Recommender Systems

Shuai Zhang, Aston Zhang, and Lina Yao

1 Introduction to Recommender Systems

Recommender system (RS) seeks to estimate and predict users' preferences for products/services by filtering from a huge pool of information base with patterns/rules discovered from data usage history [23]. It is at the core of many online services such as social networking sites, video streaming services, and online shopping sites we interact with. For example, Amazon uses recommendation engines to suggest products or goods that users might buy [30]. YouTube tailors the recommendation video list based on users' tastes [22]. Facebook recommends friends and posts that might interest end users. Recommendation is also at the core of the Netflix products. To improve the recommendation quality, it hosted the famous Netflix Prize competition [31], which popularized the research on recommender systems. Microsoft also uses recommender systems to enhance the user experience while using XBox [32]. It is seen that recommender systems are now pervasive in our daily life, and its importance cannot be overemphasized.

The prevalence of recommender systems is mainly owning to the enormous collection of options provided by online platforms. However, with the abundant resources available comes the information overload problem. This is when recommender systems come into play. By drawing and learning from huge datasets, the system can capture users' interests so as to pinpoint their preferred items accurately. The benefits of employing recommender systems are manifold: on the one hand, it can enhance customer satisfaction/delight and improve engagement by providing

S. Zhang (✉) · L. Yao
University of New South Wales, Sydney, Australia
e-mail: shuai.zhang@inf.ethz.ch; lina.yao@unsw.edu.au

A. Zhang
Amazon, Bellevue, WA, USA
e-mail: az@astonzhang.com

L. Rokach et al. (eds.), *Machine Learning for Data Science Handbook*,
https://doi.org/10.1007/978-3-031-24628-9_28

personalized recommendations based on their preferences and interests. On the other hand, it has been proven to be an effective tool to drive high conversion rate and customer retention rate, as a result, leading to revenue increase.

1.1 Concepts and Notations

Some important terms that are widely used throughout this chapter are listed and explained below.

Items/Users Items refer to the objects that are recommended. It can be movies, songs, games, news, books, blogs, etc. Users are the people to whom the items are recommended.

Explicit/Implicit Feedback Users give feedback on items to express their preferences and interests. Explicit feedback directly show how a user rates an item, such as rating scores from 1 to 5. Implicit feedback such as clicks, watches, and purchase only serve as a proxy that provides us heuristics about how a user likes an item.

Collaborative Filtering The system Tapestry first coined the term collaborative filtering [39]. Since then, it has been widely used to represent recommender systems. The basic idea of CF is that users will act on items similarly if they have similar behaviors in the past. Most recommendation methods such as neighborhood methods and factorization-based methods that utilize the collaborative filtering idea can be called as collaborative filtering techniques.

The goal of recommender systems is to recommend items to users, from which many tasks are derived. We can formulate the recommendation problem as a regression, classification, ranking, or even sequence modeling problem. For example, estimating the exact rating a user might give to an item is a regression problem; predicting whether an item will be clicked or not belongs to the classification category; generating a ranked list of items for a user can be solved as a learning to rank problem; sequence modeling models come into play if we need to take the sequential patterns of user behaviors into account.

In formal, suppose we have a corpus of M users and N items, which forms an interaction matrix or utility matrix $X \in \mathbb{R}^{M \times N}$. Let $\mathcal{U}$ denote the user set and $\mathcal{I}$ denote the item set. In this matrix, rows correspond to users and columns to items. Generally, this matrix is very sparse, and each entry of this matrix displays users' feedback such as ratings or like/dislikes to the items. Let X_{u*} denote the preferences of user u toward all items. Let X_{*i} denote the feedback (ratings) from all users for item i. These notations will be used throughout the chapter.

2 Recommendation Techniques

In this section, we will introduce some widely used recommendation techniques, including classic solutions and recent advances, and discuss their advantages and weakness.

2.1 Non-personalized and Lightly Personalized Recommendations

A non-personalized recommendation approach that makes the same recommendations for all users is the most basic form of recommender system. Despite its non-personalization, it can be remarkably effective for cases such as common displays in online communities (e.g., reddit) or recommendations for new users for whom we know nothing about. Lightly personalized recommender systems refer to methods that utilize limited information such as user profiles to infer their interests roughly so as to make weakly personalized recommendations.

Recommending based on item popularity is one of the most widely used non-personalized recommendation approaches. We can identify the most popular items by counting the number of likes/views/purchases, etc. If explicit ratings are available, we can also rank the items based on the mean of the ratings (e.g., top rated movies in IMDB). This method is simple yet computationally efficient. It is worth noting that several settings can influence the recommendation performance largely, such as the period for which the popularity is calculated and the interaction types taken into consideration.

