^{m}\|q_i\|_2^2\right) \tag{44}$$

Inspired by Multiple Kernel Learning (MKL) methods [95–97], KMF extends to Multiple KMF (MKMF). MKMF combines multiple kernels, which learn a set of weights for each kernel function based on observed data in the rating matrix to improve prediction accuracy. The predicted rating is written as the weighted sum of multiple kernel inner products:

$$\hat{r}_{ui} = \sum_{j=1}^{l} w_j p_u^T\mathbf{K}_j q_i, \quad \text{where } \sum_{j=1}^{l} w_j = 1, w_j \geq 0, j = 1, \ldots, l. \tag{45}$$

The parameter l is the number of kernels, $\mathbf{K}_j$ denotes the jth kernel, and w_j denotes the weight given to $\mathbf{K}_j$. The objective function for the case of multiple kernels is defined as:

$$J_{MKFM} = \sum_{r_{ui}\in\mathcal{R}}\left(r_{ui} - \sum_{j=1}^{l} w_j p_u^T\mathbf{K}_j q_i\right)^2 + \lambda\left(\sum_{u=1}^{n}\|p_u\|_2^2 + \sum_{i=1}^{m}\|q_i\|_2^2\right) + \lambda'\sum_{j=1}^{l} w_j^2, \tag{46}$$

where λ and λ' are regularization coefficients. The authors solve the minimization problem using an alternating algorithm, where first Q is fixed and the best P is found, then P is fixed, and the best Q is calculated until convergence. They performed experiments on several datasets: MovieLens, Flixster, Jester, Yahoo Music, ASSISTments, and Dating Agency, using the Root Mean Square Error (RMSE) [39] as

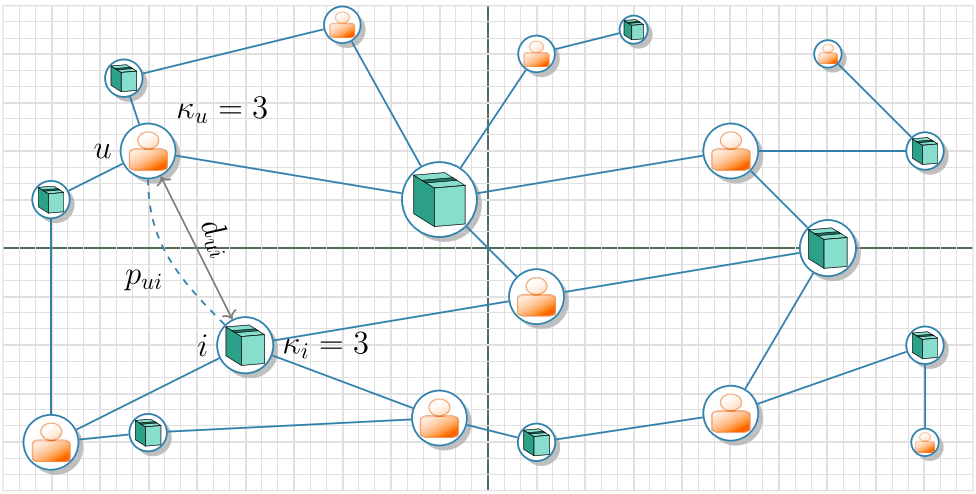

Fig. 9. The SPHM model [101] fits the rating data to a similarity-popularity network model based on a hidden metric. The predicted rating from user u to item i is proportional to the probability of connection p_{ui}, which increases with the user degree κ_u and item degree κ_i and decreases with the distance d_{ui} between them.

the performance measure. They compared Kernel Matrix Factorization (KMF), which is based on a single kernel, with Multiple Kernel Matrix Factorization (MKMF), Matrix Factorization [28], and SVD [28]. The results indicated that the MKMF model improved prediction accuracy compared with KML due to the use of multiple kernels. Additionally, MKMF outperformed both Matrix Factorization and SVD. The best RMSE results for MKMF were 0.816 on MovieLens, 0.815 on Flixster, 4.081 on Jester, and 18.503 on Yahoo Music.

While kernelized approaches, such as KMF, offer the advantage of embedding data into a higher-dimensional space through non-linear mapping, they maintain a linear relationship with respect to the model parameters, allowing for efficient optimization. Lawrence et al. [98] explored the potential of a nonlinear representation of latent factors to enhance prediction accuracy in matrix factorization. They proposed a nonlinear variant of Probabilistic Matrix Factorization (PMF) [68] by integrating the Gaussian Process Latent Variable Model (GP-LVM) [99]. This nonlinear approach involved substituting the inner product in PMF with a Radial Basis Function (RBF) kernel [100]. The latent representations were optimized using Stochastic Gradient Descent (SGD) [62].

In their experiments with the MovieLens-1M, MovieLens-10M, and EachMovie datasets, they compared the performance of the proposed nonlinear model to that of a conventional linear model using Root Mean Square Error (RMSE) [39] as one of the metrics. The RMSE values on MovieLens-1M were 0.875 for the nonlinear RBF model and 0.878 for the linear model. These results indicate that the nonlinear kernel slightly improved prediction accuracy compared to the linear model.

Authors in [101] proposed a novel method for recommender systems that leverages the structure of complex networks. This method models users and items as nodes within a network, using a similarity-popularity model to predict ratings. The primary goal is to address common challenges in recommender systems, such as data sparsity and the cold-start problem, by deriving insight from complex network models.

Similarity-popularity models assume that two factors control node connectivity: their similarity and their popularity. More similar nodes tend to connect. Popularity, generally reflected by the node degree, is an innate property of the node that indicates its capacity to connect to other nodes. Popular nodes connect to other nodes even if they are highly dissimilar. The hidden metric space model is a similarity-popularity model where similarity between nodes is prescribed by an underlying hidden space metric, where similarity is the inverse of the distance.

This approach fits a network model to the rating data, meaning that the model implicitly defines the network, but no actual edges are created (see Fig. 9). The SPHM (Similarity-Popularity Hidden Metric) model proposed in [101] first scales the rating to the interval $[p_{\min}, p_{\max}] \subset (0, 1)$:

$$\tilde{r}_{ui} = \phi(r_{ui}) = \frac{r_{ui} - r_{\min}}{r_{\max} - r_{\min}}(p_{\max} - p_{\min}) + p_{\min}. \tag{47}$$

The probability of connections between the user node and item node gives the predicted scaled ratings in SPHM:

$$\hat{\tilde{r}}_{ui} = \left(1 + \frac{d^2(p_u, q_i)}{\sqrt{\kappa_u \kappa_i}}\right)^{-1}, \tag{48}$$

where $d(p_u, q_i)$ stands for the Euclidean distance between p_u and q_i, and κ_u and κ_i are the degrees associated with the user u and item i nodes respectively. These are defined as:

$$\kappa_u = \bar{r}_u - r_{\min} + 1, \tag{49}$$

$$\kappa_i = \bar{r}_i - r_{\min} + 1, \tag{50}$$

where $\bar{r}_u$ and $\bar{r}_i$ are the average ratings of user u and item i respectively, and $r_{\min}$ is the minimum possible rating in the system. The factors p_u and q_i are obtained by minimizing the following objective function:

$$J_{SPHM} = \sum_{\tilde{r}_{ui} \in \tilde{\mathcal{R}}} \left(\hat{\tilde{r}}_{ui} - \tilde{r}_{ui}\right)^2 + \lambda \left(\sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2\right), \tag{51}$$

where $\tilde{\mathcal{R}}$ is the set of scaled ratings as defined in Eq. (47), and $\hat{\tilde{r}}_{ui}$ is the predicted scaled rating defined in Eq. (48). The authors use the conjugate gradient algorithm proposed in [102] to solve this optimization problem. The predicted ratings are then obtained using the inverse of ϕ. SPHM's performance was demonstrated through an extensive experimental analysis conducted on 21 datasets against ItemKNN, SVD++, PMF, and BiasedMF.

Neural models are highly adaptable and can be used not only in hybrid systems but also in purely collaborative filtering contexts. Their nonlinear modeling capabilities help improve the recommendation quality by capturing intricate patterns in user–item interactions that linear methods might overlook. For instance, Shang et al. [103] proposed a novel nonlinear regression model that combines the Extreme Learning Machine (ELM) [104] with Weighted Nonnegative Matrix Tri-Factorization (WNMTF) [105–107], named WNMTF Combined ELM for Collaborative Filtering (CELMCF) algorithm. WNMTF was employed as a preprocessing step to initialize unobserved ratings in the user–item matrix, effectively mitigating the data sparsity issue. Subsequently, the ELM algorithm, which includes a single hidden layer with several nodes, was used to generate recommendations. Their experiments utilized several datasets, including MovieLens, BookCrossing,

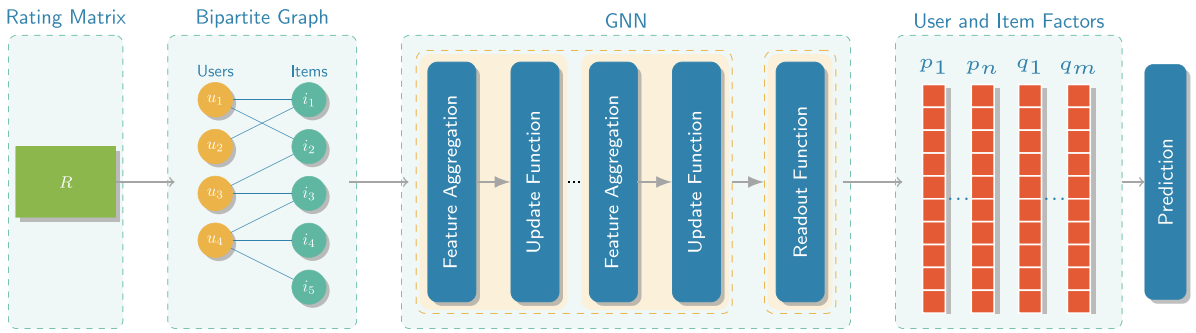

Fig. 10. Using GNN for collaborative filtering [6].

Jester, and Tencent Weibo. They compared their model with Memory-based Collaborative Filtering (MemCF) [108] and ELMCF, using the Mean Absolute Error (MAE) [39] as the evaluation metric. The results indicated that the proposed CELMCF model outperformed all other algorithms on these datasets. Specifically, CELMCF achieved the best MAE results of 0.653 on MovieLens, 0.602 on Jester, 0.521 on BookCrossing, and 0.105 on Tencent Weibo.

5.4. Graph neural networks

Graph neural networks (GNNs) are a type of neural network that is specialized in processing complex structured data, extending deep neural network applications to graph data structures [109]. These networks excel in tasks such as social network analysis, bioinformatics, and computer vision by utilizing the detailed relational information within graphs. Graph convolutional networks (GCNs), a prominent variant of GNNs, are particularly effective at learning graph representations by aggregating neighborhood features, similar to traditional convolution operations in neural networks [110]. This approach adapts convolutional kernels from image processing to graph data, facilitating parameter sharing among nodes and simplifying the model, ultimately reducing training time [111].

A GNN generally comprises multiple blocks in cascade, each performing two critical operations at each node of the graph [6]: feature aggregation, where information from the neighbors is collected, and node representation update, where the vector embedding of the node is updated based on the collected features. The last block of a GNN implements a readout function that uses the output of the previous blocks to generate the final node representations.

As shown in Fig. 10, to use a GNN for collaborative filtering, a graph is first constructed using the rating matrix. The graph contains two types of nodes: user and item nodes. The GNN then takes this graph as input, and the node embedding vectors are used as factors, which are then used to make predictions. For each of these steps, a variety of design choices are available, each influencing the effectiveness and efficiency of the network:

- **Graph construction**: Most works have applied GNN on user–item bipartite graphs. However, directly applying GNN on the original graph may not be effective or efficient due to the graph structure and computation cost [112–117]. To address these issues, strategies such as adding edges between two-hop neighbors [118,119], introducing virtual nodes [120], and sampling techniques [119,121] have been proposed. These strategies aim to enrich the original graph structure, improve expressiveness and computational efficiency. The choice of sampling strategy affects the performance of the model and requires further study.
- **Feature aggregation**: is a crucial part of information propagation for graph structures. Mean-pooling is a simple and popular aggregation operation [112,119,122,123], but it may not be suitable when the importance of neighbors varies significantly. Other works use "degree normalization" to assign weights to nodes based on the graph structure [113,124,125]. Some methods use an attention mechanism to learn the weights of neighbors [126,127].
- **Update function** is crucial for iterative information propagation in graph neural models. Existing methods can be classified into two categories based on whether they discard original node information or not. Some approaches only consider the information from the neighbors [112,114,127], while others combine the node information with its neighbors' information [114,115,123,125]. The latter method usually involves concatenation functions with non-linear transformations, which allows for more complex feature interactions. However, some recent works simplify the update operation by removing non-linearities [113,114], which increases computational efficiency while retaining or even improving performance.
- **Readout function**: Different methods are used to generate the final representations of nodes used as the GNN output. The simplest and most common approach is to use the output of the last layer as the final representation [112,121,122]. Recent studies integrate the messages from different layers to take advantage of the connections expressed by the output of different layers [128]. Mean-pooling, sum-pooling, weighted-pooling, and concatenation are examples of methods employed to integrate the messages.