Lightly personalized recommendation is a small step toward personalization that could loosely personalize the recommendation list based on certain types of side information such as demographics. The motivation behind it is that users' preferences can be vaguely identifiable with their profiles such as age, gender, race/ethnicity, financial status, location, etc. It is straightforward to break down summary statistics by demographic. For example, tastes on movies can be quite different for people in different ages and stages. Obviously, getting the data about users is critical in this method. As such, it is common to see that some online platforms require new users to take a survey before accessing the services.

Notwithstanding the usefulness of non-personalized or lightly personalized recommendations, their demerits are conspicuous, that is, the recommended items may not satisfy user's interest. That is also why personalized recommender systems start to arise. The following text will be centered on personalized recommendation.

2.2 Neighborhood Methods

There are two standard nearest neighborhood recommendation algorithms: user-based collaborative filtering and item-based collaborative filtering [1].

2.2.1 User-Based Collaborative Filtering

User-based CF aims to find the users who have similar taste (neighbors) as the target user and then recommends items based on the neighbor's interaction behavior. The similarity calculation is based on interaction behaviors. Various similarity measures such as cosine similarity, Pearson's correlation, and Jaccard similarity are viable. Formally, let $sim(X_{u*}, X_{v*})$ denote the similarity between user u and user v. The cosine similarity is defined as below:

$$sim(X_{u*}, X_{v*}) = \frac{X_{u*} \cdot X_{v*}}{\parallel X_{u*} \parallel \parallel X_{v*} \parallel}. \tag{1}$$

Pearson correlation is used to find the linear correlation between two vectors, ranging from -1 to $+1$, with -1 indicating negative relation, 0 representing no relation, and +1 representing high positive correlation. It is defined as

$$sim(X_{u*}, X_{v*}) = \frac{\sum_{i=1}^{N}(X_{ui} - \bar{X_{u*}})(X_{vi} - \bar{X_{v*}})}{\sqrt{\sum_{i=1}^{N}(X_{ui} - \bar{X_{u*}})^2 \sum_{i=1}^{N}(X_{vi} - \bar{X_{v*}})^2}}, \tag{2}$$

where $\bar{X_{u*}}$ represents the average rating of user u.

Afterward, it selects the top K similar users and takes the weighted average of recommendation scores from these K users. To avoid user bias that some users tend to give high scores and some tend to give low scores, users' average ratings are considered. As such, the predicted score is calculated as follows:

$$X_{uj} = \bar{X_{u*}} + \frac{\sum_{k=1}^{K} sim(X_{u*}, X_{k*})(X_{kj} - \bar{X_{k*}})}{\sum_{k=1}^{K} sim(X_{u*}, X_{k*})}. \tag{3}$$

2.2.2 Item-Based Collaborative Filtering

Item-based collaborative filtering applies the same idea. Instead of computing the users similarity, it considers items similarity. Let $sim(X_{*i}, X_{*j})$ denote the similarity between item i and item j. Intuitively, the similarity is measured by observing all the users who have rated both the items. The prediction function of item j for target user u is as follows:

$$X_{uj} = \frac{\sum_{k=1}^{K} sim(X_{*j}, X_{*k}) X_{uk}}{\sum_{k=1}^{K} sim(X_{*j}, X_{*k})}. \tag{4}$$

Item-based CF is more stable and faster in system where there are more users than items. Item similarity matrix can usually be calculated offline as the ratings received by an item do not change quickly (e.g., a recognized good movie usually gets higher rating scores). So it does not need to be recomputed frequently.

2.3 Factorization-Based Methods

Factorization-based approaches (or latent factor models) aim to factorize the interaction matrix with either explicit ratings or implicit feedback into low-dimensional rectangular matrices. These methods enjoy higher flexibility and efficiency than neighborhood-based algorithms.

2.3.1 Matrix Factorization

Matrix factorization [2] method decomposes the user–item interaction matrix into two lower-dimensional matrices for users and items, respectively.

Let $P \in \mathbb{R}^{M \times k}$ represent the user matrix and $Q \in \mathbb{R}^{N \times k}$ represent the item matrix. Each row of P represents the latent factors for describing user's interests and preferences. Each row of the item matrix Q describes items' characteristics. The core idea of matrix factorization is to approximate the interaction matrix with the inner product of P and Q:

$$X \approx P Q^T. \tag{5}$$

To learn the user and item matrices, we can minimize the following mean squared error (MSE) if the goal is to recover the explicit ratings:

$$\min \sum_{(u,i) \in \mathcal{K}} (X_{ui} - P_u Q_i^T)^2 + \|P\|_F^2 + \|Q\|_F^2, \tag{6}$$

where $\mathcal{K}$ is the observed ratings. The last two terms are used to regularize the model parameters. This optimization problem can be efficiently solved with methods such as stochastic gradient descent.

For implicit feedback, a pairwise loss Bayesian personalized ranking (BPR) loss [6] can be used. The BPR loss is defined as follows:

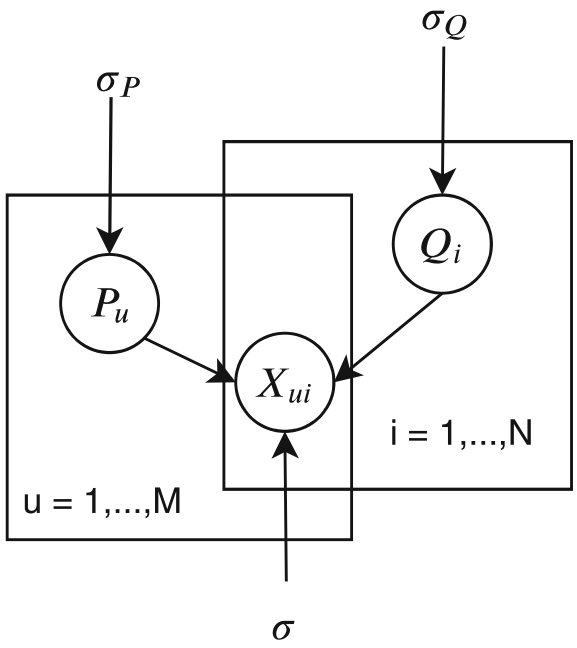

Fig. 1 Graphical model for probabilistic matrix factorization

$$\min \left(- \sum_{(u,i,j) \in \mathcal{K}} \ln \left(\sigma \left(P_u Q_i^T - P_u Q_j^T \right) \right) + \| P \|_F^2 + \| Q \|_F^2 \right). \tag{7}$$

In this loss function, $\mathcal{K}$ is composed of both observed and unobserved feedback. Here, i denotes the item that u likes and j is the item that u has never interacted with.

We have discussed the biases of user preferences in neighborhood methods. These biases should also be captured in matrix factorization. To this end, we can rewrite the scoring function as

$$X_{ui} \approx P_u Q_i^T + b_i + b_u + \mu, \tag{8}$$

where μ is the overall average rating; b_u and b_i indicate the observed deviations of user u and item i. The objective function should be reformulated accordingly.

Owing to the flexibility of matrix factorization method, additional input sources such as implicit feedback, temporal dynamics (SVD++ [3]), and social networks (SoRec [4]) can also be integrated.

The matrix factorization techniques can also be interpreted probabilistically [5]. For example, we can model the rating as a distribution parametrized by item and user latent features (Fig. 1). Assuming ratings are normally distributed:

$$p(X|P, Q, \sigma^2) = \prod_{i=1}^{M} \prod_{j=1}^{N} \left[\mathcal{N} \left(X_{ij} | P_i Q_j^T, \sigma^2 \right) \right]^{I_{ij}}, \tag{9}$$

where the mean is determined by user and item latent factors, and the variance σ^2 is used to model the noise of ratings. We define the indicator I_{ij} to be 1 if X_{ij} is known (i.e., user i has rated movie j) and 0 otherwise. We assume that users and items follow the zero-mean normal distribution with spherical Gaussian priors.

$$p(P|\sigma_P^2) = \prod_{i=1}^{M} \mathcal{N}(P_i|0, \sigma_P^2\mathbf{I}),\ p(Q|\sigma_Q^2) = \prod_{i=1}^{N} \mathcal{N}(Q_i|0, \sigma_Q^2\mathbf{I}). \tag{10}$$

This probabilistic model can be solved with expectation maximization (EM) or gradient descent algorithms. It is worth noting that eventually the optimization process of EM is identical to MSE minimization.

2.3.2 Factorization Machines

Factorization machine, as a generic method, can be used for regression, classification, and ranking tasks. It is essentially an extension of the matrix factorization algorithm and is powerful in dealing with large-scale sparse datasets and automatically modeling the feature interactions. As such, it has been widely used in fields such as products/advertisements recommendations and click-through predictions.

Let $x^{(i)} \in \mathbb{R}^D$ represent the feature vector and $y^{(i)}$ indicate the corresponding target. $x^{(i)}$ can be comprised of the one-hot representations of user/item identities and many other features such as user profiles, latest rated movies by the user, and so on. Generally, the input feature size can be very large and sparse. Label $y^{(i)}$ can represent the exact rating that the user gave to the item or a binary label indicating whether the item is clicked/liked/bought by the user or not.