GNNs have shown promising results in improving the recommendation quality of collaborative filtering. They capture complex, structured relationships in user–item interaction data and provide a flexible framework to encode direct and higher-order relationships. However, further research is needed to allow GNNs to handle dynamic graph data, increase their efficiency, and ensure their scalability [6].

5.5. Other models

Researchers have considered the impact of time in latent factor models as user preferences change over time. Xiang et al. [129] integrated temporal dynamics into Regularized Singular Value Decomposition (RSVD) [48] to enhance prediction accuracy. They identified four types of time effects within RSVD: user bias shifting, where a user may change ratings over time; item bias, indicating changes in the popularity of items; time bias, acknowledging shifts in society's interests and preferences over time; and user preference shifting, where a user may alter their opinion about some items. The authors used the MovieLens-1M and Netflix datasets, employing the RMSE evaluation metric for their experiments. The results demonstrated that TimeSVD improved prediction accuracy compared to the time-independent RSVD, achieving an RMSE of 0.836 on MovieLens-1M and 0.901 on the Netflix dataset.

Some researchers have integrated clustering methods with latent factor models to enhance recommendation performance. Zarzour et al. [130] combined Singular Value Decomposition (SVD) [28] with

the k-means algorithm [33] to cluster similar users and reduce dimensionality. Their approach uses the k-means algorithm to cluster users' ratings to determine the center-item rating matrix, which is then subjected to SVD to reduce dimensionality. Cosine similarity [17] is subsequently used to calculate similarities. For their experiments, they utilized the MovieLens-1M and MovieLens-10M datasets. They compared their model to traditional k-means using the RMSE. The results demonstrated that their approach outperformed k-means across all datasets, achieving the best RMSE of 0.620 on MovieLens-10M.

Vozalis et al. [131] combined singular value decomposition (SVD) [28] with item-based filtering to minimize the dimensionality of the user–item matrix. First, they applied the SVD on the user–item rating matrix to minimize dimensionality. Then, compute the similarity between the items using Adjacent Cosine Similarity [22] and predict the unknown ratings. They used the MovieLens-100k dataset and the Mean Absolute Error to measure the model's performance. They compared the proposed model with basic item-based filtering. The proposed model outperformed basic item-based filtering, with 0.797, compared to basic item-based filtering, with 0.840.

5.6. Discussion

This section explored advanced modeling techniques pivotal in recommendation systems, discussing probabilistic, weighted, nonlinear, kernelized, and neural models designed to tackle collaborative filtering challenges. Their goals include enhancing prediction accuracy, handling sparse data, addressing the cold-start problem, and capturing intricate user–item interaction patterns. Key findings from the reviewed papers are summarized in Table 3.

Probabilistic models provide a robust framework for large-scale, sparse datasets, with models like Probabilistic Matrix Factorization (PMF) incorporating prior knowledge to manage uncertainty in user ratings and improve understanding of user preferences.

Nonlinear models capture complex patterns that linear models may overlook but face issues like optimization challenges and scalability concerns, making them less favorable for massive datasets. Their potential to uncover hidden patterns makes them worthy of further exploration.

Kernelized models balance linear simplicity and nonlinear flexibility by introducing nonlinearities at the feature level through kernel functions. This allows them to maintain efficient optimization similar to linear models, making them useful for scenarios where linear models fail to capture the underlying complexities of the data.

Despite a trend towards sophisticated nonlinear models, especially those using neural networks, most focus on processing auxiliary data, such as textual information in hybrid models, rather than enhancing the collaborative filtering process. An exception is Graph Neural Networks (GNNs), which leverage the relational structure of data intrinsic to user–item interactions in recommendation systems. However, GNNs are still in early development, with research aimed at reducing computational overheads to improve efficiency and scalability.

A proposed link to network geometry opens the possibility of using advanced techniques developed for modeling complex networks, particularly hyperbolic geometry models, to probe the geometry of recommender system data. Embedding recommendation data into hyperbolic space can leverage geometric properties to enhance similarity measures, efficiently represent sparse data, and improve the scalability and accuracy of recommendations.

While the field is gradually shifting towards more complex and powerful neural network models for collaborative filtering, significant research efforts are still required to make these models more efficient and scalable for real-world applications, further enhancing the capabilities of recommendation systems.

The increasing complexity of recommendation models, especially with deep neural networks, has underscored the need for explainable recommendations. Recent trends focus on enhancing transparency and user trust by providing clear, interpretable reasons for recommendations. Techniques such as attention mechanisms, feature importance visualization, and natural language explanations are being integrated into recommendation algorithms [132]. These advancements not only help users understand the decision-making process but also increase satisfaction and trust. They also enable developers to diagnose and mitigate biases and errors, leading to more reliable recommendation systems. Explainable recommendation models are an exciting future research direction with the potential to enhance user experience and system robustness significantly.

6. Learning strategy

This section covers advanced learning strategies designed to enhance the performance of recommendation systems. We explore a variety of approaches, including Supervised Learning, a foundational strategy for training recommendation algorithms due to its extensive use and proven effectiveness; Self-Supervised Learning, which leverages unlabeled data through pretext tasks to improve data representation; Transfer Learning, which applies insights from one domain to improve predictions in another, effectively addressing the cold-start problem; Active Learning, which strategically queries labels for the most informative data points, optimizing the learning process in sparse data environments; and Online Learning, which continuously updates models in response to new data, ensuring adaptability and timeliness. Each technique uniquely addresses critical challenges such as data sparsity, scalability issues, and the integration of new users or items, thereby improving the overall effectiveness of recommendation systems.

6.1. Supervised learning

Supervised learning is the oldest and most widely used strategy in training recommendation algorithms. However, the recommendation task differs slightly from typical supervised learning tasks because it lacks the standard (input, output) structure commonly found in supervised learning problems.

Despite this difference, the recommendation task can still be considered a supervised learning problem for several reasons. First, observed ratings can be interpreted as labeled examples while missing ratings play the role of unseen examples [21,22]. In the absence of content data, user and item IDs serve as inputs to the algorithm. Although these IDs are not features in the traditional sense, they allow the algorithm to retrieve the corresponding learned features, p_i and q_j, which function as the actual input features to the recommendation model [28].

In terms of outputs, the recommendation task involves predicting ratings, similar to typical supervised learning tasks. This prediction task can be cast as either a classification or regression problem, depending on whether the output is discrete or continuous [15,38].

Finally, like other supervised learning tasks, recommendation algorithms are trained by minimizing a loss function that compares predicted ratings to actual ones. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.

6.2. Self-supervised learning

Self-supervised learning (SSL) has emerged as an essential strategy to address data scarcity in recent years. SSL represents a middle ground between supervised and unsupervised learning, in which the algorithm generates its own supervised signals using pretext tasks and then learns useful representations beneficial for actual tasks. The pretext tasks commonly used in SSL include data completion, where the algorithm hides part of the data (a portion of a sentence or a region of an image) and then attempts to complete the missing part. Other pretext tasks involve denoising the data or reversing transformations on image data. SSL can be broadly divided into two classes: auto-associative SSL, which

Table 3
Summary of experimental results from surveyed papers using various models.

Reference	Model Type	Datasets Used	Results
PMF [68]	Probabilistic	Netflix	Netflix: Error rate 0.886
WSVD [91]	Weighted	MovieLens-100k, MovieLens-1M, MovieLens-10M, FilmTrust	MovieLens-100k: RMSE 0.943, MovieLens-1M: RMSE 0.992, MovieLens-10M: RMSE 0.947, FilmTrust: RMSE 1.093
wSVD [92]	Weighted	MovieLens-100k, MovieLens-1M, MovieTweetings	MovieLens-100k: RMSE 0.839, MovieLens-1M: RMSE 0.801, MovieTweetings: RMSE 1.317
KMF [93]	Kernelized	MovieLens, Flixster, Jester, Yahoo Music	MovieLens: RMSE 0.816, Flixster: RMSE 0.815, Jester: RMSE 4.081, Yahoo Music: RMSE 18.503
SPHM [101]	Nonlinear	AmazonDM, AmazonIV, Anime, Book-Crossing, CiaoDVD, Epinions, FilmTrust, Food.com, ML100K, ML1M, YahooMovies, YahooMusic	AmazonDM: RMSE 0.784, MAE 0.484 AmazonIV: RMSE 1.077, MAE 0.780 Anime: RMSE 1.138, MAE 0.863 Book-Crossing: RMSE 3.416, MAE 2.767 CiaoDVD: RMSE 0.982, MAE 0.752 Epinions: RMSE 1.060, MAE 0.819 FilmTrust: RMSE 0.791, MAE 0.613 Food.com: RMSE 0.938, MAE 0.558 ML100K: RMSE 0.912, MAE 0.724 ML1M: RMSE 0.853, MAE 0.676 YahooMovies: RMSE 2.941, MAE 2.200 YahooMusic: RMSE 1.190, MAE 0.989
CELMCF [103]	Neural	MovieLens, BookCrossing, Jester, Tencent Weibo	MovieLens: MAE 0.653, Jester: MAE 0.602, BookCrossing: MAE 0.521, Tencent Weibo: MAE 0.105
TimeSVD [129]	Other	MovieLens-1M, Netflix	MovieLens-1M: RMSE 0.836, Netflix: RMSE 0.901
SVD+k-means [130]	Other	MovieLens-10M	MovieLens-10M: RMSE 0.620
SVD+Item-based [131]	Other	MovieLens-100k	MovieLens-100k: MAE 0.797

we have already discussed, involves the algorithm hiding part of the data and attempting to complete it or reconstruct the entire input. The second category is contrastive SSL, where the algorithm distinguishes between similar pairs of data points, referred to as positive, and dissimilar pairs, referred to as negative pairs. Positive pairs are generally obtained by applying a transformation or noise to a single data point, whereas negative pairs are obtained by randomly sampling from the data.

SSL is particularly useful when unlabeled data are abundant. In supervised learning, such unlabeled data is often useless, but SSL makes good use of it by learning useful representations, which can then be fine-tuned on smaller labeled datasets. SSL can be beneficial in recommender systems [7,133], where implicit feedback data are abundant in the form of clicks, views, and purchase history. This data can help build a user model that can be fine-tuned on labeled data, such as rating data.

6.2.1. Data augmentation

Data augmentation is crucial in developing self-supervised learning methods for recommender systems. Data augmentation strategies can be divided into three main categories [7], each tailored to address different aspects of SSL and its application in recommendation systems:

- **Sequence-Based Augmentation:** This involves modifying user interaction sequences to create varied learning examples [134–138]. Techniques include item masking, reordering, and substituting, which help models learn robust features by predicting the original order of items.
- **Graph-Based Augmentation:** Applied primarily in graph-based recommender systems, where a bipartite graph represents the interactions between users and items [139–144]. These methods include edge dropping, node dropping, and edge adding. Such augmentations aim to simulate variations in user–item interaction graphs, enhancing the model's ability to capture essential connectivity patterns.
- **Feature-Based Augmentation:** Focuses on altering the feature representations of users or items, such as adding noise to features or modifying feature values [145–149]. This approach is intended to make the recommendation models more adept at handling feature variability and improving their generalization capabilities.

Each of these augmentation strategies plays a crucial role in enriching the training data and enhancing the learning capacity of SSL models in recommender systems.

6.2.2. Auo-associative methods

Auto-associative SSL methods enable models to derive meaningful patterns from incomplete data, enhancing their predictive and generative capabilities. They consist of learning data representations by reconstructing entire inputs from corrupted ones or predicting missing parts of the data:

- **Structure generation:** Techniques like BERT4Rec [150] and G-BERT [151] utilize masked item prediction and graph-based reconstructions, respectively. They involve masking parts of the input (such as items in sequences or nodes in graphs) and predicting these masked parts to learn robust data representations.
- **Feature generation:** Approaches such as PMGT [152] and GPT-GNN [153] focus on regenerating missing features or entire user/item profiles from partially available data, treating the task as a regression problem.
- **Sample prediction:** Techniques include enhancing sequence recommendation models by augmenting short sequences with pseudo-prior items or using semi-supervised learning methods to improve sample quality iteratively.
- **Pseudo-label prediction:** Involves using pre-defined relations or learned continuous values as labels for training. Models predict these relations or attempt to minimize the difference between the predicted and actual values, refining user–item interaction predictions.