Theoretically, an FM can model high degree of feature interactions, but 2-way FM is usually employed for efficiency and stability concerns. The scoring function of a 2-way factorization machines is as follows:

$$\hat{y} = w_0 + \sum_{j=1}^{D} \mathbf{w}_i x_i + \sum_{i=1}^{D} \sum_{j=i+1}^{D} < V_i, V_j > x_i x_j, \tag{11}$$

where $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^D$, and $V \in \mathbb{R}^{D\times k}$ are the model parameters to be learned. Same as matrix factorization, k is the dimension of latent factors. $< \cdot, \cdot >$ denotes the dot product of vectors. This model will degrade to matrix factorization when x_i only contains the user and item one-hot identifiers.

The computation complexity of last term of Eq. 11 in a straightforward way is $O(kD^2)$, which is very expensive. Fortunately, the computation time can be reduced to linear time $O(kD)$ by expanding and reorganizing as follows:

$$\begin{aligned}
&\sum_{i=1}^{D}\sum_{j=i+1}^{D} < V_i, V_j > x_i x_j \\
&= \frac{1}{2}\sum_{i=1}^{D}\sum_{j=1}^{D} < V_i, V_j > x_i x_j - \frac{1}{2}\sum_{i=1}^{D} < V_i, V_i > x_i x_i \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{D} V_{j,f} x_i\right)\left(\sum_{j=1}^{D} V_{j,f} x_j\right) - \sum_{i=1}^{D} V_{i,f}^2 x_i^2\right) \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{D} V_{i,f} x_i\right)^2 - \sum_{i=1}^{D} V_{i,f}^2 x_i^2\right).
\end{aligned} \tag{12}$$

By doing so, the complexity is linear to the number of nonzero elements for sparse inputs. For model training, a variety of loss functions such as MSE, cross-entropy loss, and BPR loss [6] are viable.

2.3.3 Collaborative Metric Learning

Both MF and FM model the interactions between users and items with inner product. However, inner product does not satisfy the triangle inequality, which might limit the expressiveness of the recommendation models. To alleviate the issue, researchers explore using distance functions (e.g., Euclidean distance [7], hyperbolic distance [40]) to replace the inner product. In the inference stage, items that are close to the user are recommended.

Collaborative metric learning [7] is such a representative model. It assumes that the positions of users and items are represented by $P \in \mathbb{R}^{M\times k}$ and $Q \in \mathbb{R}^{N\times k}$, and the distance between user u and item i is measured by

$$d(u, i) = \| P_u - Q_i \|_2^2. \tag{13}$$

A max-margin triplet loss is usually used for model optimization. The goal of the loss is to ensure the distance between a user and the item she likes to be smaller than that between the user and the item that she dislikes.

$$L = \sum_{(u,i)\in\mathcal{S}} \sum_{(u,j)\in\mathcal{S}'} \max(0, d(u, i) + \lambda - d(u, j)), \tag{14}$$

where set $\mathcal{S}$ is made of users and their liked items and set $\mathcal{S}'$ contains users and their disliked items. Regularization (e.g., norm clipping) is usually used on P and Q to prevent the data points spread too widely.

2.4 Modeling Sequences in Recommendation

Intuitively, there usually exist sequential patterns in user behaviors and interaction trajectories. Users usually have long-term and short-term interests. So far, we only consider users' long-term taste and all short-term preferences are ignored. However, users' short-term intents play a critical role in users' decisions [8]. For example, if a user bought a digital single lens reflex (DSLR), she will probably buy a camera Lens shortly. Knowing this pattern is important for making satisfying recommendations. The capability to simultaneously model both long-term and short preferences is the key to sequence-aware recommendation models.

Suppose each user u is associated with a sequence of items $S^u = (S_1^u, \dots S_{|S^u|}^u)$, where S_t^u represents the item user u interacted with at time step t that does not need to be absolute time but just an indicator of sequence order. The goal of sequence-aware recommendation is to predict the next item that a user will interact with.

The model we will introduce is a variant of collaborative metric learning, called personalized ranking metric embedding (PRME) [9], which considers both user general and transient intents. Let $Q \in \mathbb{R}^{N \times k}$ denote item embeddings. To model the transient interest, the model aims to make adjacent items in the sequence close to one another. Therefore, the following distance shall be minimized:

$$d(S_{t-1}^u, S_t^u) = \| Q_{S_{t-1}^u} - Q_{S_t^u} \|_2^2, \tag{15}$$

where S_{t-1}^u and S_t^u are adjacent items and S_t^u is the target item. The motivation behind this is that if two items are interacted subsequent, they are more likely to be similar.

The general taste module has the same form as collaborative metric learning. The goal is to minimize the distance between user u and the target item.

$$d(u, S_t^u) = \| P_u - V_{S_t^u} \|_2^2, \tag{16}$$

where $P \in \mathbb{R}^{M \times k}$ is the user embeddings and $V \in \mathbb{R}^{N \times k}$ is the item embeddings.