Generative auto-associative methods leverage the power of architectures like Transformers for large-scale pre-training but face challenges related to computational demands. Auto-associative methods based on predictive pretext tasks offer dynamic and flexible sample generation but require careful design to ensure these tasks align with user–item interaction patterns.

6.2.3. Contrastive methods

Contrastive pretext tasks derived from recommendation data augmentation approaches and data types cane be categorized into three groups [7]: structure-level contrast, feature-level contrast, and model-level contrast.

- **Structure-level contrast** utilizes user behavior data represented as graphs or sequences, exploiting slight perturbations to infer similar semantics. This is divided into same-scale and cross-scale contrasts:
  - **Local-local contrast:** Focuses on graph-based models, maximizing mutual information between node representations from two augmented views using a shared encoder and contrastive loss (InfoNCE loss [154]). Key models include SGL [140], DCL [155], and HHGR [156], which employ various node and edge dropout techniques to enhance graph representation.
  - **Global-global contrast:** Used in sequential recommendation models, where sequence augmentations are treated as global views and contrasted using a Transformer-based encoder. Notable implementations are CL4SRec [135], H^2SeqRec [137], and UniSRec [157], utilizing methods like item masking and cropping.
  - **Local–global contrast:** Aims to integrate global information into local structures, exemplified by EGLN [143] and BiGI [158], which contrast user–item pair representations against global graph representations.
  - **Local-context contrast:** Involves contrasting node or item sequences against their respective contextual clusters, with applications in models like NCL [149] and ICL [159] for capturing semantic neighbors or user intents.
- **Feature-level contrast** leverages a variety of categorical features. SL4Rec [160] and SLMRec [124] are notable for applying correlated feature masking and dropout to augment data meaningfully.
- **Model-level contrast** modifies the model architecture itself to create augmented views dynamically. Techniques include neuron masking and adjusting hidden representations, with DuoRec [145] and SimGCL [148] as key examples that enhance model robustness and mitigate popularity bias.

The contrastive loss, essential for optimizing the mutual information between representations, includes popular estimators like Jensen–Shannon and InfoNCE [154]. These losses are crucial for learning distinct representations and managing negative sampling in contrastive learning frameworks.

Despite the rapid expansion of contrastive methods in recommender systems, challenges remain such as the lack of rigorous understanding of augmentation impacts and the potential negative effects of common augmentations on model performance [7].

Self-supervised learning (SSL) significantly enhances recommender systems by enabling the learning of representations without labeled data. Innovative data augmentation techniques and advanced model architectures play a crucial role in improving the accuracy and efficiency of these systems. Despite being a relatively new field that has attracted considerable interest in recent years, SSL in recommender systems holds substantial potential and presents numerous unresolved challenges [7,133].

6.3. Active learning

Active learning is a type of machine learning that selectively queries the data source to label new data points to achieve greater accuracy using strategies such as uncertainty sampling, estimated error reduction, or density-weighted methods [161]. This approach is practical when labeled data is scarce or expensive, making it particularly useful for recommender systems settings.

Guan et al. [162] proposed the Enhanced Singular Value Decomposition (ESVD) model, which integrates the basic matrix factorization technique of Regularized Singular Value Decomposition (RSVD) [48] with a rating completion strategy inspired by active learning. The strategy involves selecting the top N most popular items and active users to obtain the densest sub-matrix, effectively reducing the sparsity problem. This densest sub-matrix is then used with the RSVD model to predict ratings, and the missing ratings in the original matrix are filled in with the ratings obtained from the sub-matrix. Finally, RSVD is applied again to the original matrix for a comprehensive rating prediction. Additionally, they extended the ESVD model to the Multilayer ESVD (MESVD), learning the model iteratively to achieve better performance. The output generated by the lower layer was used as input for the upper layer to obtain a much denser sub-matrix. All estimated ratings were filled in the original matrix to evaluate the model's performance.

The authors further proposed two extensions of the ESVD to handle imbalanced datasets. The first was Item-wise ESVD (IESVD), which selects the top N most popular items to form a sub-matrix and then chooses only the active users. The second extension was User-wise ESVD (UESVD), which selects the top N most active users to form a sub-matrix and then chooses the most popular items from this sub-matrix.

Experiments were conducted using several datasets: MovieLens-100k, MovieLens-1M, and Netflix, employing the Root Mean Square Error (RMSE) metric [39]. They compared the ESVD with MESVD, and the MESVD demonstrated a minor improvement in prediction accuracy with four layers.

6.4. Online learning

Online learning is a machine learning paradigm where data is sequentially accessible, allowing systems to update predictions progressively as new information is acquired [163,164]. This method is necessary when training over a complete dataset is computationally prohibitive or when a system needs to be deployed immediately and learn from data generated in real-time. Incremental learning ensures the system remains functional and progressively improves, providing timely and increasingly accurate responses without initial exhaustive data.

A prominent application of online learning in recommender systems is in news recommendation. News recommender systems are distinct from other recommender systems due to unique challenges such as scalability, the transient nature of news, and dynamically changing user preferences [165]. These systems must effectively manage an enormous volume of news articles available online. Unlike static content like movies, news articles have a short relevance lifespan and need to be updated or replaced frequently. Therefore, these systems are designed to handle new and trending articles efficiently while adapting to changes in user interest over time.

News article recommendations can be modeled using a contextual multi-armed bandit (MAB) framework. Each article is considered an arm in this model, and the reward is quantified by how frequently users click the article. This approach helps balance the exploration of new articles against the exploitation of previously popular articles, with the aim of maximizing user engagement through an optimal recommendation strategy. The core challenge in multi-armed bandit problems, the exploration–exploitation trade-off, requires algorithms that can continuously assess the potential of new articles against the known rewards of familiar ones [166]. The MAB problem involves

an agent making sequential choices from a set of options, each with its reward distribution, to maximize cumulative rewards. For news recommendations, this translates into monitoring and adapting to user preferences in real-time, which makes MAB approaches particularly suitable for the dynamic and voluminous nature of news consumption.

The basic MAB problem is formulated as follows. Each arm (article) in a set $A_t \in \{1, \ldots, K\}$ is associated with unknown reward distributions $\{D_1, \ldots, D_K\}$ and mean rewards $\{\mu_1, \ldots, \mu_k\}$. The agent selects an arm $a(t)$ at each time step $t = \{1, 2, \ldots\}$, and observes a reward $r_{a(t)}$. The primary goal is to minimize the total regret R_T over a predetermined number of trials T, where regret is defined as the difference between the total reward obtained and the maximum possible reward:

$$R_T = T\mu^* - \sum_{t=1}^{T} \mu_{a(t)}, \tag{52}$$

where $\mu^* = \max_{i=1,\ldots,k} \mu_i$ is the highest expected reward achievable by any arm, and $\mu_{a(t)}$ is the reward received from arm $a(t)$ at time t. This framework emphasizes the necessity to balance exploring new articles and exploiting known successful ones to effectively cater to user preferences and maximize engagement [167].

In the realm of Multi-Armed Bandits (MAB), the primary challenge is to effectively manage the **exploration-exploitation dilemma**, where the goal is to maximize rewards by exploiting known rewards or exploring new potentially rewarding actions. The strategies are categorized into context-free and contextual algorithms, each suited for different scenarios regarding the available information about the environment and the arms.

- **Context-free bandit algorithms**: make decisions based solely on the historical rewards of each action, without considering any specific attributes of the arms. Some examples of this type of algorithms are:
  - **Epsilon-greedy Algorithm:** This approach selects the best-performing arm with a probability of $1 - \epsilon$ and any other arm with a probability of ϵ. It introduces a straightforward method to balance exploiting the best arm and exploring others, though managing the value of ϵ is critical to its effectiveness [168,169].
  - **Boltzmann Exploration (SoftMax):** It selects arms based on their expected rewards adjusted by a temperature parameter τ, which regulates the level of exploration. The selection probabilities are derived from the Gibbs or Boltzmann distribution, making it more refined than Epsilon-greedy by focusing on promising arms more frequently [169].
  - **Upper Confidence Bound (UCB):** This method selects arms based on their average rewards and the uncertainty or variance in their rewards. The critical aspect of UCB is its use of an upper confidence bound to balance exploitation and exploration naturally, favoring arms that are potentially underexplored [170].
- **Contextual bandit algorithms**: use additional context or attributes to improve the accuracy and effectiveness of decisions. This can include user profiles, article descriptions, or any relevant features that might influence decision-making. Some contextual algorithms are:
  - **Epoch-Greedy:** Operates by selecting arms using a set of hypotheses about the rewards, which could be based on user features or past interactions. It alternates between exploration by randomly choosing arms and exploitation by selecting the best arm according to the current hypothesis [171,172].
  - **LinUCB:** A more sophisticated approach that uses linear regression to estimate the rewards associated with each arm's features. It updates its estimates of the arm's rewards based on observed payoffs and adjusts the exploration-exploitation balance using a confidence bound on the estimated rewards [173].

Following extensive experimentation on the Yahoo! Front Page Today Module dataset, which contains over 33 million events, the authors in [173] conclude that contextual algorithms outperform non-contextual ones in scenarios where user and content dynamics are constantly changing, such as web services. Specifically, the new contextual bandit algorithm showed a 12.5% improvement in click lift over a standard context-free bandit algorithm. This performance enhancement was even more pronounced when the data were sparser. By using additional information about the environment and entities involved, contextual algorithms can adapt more effectively to these dynamics. They provide a more personalized experience and can achieve higher performance metrics, such as click-through rates, by tailoring decisions based on the context of each situation.

6.5. Transfer learning

Transfer learning is a powerful machine learning technique that involves reusing a pre-trained model as a starting point to develop another model for a new task [174]. This approach can help improve learning efficiency and accuracy in a new task by leveraging knowledge and data from a related task that has already been mastered. Transfer learning is particularly useful when there is a shortage of data available for the new task. It is widely used in several fields, including computer vision, natural language processing, and recommender systems [175].

Traditional recommender systems, such as factorization-based collaborative filtering, require extensive training datasets to work effectively but encounter difficulties when dealing with sparse real-world data and the cold-start problem related to new users or items. Transfer learning methods can alleviate this limitation. These methods include instance-based and feature-based approaches that aim to enhance recommendations.

- **Instance-based methods** transfer different data types, such as ratings or feedback, from one domain to another to improve recommendations. For example, Pan et al. [176] use uncertain ratings from a source domain as constraints to aid in completing rating matrix factorizations in a target domain. Similarly, Hu et al. [177] employ an attentive memory network to extract and transfer helpful information from unstructured texts.
- **Feature-based methods**, on the other hand, transfer latent feature information across domains. Pan et al.'s Coordinate System Transfer (CST) [178] uses user and item features from a source domain. It applies them as constraints in a target domain to improve recommendation accuracy significantly compared to non-transfer methods.
- **Model-based methods** involve extracting common knowledge from a source domain and transferring it to a target domain. The goal is to transfer high-level rating behaviors, such as user and item clusters or memberships, which can help alleviate the sparsity problem in the target domain. Several algorithms have been proposed, including CBT [179], RMGM [180], CLFM [181], CKT-FM [182], and DSNs [183].

Further studies explore cross-domain recommendations using advanced techniques like Bayesian neural networks and deep learning frameworks for feature mapping and domain adaptation, further enhancing the effectiveness of recommender systems across varied data sparsity levels [184–186]. Most models are primarily designed to enhance their predictive capabilities by incorporating knowledge from direct user interactions, such as quiz responses, ratings with varying levels of certainty, and straightforward like/dislike feedback.

6.6. Discussion

This section reviewed essential learning strategies in recommendation systems. Supervised learning, which uses rating data and incorporates implicit feedback and content features, is the dominant paradigm for recommendation algorithms. However, its limitations in addressing data sparsity and the cold-start problem have led to the development of advanced strategies. Self-supervised learning (SSL) uses unlabeled data for learning via pretext tasks, enhancing model capability without explicit feedback. Transfer Learning applies knowledge from one domain to improve performance in another, addressing domain-specific challenges. Active learning queries labels for informative data points, optimizing resources in sparse data situations. Online learning updates models incrementally with new data, adapting to real-time changes in user preferences. Together, these strategies enhance the effectiveness and efficiency of recommendation systems, each offering unique solutions to modeling user–item interactions.

Unlike NLP and computer vision, recommendation systems face a scarcity of openly available data. The proprietary nature of recommendation data presents significant hurdles. Companies often guard this data for competitive and privacy reasons. Furthermore, recommendation data are usually highly specialized, targeting specific applications like hotel bookings, online shopping, or media streaming, complicating generalization across domains. This emphasizes the importance of transfer learning and domain adaptation in recommendation systems. Transfer learning allows models from one domain to be adapted for another, leveraging learned patterns and knowledge even when direct data transfer is impossible. Though still a new research area, transfer learning has gained attention and holds promise for overcoming data scarcity and specialization challenges in recommendation systems.