The final recommendation score is determined by the weighted summation of these two distances:

$$\omega \cdot d(S_{t-1}^u, S_t^u) + (1 - \omega) \cdot d(u, S_t^u), \tag{17}$$

where ω determines the proportion of contributions of short-term and long-term interests.

2.5 Neural Architectures for Recommender Systems

In recent years, deep neural networks have achieved tremendous success in a number of fields such as computer vision, natural language processing, speech recognition, and so on [10]. A number of deep learning techniques such as convolutional neural networks, recurrent neural networks, generative adversary networks, graph neural networks, attention networks, and deep reinforcement learning are gaining popularity in both industry and academia.

In the meantime, deep neural networks have been revolutionizing the recommendation structures as well [11]. A large amount of deep-learning-based recommendation architectures are proposed these years. It has also been demonstrated to be especially useful in real-world recommendation scenarios, and a number of companies are building their recommender systems with deep neural networks. The major advantages are: First, deep learning is capable of modeling complex data and learning expressive and high-level representations. Second, deep neural networks are advantageous in modeling sequence data and capturing the hidden sequential patterns. Third, deep neural networks can be trained end-to-end, and they have good composability and flexibility, making the design of more powerful joint models possible. In this section, we will introduce some recent advancements on deep neural-networks-based recommender systems. It is worth noting that this section is highly relevant to the content-based recommender systems that will be introduced later.

2.5.1 From Linear to Nonlinear Recommendation Models

Methods such as MF and FM use linear transformation to model the feature interactions (e.g., interaction between user latent vector and item latent vector). However, the patterns hidden in the interaction data might be extreme complex and intricate. Using nonlinear neural networks can capture the interaction patterns more easily. Here we introduce several popular models that implement this idea.

Neural Collaborative Filtering [12]. To enrich the model expressiveness, this model consists of a multilayered perceptron (MLP) and a generalized matrix factorization. Like matrix factorization, it uses latent vectors to represent each user/item. Formally, let $P \in \mathbb{R}^{M \times k}$ and $U \in \mathbb{R}^{M \times k}$ denote user latent embeddings, and use $Q \in \mathbb{R}^{N \times k}$ and $V \in \mathbb{R}^{N \times k}$ to represent each item.

The input of MLP is the concatenation of P_u and Q_i:

$$\begin{aligned} h_1 &= \alpha_1(W_1 \cdot [P_u, Q_i] + b_1) \\ &\cdots \\ h_{\ell-1} &= \alpha_{\ell-1}(W_{\ell-1} \cdot h_{\ell-2} + b_{\ell-1}) \\ h_\ell(u, i) &= \alpha_\ell(W_\ell \cdot h_{\ell-1} + b_\ell), \end{aligned} \tag{18}$$

where $[\cdot, \cdot]$ denotes the concatenation operation. ℓ is the depth of the MLP. W_*, b_*, and α_* are weight, bias, and activation function. $h_\ell(u, i)$ is the output of the MLP. This component is mainly used to model the complex and nonlinear interactions between users and items.

The input for the generalized MF component is the entry-wise (Hadamard) product of user and item latent factors, and it is defined as

$$o(u, i) = \alpha(W \cdot (U_u \odot V_i)). \tag{19}$$

Afterward, the outputs of two components are concatenated and transformed with a nonlinear layer to get the final prediction score.

$$\hat{X_{ui}} = \alpha(W[h_\ell(u, i), o(u, i)]). \tag{20}$$

The model can be trained with commonly used MSE, BPR loss, or the cross-entropy loss.

Autoencoder for recommendation. An autoencoder is a feed-forward neural network that codes its input to output while learning a hidden representation in the bottleneck layer. It is a useful dimensionality reduction model. It can also be used to reconstruct the interaction matrix. Here, we introduce two models (AutoRec [13] for rating prediction and CDAE [14] for ranking with implicit feedback).

Similar to neighborhood methods, AutoRec can be either user-based or item-based. The input of the item-based AutoRec is the column of the rating matrix. The model consists of the following encoder and decoder:

$$\begin{aligned} Encoder &: h = \alpha_1(W_1 X_{*i} + b_1) \\ Decoder &: o = \alpha_2(W_2 h + b_2), \end{aligned} \tag{21}$$

where W_*, b_*, and α_* are weight, bias, and activation function. o has the same dimensionality as X_{*i}.

The loss function of AutoRec aims to minimize the following reconstruction error:

$$\underset{W_*, b_*}{\operatorname{argmin}} \sum_{i=1}^{M} \|X_{*i} - o\|_O^2 + \lambda(\|\mathbf{W}*\|_F^2), \tag{22}$$

where $\|\cdot\|_O$ means that only observed ratings are contributed to the gradient backpropagation. With partial observed columns as input, it targets at reconstructing the entire columns. The user-based AutoRec is similar to the item-based AutoRec, but it uses rows instead of columns of the rating matrix as input.