Another significant challenge in recommender systems is the reluctance of users to provide explicit ratings, which are crucial for training traditional supervised learning models. However, implicit data such as clicks or purchase histories can still be utilized for model training. SSL can effectively harness this implicit data by generating labels through pretext tasks, although challenges remain in developing a rigorous theoretical foundation and sophisticated, domain-specific data augmentation methods.

Federated learning, not covered in detail, is a decentralized approach where multiple devices train a shared model while keeping data localized [187]. This enhances data privacy and security, as the raw data never leaves the local devices. In the context of recommender systems, federated learning enables personalized recommendations without compromising user privacy [188] and offers scalability by leveraging many devices' computational power, reducing latency by minimizing data transfers.

As the field progresses, refining SSL approaches for implicit data and broadening transfer learning's scope to address data scarcity and specialization challenges will be key research areas. Furthermore, as privacy concerns and data security become increasingly critical, federated learning presents a promising research direction in developing recommendation technologies. These strategies not only promise to enhance the accuracy and effectiveness of recommendation models but also make them more adaptable and robust across various domains and scarce data environments.

7. Optimization

Optimization is vital in the development and effectiveness of latent factor models used in recommender systems. The quality of the resulting model depends heavily on the optimization algorithm used, as recommendation data is usually voluminous and complex. This complexity requires significant computational resources in terms of processing time and memory. Efficient optimization algorithms are, therefore, crucial as they allow the development, testing, and deployment of more powerful models. These advanced models can capture complex patterns within the data, ultimately improving the quality of recommendations provided to users [189–191].

7.1. Stochastic gradient descent

Gradient descent requires computing the gradient using the entire training set, which can be computationally expensive for large datasets. However, since the loss function is the summation of individual example losses, we can approximate the total loss using a subset $\mathcal{B}$ of the training set containing b examples:

$$\mathcal{B} = \{r_{u_1 i_1}, \dots, r_{u_b i_b}\} \subset \mathcal{R}. \tag{53}$$

The subset $\mathcal{B}$ is called a *mini-batch*, and its size, b, is a critical hyperparameter of the algorithm. The loss associated with the batch $\mathcal{B}$ is:

$$L^{\mathcal{B}}(\theta) = \frac{1}{b} \sum_{r_{ui} \in \mathcal{B}} \ell(\hat{r}_{ui}, r_{ui}), \tag{54}$$

and serves as a proxy for the total loss L, resulting in the objective function:

$$J^{\mathcal{B}}(\theta) = L^{\mathcal{B}}(\theta) + \lambda \Omega(\theta). \tag{55}$$

The update rule for stochastic gradient descent can then be written as:

$$\theta = \theta - \eta \nabla_\theta J^{\mathcal{B}}(\theta). \tag{56}$$

The algorithm thus obtained is called Stochastic Gradient Descent (SGD).[1] Although this change compared to gradient descent seems trivial, it has profound implications, both theoretical and practical, on the behavior of the algorithm and its effects on real applications:

- A single complete pass over the data, by sampling enough mini-batches, is called an *epoch*. Typically, several epochs are necessary for the algorithm to converge. SGD reduces the cost of each step, which means it performs several updates in a single epoch, whereas gradient descent updates the model parameters only once per epoch. For many problems, this leads to accelerated convergence [192,193], and for large datasets, the algorithm might reach an acceptable solution before passing through the entire dataset.
- Unlike the total gradient, where the objective value decreases at each step (for an appropriate choice of the learning rate), using a mini-batch results in fluctuations of the objective. This property can be beneficial in some cases as it allows the algorithm to skip shallow local minima in a way similar to simulated annealing. The noisy gradient, however, can lead to slow convergence even when the algorithm is exploring a good local minimum. In the extreme case of a single example, a small mini-batch size results in fast but very noisy updates, whereas large mini-batches reduce noise as the partial gradient aligns more with the total gradient. Large batch sizes, however, result in a higher computational cost and may lead the algorithm into shallow local minima near the initial position. The size of the mini-batch is, therefore, not only a computational parameter that controls the time and memory required to find a solution, but it is also a learning parameter that affects the quality of the solution found.
- The gradient of a mini-batch does not always vanish at a minimum, unlike the total gradient. The mini-batch estimates have a large variance, meaning that each update might point in a slightly different direction. As a result, the parameters can oscillate around the minimum instead of smoothly converging. This phenomenon could prevent the algorithm from reaching the minimum because the parameters keep oscillating indefinitely. Therefore, it is necessary to gradually decrease the learning rate, which can be achieved using a learning schedule [194] or by decay, such as linear or exponential decay.

[1] Some authors refer to the gradient descent algorithm that uses the total gradient as batch gradient descent and use the name Mini-batch Gradient Descent when the update is done using a mini-batch. They reserve the name SGD to the particular case when the batch size is 1.

The Stochastic Gradient Descent (SGD) algorithm and its variants, including momentum-based and adaptive learning rate algorithms, have been pivotal in advancing deep learning. These optimization techniques are essential for effectively training large-scale deep models on massive datasets, enabling the practical implementation and success of complex neural network architectures.

7.2. Momentum methods

SGD updates the model parameters using the gradient computed on a randomly sampled mini-batch, which is computationally efficient but introduces some challenges. Mini-batch gradients often differ from the total gradient, causing noisy updates that slow convergence. Additionally, the objective function in learning tasks often has a landscape of deep valleys with flat basins. SGD struggles in flat basins of objective function landscapes, where gradients diminish, despite performing well in steep regions.

The momentum method [195] addresses these issues by using a smoothed gradient to update the model parameters, thereby reducing noisy updates. It maintains an exponentially decaying average of the previous gradients through a variable named v (for velocity) and uses it as the descent direction:

$$v = -\eta \nabla_\theta J^{\mathcal{B}}(\theta) + \alpha v, \tag{57}$$

$$\theta = \theta + v, \tag{58}$$

where $0 \leq \alpha \leq 1$ is the momentum coefficient. A higher α (close to 1) gives more weight to past gradients, smoothing the parameters' trajectory and reducing oscillations. This can be particularly beneficial in the flat regions of the objective function, as it allows the algorithm to maintain momentum and make steady progress. Conversely, a lower α (close to 0) relies more on the current gradient, making updates more responsive to the immediate gradient but potentially increasing noise and oscillations.

Nesterov's momentum method [54], also known as Nesterov's Accelerated Gradient (NAG), improves the original momentum method by computing the gradient at the updated parameter position instead of the current one:

$$\tilde{\theta} = \theta + \alpha v, \tag{59}$$

$$v = -\eta \nabla_\theta J^{\mathcal{B}}(\tilde{\theta}) + \alpha v, \tag{60}$$

$$\theta = \theta + v, \tag{61}$$

The lookahead included in the computation of the descent direction provides additional information that helps anticipate the future position of the parameters, which reduces oscillations and speeds up convergence.

7.3. Adaptive learning rate methods

The learning rate is a crucial parameter for all gradient descent-type algorithms, particularly stochastic gradient descent ones. Inappropriate choices of the learning rate can cause slow convergence, oscillations, or even divergence. Using predefined learning schedules [194] or simple decay strategies can mitigate this issue, but these require extensive tuning and domain-specific knowledge.

Adaptive learning rate methods address these challenges by automatically adjusting the learning rate both across model parameters and throughout the optimization process. Adapting to model parameters stems from the understanding that not all parameters have the same impact on the objective function. Certain parameters may have a greater influence on the objective function due to the impact of their corresponding features on the model's output. Small changes in such parameters can lead to significant shifts in the objective value. Using the same learning rate for all parameters can thus be problematic. A small learning rate is necessary to prevent overshooting sensitive parameters, but slows convergence for less sensitive ones, while a large learning rate can cause the opposite effect.

The Adaptive Gradient (AdaGrad) algorithm [55] remedies this issue by using a different learning rate for each model parameter. It adjusts these learning rates individually by scaling them inversely proportional to the square root of the sum of all historical squared values of the gradient of their respective parameters.

$$g = \nabla_\theta J^{\mathcal{B}}(\theta), \tag{62}$$

$$r = r + g \odot g, \tag{63}$$

$$v = -\eta \frac{g}{\delta + \sqrt{r}}, \quad \text{(element-wise)} \tag{64}$$

$$\theta = \theta + v, \tag{65}$$

where r is a vector that stores the sum of squared partial derivatives of all model parameters, $\odot$ denotes element-wise multiplication, and δ is a small number used for numerical stability.

This strategy rapidly reduces the learning rate for parameters with large partial derivatives, while those with small partial derivatives experience a smaller reduction. This ensures greater progress in the more gently sloped directions of the parameter space.

Root Mean Square Propagation (RMSProp) [56] is an improvement over AdaGrad that adjusts the learning rate during the optimization process. AdaGrad is effective for convex functions but less suitable for non-convex functions with complex landscapes. In such cases, the sensitivity of the objective function to the model parameters may drastically change from one region to another. Consequently, old values of r can be misleading and hinder the algorithm's progress. RMSProp addresses this issue by giving more weight to recent gradients through the use of an exponentially weighted moving average instead of a cumulative sum:

$$g = \nabla_\theta J^{\mathcal{B}}(\theta), \tag{66}$$

$$r = \rho r + (1 - \rho) g \odot g, \tag{67}$$

$$v = -\eta \frac{g}{\delta + \sqrt{r}}, \quad \text{(element-wise)} \tag{68}$$

$$\theta = \theta + v, \tag{69}$$

where $0 \leq \rho \leq 1$ is the decay rate or smoothing factor.

Using an exponentially decaying average helps maintain efficient learning rates throughout the optimization process. The value of ρ is typically chosen empirically. Smaller ρ prioritizes recent gradients, supporting rapidly changing objectives, while larger ρ provides smoother updates, benefiting stable objectives.

Adaptive Moment (Adam) [57] is another adaptive learning rate algorithm that combines RMSProp and the momentum algorithm. It uses two momentum vectors: one for the gradient, denoted by r, and another for the squared partial derivatives, denoted by s. Before updating the model parameters, Adam performs bias-correction terms for these moments, ensuring more precise estimates, especially during the initial stages of training.

$$g = \nabla_\theta J^{\mathcal{B}}(\theta), \tag{70}$$

$$s = \rho_1 s + (1 - \rho_1) g, \tag{71}$$

$$r = \rho_2 r + (1 - \rho_2) g \odot g, \tag{72}$$

$$\tilde{s} = s / (1 - \rho_1^t), \tag{73}$$

$$\tilde{r} = r / (1 - \rho_2^t), \tag{74}$$

$$v = -\eta \frac{\tilde{s}}{\delta + \sqrt{\tilde{r}}}, \quad \text{(element-wise)} \tag{75}$$

$$\theta = \theta + v, \tag{76}$$

where $\rho_1, \rho_2 \in (0, 1)$ are smoothing factors, and t is the time step (starting from 1). When the number of steps increases, ρ_1^t and ρ_2^t tend to zero, and the bias correction practically vanishes. By combining the

strategies of the momentum algorithm and RMSProp, Adam efficiently adapts to variations in the objective function's behavior. This adaptability allows it to navigate the complex landscapes of the objective functions in large learning models, making it one of the most widely used algorithms for training latent factor models or deep neural models.

Adaptive Moment Estimation (Adam) [57] is another adaptive learning rate algorithm that combines RMSProp and the momentum algorithm. It uses two momentum vectors: one for the gradient, denoted by s, and another for the squared partial derivatives, denoted by r. Before updating the model parameters, Adam performs bias correction for these moments, ensuring more precise estimates, especially during the initial stages of training.

$$g = \nabla_\theta J^{\mathcal{B}}(\theta), \tag{77}$$

$$s = \rho_1 s + (1-\rho_1) g, \tag{78}$$

$$r = \rho_2 r + (1-\rho_2) g \odot g, \tag{79}$$

$$\tilde{s} = s/(1-\rho_1^t), \tag{80}$$

$$\tilde{r} = r/(1-\rho_2^t), \tag{81}$$

$$v = -\eta \frac{\tilde{s}}{\delta + \sqrt{\tilde{r}}}, \quad \text{(element-wise)} \tag{82}$$

$$\theta = \theta + v, \tag{83}$$

where $\rho_1, \rho_2 \in (0,1)$ are smoothing factors, and t is the time step. Note that as the number of steps increases, ρ_1^t and ρ_2^t tend to zero, and the bias correction practically vanishes. The combination of the momentum algorithm and RMSProp allows Adam to efficiently adapt to significant variations in the behavior of the objective function. This adaptability enables Adam to navigate the complex landscapes of the objective functions typically encountered in large, complex learning models, making it one of the most widely used algorithms for training large models, whether in latent factor models or deep neural networks.

The momentum method and Adam can be seen as gradient variance reduction techniques whereby the gradient is smoothed out to maintain a consistent descent direction. Other methods that have been proposed in this direction are Stochastic Average Gradient (SAG) [196] and Stochastic Variance Reduced Gradient (SVRG) [197], which both aim to reduce the variance of the gradient estimates to achieve faster and more stable convergence.

7.4. Dedicated methods

In contrast to the general optimization algorithms discussed in the previous sections, which are widely used across various machine learning problems, including neural network training, this section focuses on specialized algorithms explicitly tailored for latent factor models in recommender systems. By leveraging the specific structure and characteristics of recommender system data, these methods enhance latent factor models' computational efficiency and predictive performance.

Among these specialized methods, Alternating Least Squares (ALS) [198] is an optimization technique widely used for matrix factorization. The term "alternating" refers to the optimization process, which alternates between fixing one set of variables while optimizing another set. In the case of matrix factorization, this typically involves holding user factors constant to solve for item factors and vice versa. ALS uses the least squares approach to minimize the squared differences between observed and predicted ratings, adjusting factors to fit the data as closely as possible. To solve the optimization problem associated with the matrix factorization model:

$$\min_{\{p_u\},\{q_i\}} \frac{1}{2} \sum_{r_{ui}\in\mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2 + \frac{\lambda}{2}\left(\sum_{u=1}^{n} \|p_u\|_2^2 + \sum_{i=1}^{m} \|q_i\|_2^2\right), \tag{84}$$

ALS proceeds as follows:

1. Initialize $\{q_i\}$ randomly.
2. With $\{q_i\}$ fixed, solve for $\{p_u\}$ the following optimization problem:

$$\min_{\{p_u\}} \frac{1}{2} \sum_{r_{ui}\in\mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2 + \frac{\lambda}{2} \sum_{u=1}^{n} \|p_u\|_2^2. \tag{85}$$

   This linear least squares problem can be solved using various efficient methods [199], including direct linear algebraic methods.
3. With $\{p_u\}$ found in the previous step fixed, solve for $\{q_i\}$ the following optimization problem:

$$\min_{\{q_i\}} \frac{1}{2} \sum_{r_{ui}\in\mathcal{R}} \left(p_u^T q_i - r_{ui}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \|q_i\|_2^2. \tag{86}$$

   This is also a linear least squares problem that can be solved similarly to the previous problem.
4. Repeat Steps 2 and 3 until convergence.

Each subproblem that results from fixing either user or item factors is a simple linear least squares problem that can be solved efficiently using several efficient algorithms and highly optimized linear algebra libraries. The ability to solve these subproblems efficiently renders the overall algorithm highly efficient.

To enhance the computational efficiency of SVD++ [28], Wang et al. [200] introduced an improved Singular Value Decomposition++ (SVD++) that incorporates a Backtracking Line Search [201] in the SVD++ algorithm (BLS-SVD++). This approach accelerates SVD++ and improves prediction accuracy by employing a backtracking line search strategy to determine the optimal step size along a particular descent direction. This optimization is based on the local gradient of the objective function. Their experiments utilized the MovieLens-1M, MovieLens-10M, and FilmTrust datasets. The BLS-SVD++ model was compared with traditional SVD [28], regularized SVD (RSVD) [48], and SVD++ [28]. The results demonstrate the effectiveness of integrating a backtracking line search within the SVD++ algorithm, significantly reducing the number of iterations and enhancing prediction accuracy.

Nasiri [202] addressed the challenges of convergence speed and data sparsity by proposing a novel method for the optimization-based matrix factorization technique, which serves as a preprocessing step to initialize the latent factor matrices of users and items. The proposed method consists of two parts: First, they employed the Singular Value Decomposition (SVD) method to decompose the user–item matrix into component matrices. These matrices were then used as initial values for the latent factor matrices in the Stochastic Gradient Descent (SGD) technique [62], facilitating faster algorithm convergence. The authors conducted experiments on the MovieLens-100k dataset and used RMSE to evaluate the model's performance. They compared the performance of SGD with initialization against SGD without initialization. The results showed that initializing SGD significantly improved prediction accuracy and reduced the number of iterations required to reach a minimal error rate.

7.5. Discussion

Efficient optimization algorithms play a crucial role in developing effective latent factor models that provide high-quality recommendations to users. Stochastic Gradient Descent (SGD) is a key algorithm in this context, enabling the training of large-scale models on massive datasets. Using mini-batches in SGD allows for accelerated convergence and the exploration of good local minima, but it also introduces challenges, such as noisy updates and oscillations around the minimum. To address these issues, several variants of SGD have been proposed, including momentum-based and adaptive learning rate algorithms such as AdaGrad, RMSProp, and Adam. These techniques are fundamental for practically implementing large latent factor models, particularly those relying on deep neural networks. We also present algorithms

specifically designed for collaborative filtering, leveraging the unique structure of latent factor models to efficiently solve the associated optimization problem.

Optimization is a vast field, and this section does not cover all existing algorithms. An important class of algorithms not covered here is high-order algorithms [203], such as Conjugate Gradient [102,204], Quasi-Newton methods [205,206], and Limited-memory Broyden–Flet cher–Goldfarb–Shanno algorithm (L-BFGS) [207,208], which also play significant roles in various machine learning and recommender systems applications. High-order methods use curvature information obtained from second derivatives to accelerate convergence. They require fewer steps than first-order methods, but each step involves more complex computations and takes more time.

The main issue in optimization today is the shift from earlier models, such as Matrix Factorization, which were convex, to more complex models that often result in non-convex optimization problems. The widespread adoption of deep learning models largely drives this trend. Non-convex optimization has long been recognized as a challenging problem, with researchers identifying convexity, not linearity, as the main line separating easy and complex optimization problems. However, non-convex optimization lacks sufficient research and efficient algorithms for large-scale non-convex problems. The rise of deep learning may drive further exploration in this promising field.

A vital research direction involves analyzing the behavior of optimization algorithms with latent factor models in collaborative filtering. Recommendation data are typically sparse, high-dimensional, and noisy, presenting unique challenges for these algorithms. Analyzing their performance can yield valuable insights into their effectiveness and limitations. Understanding the strengths and weaknesses of these algorithms can help in making informed decisions to improve recommendation engine performance. Additionally, these insights can aid in designing more specialized algorithms for latent factor models in collaborative filtering. While some exist, further efforts are needed to enhance the efficiency and accuracy of recommendations.

8. Discussion

We examined the field of latent factor models by structuring previous research into a clear taxonomy that includes four key aspects: learning data, model architecture, learning strategies, and optimization techniques. This framework allowed us to highlight the strengths of various approaches and identify emerging research trends:

- **Learning Data:** Recent research highlights using contextual data to enhance collaborative filtering, focusing on implicit feedback, trust data, and content data. Implicit feedback, derived from user interactions like clicks and purchases, helps infer preferences without explicit ratings. Trust data from social networks enhances personalization and addresses cold-start problems. Content data, including product descriptions and user profiles, combines with collaborative filtering to improve recommendations. Advanced methods such as multimodal data integration and context-aware techniques factor in elements like time and location, while physiological signals offer deeper insights into user preferences.
- **Models:** Advanced modeling techniques tackle challenges in collaborative filtering, such as data sparsity. Probabilistic models like Probabilistic Matrix Factorization manage large datasets and rating uncertainties. Nonlinear models capture complex patterns despite optimization challenges. Kernelized methods balance nonlinearity and optimization benefits, while neural models are prominent in hybrid systems. Graph Neural Networks also leverage user–item relationships, though computational efficiency remains important, with a common theme across methods being the effective handling of sparse, high-dimensional data.
- **Learning Strategies:** Supervised learning remains dominant, utilizing rating data enriched by implicit feedback. However, challenges like data sparsity have led to adopting strategies such as self-supervised learning, which uses unlabeled data for model training, and transfer learning, which applies knowledge from one domain to another. Active learning focuses on acquiring informative user feedback, while online learning updates models in real-time. Federated learning trains models on user devices, ensuring privacy and reducing central data collection needs.
- **Optimization Algorithms:** Optimization techniques are critical for training latent factor models in recommendation systems. Stochastic Gradient Descent (SGD) and its variants efficiently handle large datasets, while mini-batch updates increase convergence speed at the cost of noisier gradients. Higher-order methods like Conjugate Gradient can improve convergence but add complexity. Non-convex optimization in deep learning poses challenges, as deep architectures may not guarantee global optima. Nevertheless, analyzing algorithm behavior has led to enhanced efficiency and accuracy in recommendations.

Building on these insights, we now proceed with a functional comparison of the different categories of recommendation approaches we surveyed. After that, we will discuss emerging and future research directions that could help address existing limitations and enhance the capabilities of recommender systems.

8.1. Functional comparison of approaches

We conduct a functional comparison based on the proposed typology of recommendation system approaches, following the four classification axes: learning data, model, learning strategy, and optimization algorithm. Since these axes are orthogonal, a single approach may contribute to multiple axes. For instance, a recommendation method may leverage new additional learning data while simultaneously adopting a novel learning strategy. To isolate and assess the added value of each category independently, we focus on evaluating the effect of each category of approaches on the specified comparison criteria relative to the basic matrix factorization method.

To facilitate this evaluation, we adopt a simple **increase/decr ease/neutral** scale. This scale allows us to summarize whether a given approach improves (▲), worsens (▼), or has no significant effect (–) on each criterion. The criteria considered in this comparison include:

- **Scalability**: The capacity to learn effectively from large datasets, ensuring performance improvements as data volume increases.
- **Model Size**: The size of the model in terms of the number of trainable parameters, reflecting its representational capacity.
- **Efficiency**: The computational cost and resource requirements for training and inference.
- **Robustness**: The capacity to mitigate sparsity and cold-start issues.
- **Accuracy**: The effectiveness in predicting user preferences.
- **Adaptability**: Dynamism in responding to real-time changes or new data.
- **Integration**: The extent to which additional data sources are utilized.

Using this framework, Table 4 highlights the relative strengths and trade-offs of each approach, providing a holistic view of their functional impact. Approaches that add new data sources such as implicit feedback, trust, or content, typically improve scalability, robustness, accuracy, and integration but also increase model size and reduce efficiency due to added parameters and computations. More sophisticated model architectures (such as weighted, nonlinear, or graph neural networks) tend to enlarge the parameter space and slow down training but offer gains in robustness and accuracy by capturing richer patterns. Strategies like self-supervised, active, online, or transfer learning

Table 4
Functional comparison of recommendation approaches across key criteria.

	Scalability	Model Size	Efficiency	Robustness	Accuracy	Adaptability	Integration
Learning data							
Implicit feedback	▲	▲	▼	▲	▲	–	▲
Trust	▲	▲	▼	▲	▲	–	▲
Content data	▲	▲	▼	▲	▲	–	▲
Model							
Probabilistic models	–	–	▼	▲	▲	–	–
Weighted models	–	▲	▼	▲	▲	–	▲
Kernelized and nonlinear models	–	▲	▼	▲	▲	–	–
Graph neural networks	–	▲	▼	▲	▲	–	▲
Learning strategy							
Self-supervised learning	▲	–	▼	▲	▲	▲	▲
Active learning	▲	–	▼	▲	▲	▲	–
Online learning	▲	–	▼	▲	▲	▲	–
Transfer learning	▲	–	▼	▲	▲	▲	▲
Optimization							
Momentum methods	▲	–	▲	–	▲	–	–
Adaptive learning rate methods	▲	–	▲	–	▲	–	–
Dedicated methods	▲	–	▲	–	▲	–	–

improve scalability, robustness, accuracy, and adaptability by leveraging large or continuous data streams, yet often introduce extra steps or tasks, usually reducing efficiency. Finally, advanced optimization techniques (momentum, adaptive learning rates, or dedicated methods) enhance scalability by stabilizing or accelerating convergence and can boost accuracy, although they do not generally affect model size or integration directly.

8.2. Future research directions

Recent research in learning data concentrates on integrating multiple data modalities, including text, images, and audio, to create richer representations of users and items. Deep learning techniques are increasingly being employed to process these diverse data types, enhancing the contextual awareness of recommendations. Researchers also focus on context-aware approaches that consider time, location, and device usage to provide more personalized and timely suggestions. Additionally, incorporating physiological and behavioral signals, such as EEG measurements and gaze tracking, allows for a deeper understanding of cognitive and emotional aspects of user behavior, further refining personalization strategies.

There is a growing emphasis on developing advanced nonlinear and neural network architectures that capture the complexity of user–item interactions. Graph neural networks (GNNs) and advanced geometry-based methods, such as hyperbolic embeddings, show particular promise in effectively modeling relational structures. However, these powerful models also introduce computational and scalability challenges. An important trend within this area is the push for explainable recommendation systems. Techniques like attention mechanisms and interpretability tools are being developed to provide insights into the rationale behind specific recommendations, thereby increasing user trust and transparency.