CDAE also employs an autoencoder framework, but it is designed for recommendation with implicit feedback. Simply put, the model architecture is defined as

$$o = \alpha_2(W_2 \cdot \alpha_1(W_1 X_{u*} + U_u + b_1) + b_2), \tag{23}$$

where W_*, b_*, and α_* are weight, bias, and activation function. $U_u \in \mathbb{R}^{M \times k}$ is a user-specific bias. The loss function of CDAE is

$$\min_{W_*, U_*, b_*} \sum_{i=1}^{N} \ell(X_{u*}, o) + \lambda(\|\mathbf{W}*\|_F^2 + \|\mathbf{U}*\|_F^2), \tag{24}$$

where ℓ can be MSE or logistic loss. It is worth noting that instead of masking all unobserved input such as AutoRec, CDAE allows sampled unobserved feedback as input.

2.5.2 Representation Learning with Neural Architectures

Deep neural networks are powerful feature representation tools. They map raw features with a number of neural layers and get an abstraction of the input features in either supervised or unsupervised manner.

Multilayer perceptron is an effective tool for feature representation learning in recommender systems. The model **Wide & Deep** learning [15] proposed by Google is a good example. This model has shown good performance in Google play app store. This model consists of a wide component and a deep component. The wide component is a linear regression model that is helpful for memorization of feature interactions (e.g., co-occurrence of items), while the deep component a multilayer perceptron that could generalize to unseen feature combinations through low-dimensional dense embeddings.

In formal, the input is split into two parts: one for the wide network and the other for the deep network. We denote them with x^{wide} and x^{deep}, respectively. Same as FMs, the features are sparse. For the deep component, we let $V \in \mathbb{R}^{D \times k}$ denote the dense embeddings for the sparse feature inputs x^{deep}. For simplicity, we assume that the input features are made of m fields. After looking up from V with x^{deep} and concatenation, we get

$$h(V, x) = [e_1, e_2, \ldots, e_m]. \tag{25}$$

It is used as the input of the deep component. The final scoring function is defined as

$$\hat{y} = \sigma(W * x^{\text{wide}} + f_{MLP}(h(V, x^{\text{deep}})) + b), \tag{26}$$

where f_{MLP} is the MLP network.

DeepFM [16] replaces the linear part of wide and deep model with factorization machines. Even though linear model is effective for memorization, it is not capable of model direct feature interactions. As introduced in earlier section, FM can model 2-way interactions efficiently. Using FM as a replacement of the wide part enables

it to explicitly model feature interactions but will not incur additional computation cost.

Using DeepFM, the explicit split of features into two parts is no longer necessary, which could extensively reduce the efforts in feature engineering. The scoring function of DeepFM is

$$\hat{y} = \sigma(\hat{y}_{FM}(x) + f_{MLP}(h(V, x))). \tag{27}$$

Item2Vec [33]. Neural networks can also be used for item representations learning. Barkan et al. [33] proposed item2vec to learn item representations in a similar way to the word2vec approach [34]. In item2vec, items can be viewed as words, and the sequences of items a user liked can be seen as sentences. In doing so, the same skip-gram with negative sampling algorithm as that of word2vec can be utilized for item embedding learning.

2.5.3 Sequence-Aware Recommendation with Neural Networks

We introduced the concept of sequence-aware recommendation with a representative model PRME. However, the sequence length in PRME is merely one. With neural networks, we can model longer sequences. Let L denote the length of the sequence. The L embedding vectors form a matrix:

$$E^{(u,t)} = \begin{bmatrix} Q_{S^u_{t-L}} \\ \cdots \\ Q_{S^u_{t-1}} \end{bmatrix}. \tag{28}$$

To learn patterns from this matrix, a number of neural architectures are viable, such as RNN, CNN, and attention networks. In this section, we will introduce a self-attention-based sequential recommendation method [18].

There are three important concepts in an attention network: query, key, and value. For self-attention, all of them are equal to $E^{(u,t)}$. At first, nonlinear transformations are applied on query and key:

$$Q' = \sigma(E^{(u,t)} W_Q) \tag{29}$$

$$K' = \sigma(E^{(u,t)} W_K), \tag{30}$$

where $W_Q \in \mathbb{R}^{k \times k} = W_K \in \mathbb{R}^{k \times k}$ are weight matrices. σ is activation function (usually ReLU). Then, the affinity matrix is calculated as follows:

$$z^{(u,t)} = \text{softmax}\left(\frac{Q' K'^T}{\sqrt{k}}\right), \tag{31}$$

where the output is an $L \times L$ affinity matrix (or attention map). $\sqrt{d}$ is used to scale the dot product attention. Afterward, it weights the value matrix with this attention map. Then it uses mean pooling to aggregate all the L vectors into a single vector.

$$m^{(u,t)} = \frac{1}{L} \sum_{l=1}^{L} z^{(u,t)} E^{(u,t)}. \tag{32}$$

Lastly, we replace the $Q_{S_{t-1}^u}$ in Eq. (15) and train the model in the same way. This attention module could greatly improve the expressiveness of PRME in short-term interest modeling.