Regarding learning strategies, one key focus is self-supervised learning, which utilizes unlabeled or implicit data to address data scarcity issues. Transfer learning and domain adaptation are also gaining attention as methods to share knowledge across different domains, helping to mitigate the highly specialized nature of real-world recommendation scenarios. Meanwhile, federated learning offers a privacy-preserving, decentralized approach, allowing collaborative model training without exposing raw user data. This is particularly important as concerns about data security and user privacy intensify.

Finally, in optimization, future research aims to tackle the challenges posed by the increasing prevalence of non-convex, large-scale problems associated with deep learning-based recommendation models. While high-order optimization methods like Quasi-Newton or L-BFGS can achieve convergence in fewer steps than first-order algorithms, their computational overhead remains an ongoing research challenge. Another critical issue is the need to tailor optimization algorithms specifically for latent factor models in collaborative filtering, which often involve sparse and high-dimensional user–item matrices. Addressing these optimization complexities will lead to more efficient and accurate training processes that can keep pace with the scale and variety of modern recommender systems.

9. Conclusion

This survey significantly contributes to the understanding of latent factor models by providing a systematic review that organizes prior work into a well-defined taxonomy based on four key aspects where advancements in latent factor models have been made: learning data, model architecture, learning strategies, and optimization techniques, where advancements in latent factor models have been made. It highlights strengths, identifies trends, and points out gaps in current research while demonstrating the effectiveness of latent factor models in addressing challenges such as data sparsity and scalability in recommendation tasks.

Additionally, the analysis offers insights into how different machine-learning paradigms can enhance recommender systems and discusses potential future research directions, emphasizing the need for more robust, adaptable, and context-aware recommendation methods.

Ultimately, by surveying and categorizing existing methodologies, this survey aims to guide researchers and practitioners in developing more effective and personalized recommender systems.

CRediT authorship contribution statement

Hind I. Alshbanat: Writing – original draft. **Hafida Benhidour:** Writing – review & editing, Supervision, Project administration. **Said Kerrache:** Writing – review & editing, Writing – original draft, Supervision.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT and Grammarly to improve the readability and language of the manuscript. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research project was supported by a grant from the "Research Center of the Female Scientific and Medical Colleges", Deanship of Scientific Research, King Saud University.

Data availability

No data was used for the research described in the article.

References

[1] A. Tegene, Q. Liu, Y. Gan, T. Dai, H. Leka, M. Ayenew, Deep learning and embedding based latent factor model for collaborative recommender systems, Appl. Sci. 13 (2) (2023) 726.
[2] A. Mongia, N. Jhamb, E. Chouzenoux, A. Majumdar, Deep latent factor model for collaborative filtering, Signal Process. 169 (2020) 107366.
[3] L. Yang, K. Liu, R. Satapathy, S. Wang, E. Cambria, Recent developments in recommender systems: A survey, 2023, arXiv (Cornell University).
[4] Q. Liu, J. Hu, Y. Xiao, J. Gao, X. Zhao, Multimodal recommender systems: A survey, 2023, arXiv.org.
[5] M. Casillo, F. Colace, D. Conte, M. Lombardi, D. Santaniello, C. Valentino, Context-aware recommender systems and cultural heritage: A survey, J. Ambient Intell. Humaniz. Comput. 14 (4) (2023) 3109–3127.
[6] S. Wu, F. Sun, W. Zhang, X. Xie, B. Cui, Graph neural networks in recommender systems: A survey, 2022, arXiv:2011.02260.
[7] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, Z. Huang, Self-supervised learning for recommender systems: A survey, 2023, arXiv:2203.15876.
[8] N. Taghipour, A. Kardan, A hybrid web recommender system based on q-learning, in: Proceedings of the 2008 ACM Symposium on Applied Computing, 2008, pp. 1164–1168.
[9] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich, Recommender Systems: An Introduction, Cambridge University Press, 2010.
[10] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 734–749.
[11] J.B. Schafer, D. Frankowski, J. Herlocker, S. Sen, Collaborative filtering recommender systems, in: The Adaptive Web, Springer, 2007, pp. 291–324.
[12] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, 2013, arXiv preprint arXiv:1301.7363.
[13] P.K. Singh, P.K.D. Pramanik, P. Choudhury, A comparative study of different similarity metrics in highly sparse rating dataset, in: Data Management, Analytics and Innovation, Springer, 2019, pp. 45–60.
[14] O.C. Santos, J.G. Boticario, D. Pérez-Marín, Extending web-based educational systems with personalised support through user centred designed recommendations along the e-learning life cycle, Sci. Comput. Program. 88 (2014) 92–109.
[15] G. Adomavicius, A. Tuzhilin, Context-aware recommender systems, in: Recommender Systems Handbook, Springer, 2011, pp. 217–253.
[16] D. Billsus, M.J. Pazzani, et al., Learning collaborative information filters, in: ICML, vol. 98, 1998, pp. 46–54.
[17] D. Billsus, M.J. Pazzani, User modeling for adaptive news access, User Modeling User Adapt. Int. 10 (2) (2000) 147–180.
[18] K. Lang, Newsweeder: Learning to filter netnews, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 331–339.
[19] H. Koohi, K. Kiani, A new method to find neighbor users that improves the performance of collaborative filtering, Expert Syst. Appl. 83 (2017) 30–39.
[20] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl, Grouplens: An open architecture for collaborative filtering of netnews, in: Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, 1994, pp. 175–186.
[21] J.L. Herlocker, J.A. Konstan, A. Borchers, J. Riedl, An algorithmic framework for performing collaborative filtering, in: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999, pp. 230–237.
[22] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th International Conference on World Wide Web, 2001, pp. 285–295.
[23] H. Liu, Z. Hu, A. Mian, H. Tian, X. Zhu, A new user similarity model to improve the accuracy of collaborative filtering, Knowl.-Based Syst. 56 (2014) 156–166.
[24] P. Moradi, S. Ahmadian, A reliability-based recommendation method to improve trust-aware recommender systems, Expert Syst. Appl. 42 (21) (2015) 7386–7398.
[25] J. Salter, N. Antonopoulos, Cinemascreen recommender agent: Combining collaborative and content-based filtering, IEEE Intell. Syst. 21 (1) (2006) 35–41.
[26] C. Ju, C. Xu, A new collaborative recommendation approach based on users clustering using artificial bee colony algorithm, Sci. World J. 2013 (2013).
[27] A. Salah, N. Rogovschi, M. Nadif, A dynamic collaborative filtering system via a weighted clustering approach, Neurocomputing 175 (2016) 206–215.
[28] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[29] M. Ranjbar, P. Moradi, M. Azami, M. Jalili, An imputation-based matrix factorization method for improving accuracy of collaborative filtering systems, Eng. Appl. Artif. Intell. 46 (2015) 58–66.
[30] B. Huang, X. Yan, J. Lin, Collaborative filtering recommendation algorithm based on joint nonnegative matrix factorization, Pattern Recognit. Artif. Intell. 29 (8) (2016) 725–734.
[31] J. Bobadilla, R. Bojorque, A.H. Esteban, R. Hurtado, Recommender systems clustering using Bayesian non negative matrix factorization, IEEE Access 6 (2017) 3549–3564.
[32] X. Yuan, L. Han, S. Qian, G. Xu, H. Yan, Singular value decomposition based recommendation using imputed data, Knowl.-Based Syst. 163 (2019) 485–494.
[33] M. O'Connor, J. Herlocker, Clustering items for collaborative filtering, in: Proceedings of the ACM SIGIR Workshop on Recommender Systems, vol. 128, Citeseer, 1999.
[34] M.-H. Park, J.-H. Hong, S.-B. Cho, Location-based recommendation system using bayesian user's preference model in mobile devices, in: International Conference on Ubiquitous Intelligence and Computing, Springer, 2007, pp. 1130–1139.
[35] L. Getoor, M. Sahami, et al., Using probabilistic relational models for collaborative filtering, in: Workshop on Web Usage Analysis and User Profiling, WEBKDD'99, 1999, pp. 1–6.
[36] R. Van Meteren, M. Van Someren, Using content-based filtering for recommendation, in: Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop, vol. 30, 2000, pp. 47–56.
[37] P. Lops, M. d. Gemmis, G. Semeraro, Content-based recommender systems: State of the art and trends, in: Recommender Systems Handbook, 2011, pp. 73–105.
[38] M. Pazzani, D. Billsus, Learning and revising user profiles: The identification of interesting web sites, Mach. Learn. 27 (3) (1997) 313–331.
[39] J.L. Herlocker, J.A. Konstan, L.G. Terveen, J.T. Riedl, Evaluating collaborative filtering recommender systems, ACM Trans. Inf. Syst. 22 (1) (2004) 5–53.
[40] P. Melville, R.J. Mooney, R. Nagarajan, et al., Content-boosted collaborative filtering for improved recommendations, AAAI/IAAI 23 (2002) 187–192.
[41] D.-K. Chen, F.-S. Kong, Hybrid gaussian plsa model and item based collaborative filtering recommendation, Comput. Eng. Appl. 46 (23) (2010).
[42] T. Miranda, M. Claypool, A. Gokhale, T. Mir, P. Murnikov, D. Netes, M. Sartin, Combining content-based and collaborative filters in an online newspaper, in: Proceedings of ACM SIGIR Workshop on Recommender Systems, Citeseer, 1999.
[43] Y. Hijikata, Offline evaluation for recommender systems, J. Japan Soc. Artif. Intell. 29 (6) (2014) 658–689.
[44] M. Fu, H. Qu, Z. Yi, L. Lu, Y. Liu, A novel deep learning-based collaborative filtering model for recommendation system, IEEE Trans. Cybern. 49 (3) (2018) 1084–1096.
[45] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Application of Dimensionality Reduction in Recommender System—A Case Study, Tech. rep., Minnesota Univ Minneapolis Dept of Computer Science, 2000.
[46] X. Zhou, J. He, G. Huang, Y. Zhang, Svd-based incremental approaches for recommender systems, J. Comput. System Sci. 81 (4) (2015) 717–733.
[47] A. Paterek, Improving regularized singular value decomposition for collaborative filtering, in: Proceedings of KDD Cup and Workshop, vol. 2007, 2007, pp. 5–8.
[48] Y. Koren, Factorization meets the neighborhood: A multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 426–434.
[49] G. Guo, J. Zhang, N. Yorke-Smith, TrustSVD: Collaborative filtering with both the explicit and implicit influence of user trust and of item ratings, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, 2015.
[50] B. Yang, Y. Lei, J. Liu, W. Li, Social collaborative filtering by trust, IEEE Trans. Pattern Anal. Mach. Intell. 39 (8) (2016) 1633–1647.
[51] S. Rendle, Factorization machines, in: 2010 IEEE International Conference on Data Mining, IEEE, 2010, pp. 995–1000.
[52] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, URL http://www.deeplearningbook.org.
[53] H. Robbins, S. Monro, A stochastic approximation method, Annals Math. Stat. 22 (3) (1951) 400–407.
[54] Y.E. N., A method for solving the convex programming problem with convergence rate $o(1/k^2)$, Dokl. Akad. Nauk SSSR 269 (1983) 543–547.
[55] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (61) (2011) 2121–2159, URL http://jmlr.org/papers/v12/duchi11a.html.

[56] G.E. Hinton, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: G. Montavon, G.B. Orr, K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade, Springer Berlin Heidelberg, 2012, pp. 537–550.

[57] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, arXiv (Cornell University).

[58] Y. Koren, Factorization meets the neighborhood: A multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 426–434, http://dx.doi.org/10.1145/1401890.1401944.

[59] W. Shi, L. Wang, J. Qin, User embedding for rating prediction in SVD++-based collaborative filtering, Symmetry 12 (1) (2020) 121.

[60] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 173–182.

[61] Y. Zhang, C. Zhao, M. Chen, M. Yuan, Integrating stacked sparse auto-encoder into matrix factorization for rating prediction, IEEE Access 9 (2021) 17641–17648.

[62] S. Funk, Netflix Update: Try this At Home, 2006.

[63] H. Liu, W. Wang, Y. Zhang, R. Gu, Y. Hao, Neural matrix factorization recommendation for user preference prediction based on explicit and implicit feedback, Comput. Intell. Neurosci. 2022 (2022).

[64] H. Parvina, P. Moradi, S. Esmaeilib, M. Jalilic, An efficient recommender system by integrating non-negative matrix factorization with trust and distrust relationships, in: 2018 IEEE Data Science Workshop, DSW, IEEE, 2018, pp. 135–139.

[65] X. Luo, M. Zhou, S. Li, Z. You, Y. Xia, Q. Zhu, A nonnegative latent factor model for large-scale sparse matrices in recommender systems via alternating direction method, IEEE Trans. Neural Netw. Learn. Syst. 27 (3) (2015) 579–592.

[66] D. Kim, C. Park, J. Oh, S. Lee, H. Yu, Convolutional matrix factorization for document context-aware recommendation, in: Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 233–240.

[67] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.

[68] A. Mnih, R.R. Salakhutdinov, Probabilistic matrix factorization, Adv. Neural Inf. Process. Syst. 20 (2007).

[69] H. Wang, N. Wang, D.-Y. Yeung, Collaborative deep learning for recommender systems, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1235–1244.

[70] B. Mohd Aboobaider, et al., Word sequential using deep lstm and matrix factorization to handle rating sparse data for e-commerce recommender system, Comput. Intell. Neurosci. 2021 (2021).

[71] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1532–1543.

[72] S. Sun, Y. Xiao, Y. Huang, S. Zhang, H. Zheng, W. Xiao, X. Su, Joint matrix factorization: A novel approach for recommender system, IEEE Access 8 (2020) 224596–224607.

[73] J. Schmidhuber, S. Hochreiter, et al., Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[74] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, DeepFM: A factorization-machine based neural network for CTR prediction, 2017, arXiv preprint arXiv:1703.04247.

[75] K. Ong, K.-W. Ng, S.-C. Haw, Neural matrix factorization++ based recommendation system, F1000Research 10 (1079) (2021) 1079.

[76] F. Strub, J. Mary, P. Philippe, Collaborative filtering with stacked denoising autoencoders and sparse inputs, in: NIPS Workshop on Machine Learning for ECommerce, 2015.

[77] S. Zhang, L. Yao, X. Xu, AutoSVD++ an efficient hybrid collaborative filtering model via contractive auto-encoders, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017, pp. 957–960.

[78] R. Salah, P. Vincent, X. Muller, et al., Contractive auto-encoders: Explicit invariance during feature extraction, in: Proc. of the 28th International Conference on Machine Learning, 2011, pp. 833–840.

[79] N. Nassar, A. Jafar, Y. Rahhal, Multi-criteria collaborative filtering recommender by fusing deep neural network and matrix factorization, J. Big Data 7 (1) (2020) 1–12.

[80] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97.

[81] N. Nassar, A. Jafar, Y. Rahhal, A novel deep multi-criteria collaborative filtering model for recommendation system, Knowl.-Based Syst. 187 (2020) 104811.

[82] S. Geng, J. Tan, S. Liu, Z. Fu, Y. Zhang, Vip5: Towards multimodal foundation models for recommendation, 2023, arXiv.org.

[83] N. Hariri, B. Mobasher, R. Burke, Context-aware music recommendation based on latent topic sequential patterns, in: Proceedings of the 6th ACM Conference on Recommender Systems, ACM, 2012, pp. 131–138.

[84] S.-Y. Jeong, Y.-K. Kim, Deep learning-based context-aware recommender system considering contextual features, Appl. Sci. 12 (1) (2022) 45.

[85] D. Panda, D.D. Chakladar, S. Rana, S. Parayitam, An EEG-based neuro-recommendation system for improving consumer purchase experience, J. Consum. Behav. 23 (1) (2024) 61–75.

[86] P. Sulikowski, T. Zdziebko, et al., Gaze and event tracking for evaluation of recommendation-driven purchase, Sensors 21 (4) (2021) 1381.

[87] S. Castagnos, N. Jones, P. Pu, Eye-tracking product recommenders' usage, in: Proceedings of the Fourth ACM Conference on Recommender Systems, RecSys '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 29–36, http://dx.doi.org/10.1145/1864708.1864717.

[88] J.N. Sari, L.E. Nugroho, P.I. Santosa, R. Ferdiana, Product recommendation based on eye tracking data using fixation duration, Int. J. Inf. Technol. Electr. Eng. 5 (4) (2021) 109.

[89] A.T. Duchowski, Eye Tracking Methodology: Theory and Practice, Springer-Verlag, 2007.

[90] S. Yousefian Jazi, M. Kaedi, A. Fatemi, An emotion-aware music recommender system: Bridging the user's interaction and music recommendation, Multimedia Tools Appl. 80 (9) (2021) 13559–13574.

[91] H.-H. Chen, Weighted-SVD: Matrix factorization with weights on the latent factors, 2017, arXiv preprint arXiv:1710.00482.

[92] Y. Gu, X. Yang, M. Peng, G. Lin, Robust weighted SVD-type latent factor models for rating prediction, Expert Syst. Appl. 141 (2020) 112885.

[93] X. Liu, C. Aggarwal, Y.-F. Li, X. Kong, X. Sun, S. Sathe, Kernelized matrix factorization for collaborative filtering, in: Proceedings of the 2016 SIAM International Conference on Data Mining, SIAM, 2016, pp. 378–386.

[94] B. Schölkopf, A.J. Smola, F. Bach, et al., Learning with Kernels: Support Vector Machines, Regularization, Optimization, and beyond, MIT Press, 2002.

[95] F.R. Bach, G.R. Lanckriet, M.I. Jordan, Multiple kernel learning, conic duality, and the smo algorithm, in: Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 6.

[96] M. Gönen, E. Alpaydın, Multiple kernel learning algorithms, J. Mach. Learn. Res. 12 (2011) 2211–2268.

[97] G.R. Lanckriet, N. Cristianini, P. Bartlett, L.E. Ghaoui, M.I. Jordan, Learning the kernel matrix with semidefinite programming, J. Mach. Learn. Res. 5 (Jan) (2004) 27–72.

[98] N.D. Lawrence, R. Urtasun, Non-linear matrix factorization with gaussian processes, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 601–608.

[99] M. Seeger, Gaussian processes for machine learning, Int. J. Neural Syst. 14 (02) (2004) 69–106.

[100] S. Han, C. Qubo, H. Meng, Parameter selection in svm with rbf kernel function, in: World Automation Congress, vol. 2012, IEEE, 2012, pp. 1–4.

[101] A. Alhadlaq, S. Kerrache, H. Aboalsamh, A recommendation approach based on similarity-popularity models of complex networks, 2022, arXiv:2210.07816.

[102] W.W. Hager, H. Zhang, A new conjugate gradient method with guaranteed descent and an efficient line search, SIAM J. Optim. 16 (1) (2005) 170–192.

[103] T. Shang, Q. He, F. Zhuang, Z. Shi, Extreme learning machine combining matrix factorization for collaborative filtering, in: The 2013 International Joint Conference on Neural Networks, IJCNN, IEEE, 2013, pp. 1–8.

[104] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: A new learning scheme of feedforward neural networks, in: 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), vol. 2, IEEE, 2004, pp. 985–990.

[105] Q. Gu, J. Zhou, C. Ding, Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs, in: Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, 2010, pp. 199–210.

[106] C. Ding, T. Li, W. Peng, H. Park, Orthogonal nonnegative matrix t-factorizations for clustering, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 126–135.

[107] D. Seung, L. Lee, Algorithms for non-negative matrix factorization, Adv. Neural Inf. Process. Syst. 13 (2001) 556–562.

[108] X. Su, T.M. Khoshgoftaar, A survey of collaborative filtering techniques, Adv. Artif. Intell. 2009 (2009).

[109] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (1) (2021) 4–24.

[110] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2017, arXiv.org.

[111] G. Nikolentzos, G. Siglidis, M. Vazirgiannis, Graph kernels: A survey, J. Artificial Intelligence Res. 72 (2021) 943–1027.

[112] R. van den Berg, T.N. Kipf, M. Welling, Graph convolutional matrix completion, 2017, arXiv:1706.02263.

[113] L. Chen, L. Wu, R. Hong, K. Zhang, M. Wang, Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach, Proc. AAAI Conf. Artif. Intell. 34 (1) (2020) 27–34.

[114] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang, LightGCN: Simplifying and powering graph convolution network for recommendation, 2020, arXiv (Cornell University).

[115] J. Sun, Y. Zhang, W. Guo, H. Guo, R. Tang, X. He, C. Ma, M. Coates, Neighbor interaction aware graph convolution networks for recommendation, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1289–1298, http://dx.doi.org/10.1145/3397271.3401123.

[116] H. Wang, D. Lian, Y. Ge, Binarized collaborative filtering with distilling graph convolutional networks, 2019, arXiv.org.

[117] L. Zheng, C.-T. Lu, F. Jiang, J. Zhang, P.S. Yu, Spectral collaborative filtering, in: Proceedings of the 12th ACM Conference on Recommender Systems, RecSys '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 311–319, http://dx.doi.org/10.1145/3240323.3240343.

[118] Z. Liu, M. Lin, F. Jiang, J. Zhang, P.S. Yu, Deoscillated graph collaborative filtering, 2021, arXiv.org.

[119] J. Sun, Y. Zhang, C. Ma, M. Coates, H. Guo, R. Tang, X. He, Multi-graph convolution collaborative filtering, in: 2019 IEEE International Conference on Data Mining, ICDM, IEEE, 2019, pp. 1306–1311.

[120] X. Wang, H. Jin, A. Zhang, X. He, T. Xu, T.-S. Chua, Disentangled graph collaborative filtering, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1001–1010, http://dx.doi.org/10.1145/3397271.3401137.

[121] R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 974–983, http://dx.doi.org/10.1145/3219819.3219890.

[122] Q. Tan, N. Liu, X. Zhao, H. Yang, J. Zhou, X. Hu, Learning to hash with graph neural networks for recommender systems, in: Proceedings of the Web Conference 2020, WWW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1988–1998, http://dx.doi.org/10.1145/3366423.3380266.

[123] M. Zhang, Y. Chen, Inductive matrix completion based on graph neural networks, 2020, arXiv (Cornell University).

[124] Z. Tao, X. Liu, Y. Xia, X. Wang, L. Yang, X. Huang, T.-S. Chua, Self-supervised learning for multimedia recommendation, IEEE Trans. Multimed. 25 (2023) 5107–5116.

[125] L. Wu, Y. Yang, K. Zhang, R. Hong, Y. Fu, M. Wang, Joint item recommendation and attribute inference: An adaptive graph convolutional network approach, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 679–688, http://dx.doi.org/10.1145/3397271.3401144.

[126] X. Wang, R. Wang, C. Shi, G. Song, Q. Li, Multi-component graph convolutional collaborative filtering, Proc. AAAI Conf. Artif. Intell. 34 (4) (2020) 6267–6274.

[127] J. Ma, C. Zhou, H. Yang, P. Cui, X. Wang, W. Zhu, Disentangled self-supervision in sequential recommenders, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 483–491, http://dx.doi.org/10.1145/3394486.3403091.

[128] X. Wang, X. He, M. Wang, F. Feng, T.-S. Chua, Neural graph collaborative filtering, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 165–174, http://dx.doi.org/10.1145/3331184.3331267.

[129] L. Xiang, Q. Yang, Time-dependent models in collaborative filtering based recommender system, in: 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, vol. 1, IEEE, 2009, pp. 450–457.

[130] H. Zarzour, Z. Al-Sharif, M. Al-Ayyoub, Y. Jararweh, A new collaborative filtering recommendation algorithm based on dimensionality reduction and clustering techniques, in: 2018 9th International Conference on Information and Communication Systems, ICICS, IEEE, 2018, pp. 102–106.

[131] M.G. Vozalis, K.G. Margaritis, Applying svd on item-based filtering, in: 5th International Conference on Intelligent Systems Design and Applications, ISDA'05, IEEE, 2005, pp. 464–469.

[132] Y. Zhang, X. Chen, Explainable recommendation: A survey and new perspectives, Found. Trends Inf. Retr. 14 (1) (2020) 1–101.

[133] X. Ren, W. Wei, L. Xia, C. Huang, A comprehensive survey on self-supervised learning for recommendation, 2024, arXiv:2404.03354.

[134] K. Zhou, H. Wang, W.X. Zhao, Y. Zhu, S. Wang, F. Zhang, Z. Wang, J.-R. Wen, S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1893–1902, http://dx.doi.org/10.1145/3340531.3411954.

[135] X. Xie, F. Sun, Z. Liu, S. Wu, J. Gao, J. Zhang, B. Ding, Contrastive learning for sequential recommendation, in: The Institute of Electrical and Electronics Engineers, Inc., IEEE Conference Proceedings, The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Piscataway, 2022.

[136] M. Cheng, F. Yuan, Q. Liu, X. Xin, E. Chen, Learning transferable user representations with sequential behaviors via contrastive pre-training, in: 2021 IEEE International Conference on Data Mining, ICDM, IEEE, Piscataway, 2021, pp. 51–60.