2.5.4 Advanced Topics and New Frontiers

In recent years, two topics including graph neural networks [20] and deep reinforcement learning [10] are getting increasing popularity in both academia and industry. Graph neural networks work on graphs and utilize the message passing mechanism for node/graph representation learning. It is natural to apply this technique to recommendation tasks [19] as entities in recommender systems can be organized as graphs. For example, the interaction matrix can be viewed as a bipartite graph between users and items; relations between users can also form a social graph. Deep reinforcement learning is another promising technique for recommender systems. The idea behind reinforcement learning is that an agent will learn from the environment by interacting with it and receiving rewards for performing actions. There are five key concepts, including environment, agent, reward, state, and action. For recommendation task, we can consider the users and items pool as the environment, the recommendation model as agents, clicks/no clicks as rewards, features of users as states, and features of candidature items as actions [21]. However, there are still a number of challenges (e.g., scalability) in these fields that remain to be solved.

In addition, increasing the explainability of recommendations using neural networks is also useful. In many cases, both customers and developers want to know the reason why a specific item is recommended. However, most current models lack this capability. To enhance the explainability, a lot of effort has been made [35]. Readers are referred to [36] for a comprehensive survey.

2.6 Content-Based Recommender Systems

Content-based recommender systems recommend items based on the content (e.g., descriptions, article, videos, etc.) of items and a profile of the user's interests [25]. Content is the essential element in a digital world. Content can be created in many

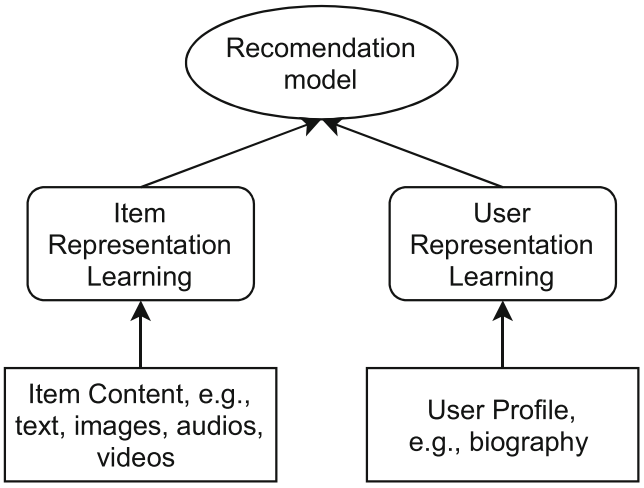

Fig. 2 The framework of content-based recommender systems

different formats. It can be structured tables/graphs or unstructured text, images, audios, videos, etc. To make effective recommendations, it is important for the recommender system to understand the content. Unlike the recommendation methods purely based on the user–item interaction matrix, content-based recommender systems are methods that combine both content and the interactions. In general, content-based recommender systems need a component to learn representations from content and a recommendation module such as MF. Naturally, many collaborative filtering approaches can be integrated as a part of content-based recommender systems. It is also called hybrid recommender systems used frequently in the literature [26, 28]. Figure 2 is a typical framework of content-based recommender systems.

The choice of the methods used for content representation learning is highly dependent on the content format. Early works used tf*idf, decision trees, and linear classifiers to model the content [25]. Nowadays, with the development of neural networks, it is a more common choice to handle the content with deep neural networks. If we have a table of categorical features, using aforementioned methods such as DeepFM and Wide&Deep learning model would be a good fit. Other neural architectures such as convolution neural networks, autoencoder, and transformer [27] can be used for more complex features. Specifically, these methods are especially effective for multimedia data sources such as text [29], image, audio, and even video. For example, convolutional neural etworks with flexible convolution and pooling operations are effective in capturing the spatial and temporal dependencies in images and texts. A number of well-defined CNN architectures such as GoogleNet and ResNet [17, 10] are ready for use. Readers are referred to the survey [11] for more detailed descriptions of deep learning solutions for these tasks.

3 Recommendation Tasks and Applications

Recommender systems are growing more popular mainly due to its usefulness in real-world applications. In this section, we will present some widely studied applications, concerns in industrial-scale recommendation, and a few of open-source toolkits that enable practitioners to get hands-on experience.