[137] Y. Li, H. Chen, X. Sun, Z. Sun, L. Li, L. Cui, P.S. Yu, G. Xu, Hyperbolic hypergraphs for sequential recommendation, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, CIKM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 988–997, http://dx.doi.org/10.1145/3459637.3482351.

[138] Z. Liu, Y. Chen, J. Li, P.S. Yu, J. McAuley, C. Xiong, Contrastive self-supervised sequential recommendation with robust augmentation, 2021, arXiv.org.

[139] J. Yu, H. Yin, J. Li, Q. Wang, N.Q.V. Hung, X. Zhang, Self-supervised multi-channel hypergraph convolutional network for social recommendation, 2022, arXiv (Cornell University).

[140] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, X. Xie, Self-supervised graph learning for recommendation, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2021.

[141] J. Yu, H. Yin, M. Gao, X. Xia, X. Zhang, N.Q. Viet Hung, Socially-aware self-supervised tri-training for recommendation, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, KDD '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 2084–2092, http://dx.doi.org/10.1145/3447548.3467340.

[142] X. Zhou, A. Sun, Y. Liu, J. Zhang, C. Miao, SelfCF: A simple framework for self-supervised collaborative filtering, 2023, arXiv (Cornell University).

[143] Y. Yang, L. Wu, R. Hong, K. Zhang, M. Wang, Enhanced graph learning for collaborative filtering via mutual information maximization, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 71–80, http://dx.doi.org/10.1145/3404835.3462928.

[144] H. Yang, H. Chen, L. Li, P.S. Yu, G. Xu, Hyper meta-path contrastive learning for multi-behavior recommendation, in: 2021 IEEE International Conference on Data Mining, ICDM, IEEE, Piscataway, 2021, pp. 787–796.

[145] R. Qiu, Z. Huang, H. Yin, Z. Wang, Contrastive learning for representation degeneration problem in sequential recommendation, in: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, WSDM '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 813–823, http://dx.doi.org/10.1145/3488560.3498433.

[146] R. Qiu, Z. Huang, H. Yin, Memory augmented multi-instance contrastive predictive coding for sequential recommendation, 2021, arXiv (Cornell University).

[147] Y. Kalantidis, M.B. Sariyildiz, N. Pion, P. Weinzaepfel, D. Larlus, Hard negative mixing for contrastive learning, 2020, arXiv.org.

[148] J. Yu, H. Yin, X. Xia, T. Chen, L. Cui, Q.V.H. Nguyen, Are graph augmentations necessary? Simple graph contrastive learning for recommendation, 2022, arXiv (Cornell University).

[149] Z. Lin, C. Tian, Y. Hou, W.X. Zhao, Improving graph collaborative filtering with neighborhood-enriched contrastive learning, 2022, arXiv.org.

[150] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, P. Jiang, BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1441–1450, http://dx.doi.org/10.1145/3357384.3357895.

[151] J. Shang, T. Ma, C. Xiao, J. Sun, Pre-training of graph augmented transformers for medication recommendation, 2019, arXiv:1906.00346.

[152] Y. Liu, S. Yang, C. Lei, G. Wang, H. Tang, J. Zhang, A. Sun, C. Miao, Pre-training graph transformer with multimodal side information for recommendation, in: Proceedings of the 29th ACM International Conference on Multimedia, MM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 2853–2861, http://dx.doi.org/10.1145/3474085.3475709.

[153] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, Y. Sun, GPT-GNN: Generative pre-training of graph neural networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1857–1867, http://dx.doi.org/10.1145/3394486.3403237.

[154] A. van den Oord, Y. Li, O. Vinyals, Representation learning with contrastive predictive coding, 2019, arXiv (Cornell University).

[155] Z. Liu, Y. Ma, Y. Ouyang, X. Zhang, Contrastive learning for recommender system, 2021, arXiv.org.

[156] J. Zhang, M. Gao, J. Yu, L. Guo, J. Li, H. Yin, Double-scale self-supervised hypergraph learning for group recommendation, 2022, arXiv.org.

[157] Y. Hou, S. Mu, W.X. Zhao, Y. Li, B. Ding, J.-R. Wen, Towards universal sequence representation learning for recommender systems, in: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 585–593, http://dx.doi.org/10.1145/3534678.3539381.

[158] J. Cao, X. Lin, S. Guo, L. Liu, T. Liu, B. Wang, Bipartite graph embedding via mutual information maximization, in: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, WSDM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 635–643, http://dx.doi.org/10.1145/3437963.3441783.

[159] Y. Chen, Z. Liu, J. Li, J. McAuley, C. Xiong, Intent contrastive learning for sequential recommendation, in: Proceedings of the ACM Web Conference 2022, WWW '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 2172–2182, http://dx.doi.org/10.1145/3485447.3512090.

[160] T. Yao, X. Yi, D.Z. Cheng, F. Yu, T. Chen, A. Menon, L. Hong, E.H. Chi, S. Tjoa, J.J. Kang, E. Ettinger, Self-supervised learning for large-scale item recommendations, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, CIKM '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 4321–4330, http://dx.doi.org/10.1145/3459637.3481952.

[161] B. Settles, Active Learning Literature Survey, Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009, URL http://axon.cs.byu.edu/martinez/classes/778/Papers/settles.activelearning.pdf.

[162] X. Guan, C.-T. Li, Y. Guan, Matrix factorization with rating completion: An enhanced SVD model for collaborative filtering recommender systems, IEEE Access 5 (2017) 27668–27678.

[163] S. Shalev-Shwartz, Online learning and online convex optimization, Found. Trends Mach. Learn. 4 (2012) 107–194, URL https://api.semanticscholar.org/CorpusID:51730029.

[164] F. Orabona, A modern introduction to online learning, 2023, arXiv:1912.13213.

[165] L. Li, D.-D. Wang, S.-Z. Zhu, T. Li, Personalized news recommendation: A review and an experimental investigation, J. Comput. Sci. Tech. 26 (5) (2011) 754–766.

[166] D.A. Levinthal, J.G. March, The myopia of learning, Strateg. Manag. J. 14 (2) (1993) 95–112.

[167] A.K. Gupta, K.G. Smith, C.E. Shalley, The interplay between exploration and exploitation, Acad. Manag. J. 49 (4) (2006) 693–706.

[168] J. Vermorel, M. Mohri, Multi-armed bandit algorithms and empirical evaluation, in: European Conference on Machine Learning, Springer, 2005, pp. 437–448.

[169] N. Cesa-Bianchi, P. Fischer, Finite-time regret bounds for the multiarmed bandit problem, in: ICML, Citeseer, 1998, pp. 100–108.

[170] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2–3) (2002) 235–256.

[171] V. Kuleshov, D. Precup, Algorithms for multi-armed bandit problems, 2014, arXiv preprint arXiv:1402.6028.

[172] J. Langford, T. Zhang, The epoch-greedy algorithm for multi-armed bandits with side information, in: Adv. Neural Inf. Process. Syst., 2008, pp. 817–824.

[173] L. Li, W. Chu, J. Langford, R.E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Proceedings of the 19th International Conference on World Wide Web, ACM, 2010, pp. 661–670.

[174] Q. Yang, Y. Zhang, W. Dai, S.J. Pan, Transfer Learning, Cambridge University Press, 2020.

[175] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, 2020, arXiv:1911.02685.

[176] W. Pan, E. Xiang, Q. Yang, Transfer learning in collaborative filtering with uncertain ratings, Proc. AAAI Conf. Artif. Intell. 26 (1) (2021) 662–668.

[177] G. Hu, Y. Zhang, Q. Yang, Transfer meets hybrid: A synthetic approach for cross-domain collaborative filtering with text, in: The World Wide Web Conference, WWW '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 2822–2829, http://dx.doi.org/10.1145/3308558.3313543.

[178] W. Pan, E. Xiang, N. Liu, Q. Yang, Transfer learning in collaborative filtering for sparsity reduction, Proc. AAAI Conf. Artif. Intell. 24 (1) (2010) 230–235.

[179] B. Li, Q. Yang, X. Xue, Can movies and books collaborate? Cross-domain collaborative filtering for sparsity reduction, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009, pp. 2052–2057.

[180] B. Li, Q. Yang, X. Xue, Transfer learning for collaborative filtering via a rating-matrix generative model, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 617–624, http://dx.doi.org/10.1145/1553374.1553454.

[181] S. Gao, H. Luo, D. Chen, S. Li, P. Gallinari, J. Guo, Cross-domain recommendation via cluster-level latent factor model, in: Advanced Information Systems Engineering, Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 161–176.

[182] W. Pan, Z. Liu, Z. Ming, H. Zhong, X. Wang, C. Xu, Compressed knowledge transfer via factorization machine for heterogeneous collaborative recommendation, Knowl.-Based Syst. 85 (2015) 234–244.

[183] H. Kanagawa, H. Kobayashi, N. Shimizu, Y. Tagami, T. Suzuki, Cross-domain recommendation via deep domain adaptation, in: Advances in Information Retrieval, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 20–29.

[184] W. Pan, Q. Yang, Transfer learning in heterogeneous collaborative filtering domains, Artif. Intell. 197 (2013) 39–55.

[185] F.-Z. Zhuang, Y.-M. Zhou, H.-C. Ying, F.-Z. Zhang, X. Ao, X. Xie, Q. He, H. Xiong, Sequential recommendation via cross-domain novelty seeking trait mining, J. Comput. Sci. Tech. 35 (2) (2020) 305–319.

[186] F. Zhuang, J. Zheng, J. Chen, X. Zhang, C. Shi, Q. He, Transfer collaborative filtering from multiple sources via consensus regularization, Neural Netw. 108 (2018) 287–295.

[187] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19.

[188] Z. Sun, Y. Xu, Y. Liu, W. He, L. Kong, F. Wu, Y. Jiang, L. Cui, A survey on federated recommendation systems, IEEE Trans. Neural Netw. Learn. Syst. PP (2024) 1–15.

[189] S. Ruder, An overview of gradient descent optimization algorithms, 2017, arXiv.org.

[190] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective, IEEE Trans. Cybern. 50 (8) (2020) 3668–3681.

[191] R.-Y. Sun, Optimization for deep learning: An overview, J. Oper. Res. Soc. China 8 (2) (2020) 249–294.

[192] A. Nemirovski, A. Juditsky, G. Lan, A. Shapiro, Robust stochastic approximation approach to stochastic programming, SIAM J. Optim. 19 (4) (2009) 1574–1609.

[193] A. Agarwal, P.L. Bartlett, P. Ravikumar, M.J. Wainwright, Information-theoretic lower bounds on the oracle complexity of stochastic convex optimization, IEEE Trans. Inform. Theory 58 (5) (2012) 3235–3249.

[194] C. Darken, J. Chang, J. Moody, Learning rate schedules for faster stochastic gradient search, in: Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, IEEE, 1992, pp. 3–12.

[195] B. Polyak, Some methods of speeding up the convergence of iteration methods, USSR Comput. Math. Math. Phys. 4 (5) (1964) 1–17, http://dx.doi.org/10.1016/0041-5553(64)90137-5, URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

[196] N.L. Roux, M. Schmidt, F. Bach, A stochastic gradient method with an exponential convergence rate for finite training sets, 2013, arXiv.org.

[197] R. Johnson, T. Zhang, Accelerating stochastic gradient descent using predictive variance reduction, in: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 26, Curran Associates, Inc., 2013, URL https://proceedings.neurips.cc/paper_files/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Paper.pdf.

[198] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 263–272, http://dx.doi.org/10.1109/ICDM.2008.22.

[199] C.L. Lawson, Solving least squares problems, in: Prentice-Hall Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, N.J, 1974.

[200] S. Wang, G. Sun, Y. Li, Svd++ recommendation algorithm based on backtracking, Information 11 (7) (2020) 369.

[201] C.W. Royer, S.J. Wright, Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization, SIAM J. Optim. 28 (2) (2018) 1448–1477.

[202] M. Nasiri, B. Minaei, Increasing prediction accuracy in collaborative filtering with initialized factor matrices, J. Supercomput. 72 (6) (2016) 2157–2169.

[203] J. Nocedal, S.J. Wright, Numerical optimization, in: Fundamental Statistical Inference, 2018, URL https://api.semanticscholar.org/CorpusID:189864167.

[204] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand. 49 (1952) 409–435, URL https://api.semanticscholar.org/CorpusID:2207234.

[205] C.G. Broyden, The convergence of a class of double-rank minimization algorithms, IMA J. Appl. Math. 6 (3) (1970) 222–231.

[206] R. Fletcher, A new approach to variable metric algorithms, Comput. J. 13 (3) (1970) 317–322.

[207] J. Nocedal, Updating quasi-Newton matrices with limited storage, Math. Comput. 35 (151) (1980) 773–782.

[208] D.C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Math. Program. 45 (1) (1989) 503–528.