3.1 Applications of Recommender Systems

Point-of-Interest Recommendation Point-of-interest (POI) becomes popular with the emergence of location-based social network (LBSN) such as Foursquare,[1] Gowalla, and Facebook Place.[2] On these online platforms, users can check in and share their experiences about the places. The task of POI recommendation is to recommend places for users to visit. It can increase the users' viscosity to the LBSN service provider and help advertising agency to locate potential customers. POI recommendation is a representative application of sequence-aware recommendation systems as users' check-in data usually show strong sequence patterns in terms of time and geography.

Social Recommendation Social media platforms such as Facebook, Twitter, and Instagram connect users with people they are familiar with or business they are interested in. Recommender algorithms in these platforms aim to recommend people to follow, pages to like, and posts to read based on users' previous engagement and usage. A key consideration in social recommendation is the social relations such as friendship, following relationship, and membership. Social relations explicitly show the neighborhood of a user and the trust relations between users that could act as a strong regularization for recommendations. For example, we can force the users' representations to be close if they are friends.

Multimedia Recommendation Multimedia data are ubiquitous in our daily life nowadays. For example, news and blogs usually consist of text, images, and video; YouTube videos have text descriptions and subtitles; music pieces have text lyrics and album cover apart from the music audio resources. Recommending multimedia content requires the recommendation model to properly process these multimedia signals. Extracting content descriptors in these data is a well-established research task. A number of recent advanced techniques in fields such as NLP, CV, and Multimedia might be useful. For example, we can learn text representations with BERT [41], extract visual features with ResNet or Vision Transformers, and so on.

[1] https://foursquare.com/.

[2] https://www.facebook.com/places/.

Other Domains The usage of recommender systems is not limited to the aforementioned domains. Here, we will list a few more cases. For example, recommending games, news, mobile applications, cars, etc., is of great practical use for vendors. Another interesting direction is fashion-aware recommendation that involves recommending fashion-related products (e.g., clothes) by inspecting the fashion elements [37]. In addition, choosing suitable food recipe based on users' health condition [38] and recommending beverage (e.g., wine) based on customers' taste are also possible with some dedicate designs.

3.2 Practice for Industrial-Scale Recommendation

For real-world industrial-level recommender systems, the scale of dataset is usually way larger than the dataset used in academia research and beyond the ability of commonly used model. To address this issue, a two-step process that includes candidate generation and ranking is usually deployed [22]. First, it generates a set of recommendation candidates from the massive corpus with techniques such as matrix factorization. Then, it fine-tunes the candidate set with more detailed inputs with more advanced models. This is a compromise between accuracy and complexity, but it now becomes a common practice in industry. Another important aspect of industrial-level recommendation is feature engineering. Deciding which features are predictive heavily relies on expert's experiences. For some certain recommendation tasks, some specific features or combinations of features are critical for model performances. Additionally, online test is a very important step in evaluating the actual effectiveness of a recommendation model in industry. A/B testing (bucket testing) is one of the most popular online test approaches where two models are deployed to randomly serving visitors for comparison to determine which one performs better.

3.3 Tool Kits for Building Recommender Systems

To become familiar with the concepts and techniques in recommender systems, it is a good idea to get some hands-on experiences. However, implementing a recommender system from scratch is usually troublesome and time-consuming. As such, we collected some open-source recommendation libraries that aim to help us demonstrate or build a simple recommender model easily.

- **MyMediaLite**.[3] It is an open-source recommendation library published in 2011. It supports three programming languages: C#, Clojure, and F#. It provides algorithms on both rating prediction and item ranking tasks.

[3] http://mymedialite.net/index.html.

- **DeepRec**.[4] It is an open-source library for recommendation with deep neural networks. It is a Python library that uses TensorFlow as its backend and addresses tasks such as rating prediction, item ranking, and sequence-aware recommendation.
- **LibRec**.[5] It is a Java library for recommendation. It aims to solve the rating prediction and item ranking tasks. A number of traditional recommendation algorithms are provided.
- **Suprise**.[6] It is a Python toolkit that provides a limited amount of rating prediction models.
- **OpenRec**.[7] OpenRec is also a Python recommendation library. In this library, each recommender is a structured ensemble of reusable modules. However, there are only a few algorithms implemented.

4 Evaluate Recommender Systems

Proper evaluation measures are critical to building a satisfying recommender system. Evaluation is an important step in deciding which recommendation algorithm is the best. For different recommendation tasks, we may need different evaluation measures. Here, we introduce some commonly used ones.

4.1 Evaluation on Recommendation Accuracy

In general, accuracy is the top priority and the major concern of most recommender systems. Here, we summarize several commonly used accuracy measures. We omit the measures used in classification tasks as they are commonplace in other areas.

Root Mean Square Error (RMSE) RMSE is a widely used evaluation measure for measuring the accuracy of ratings prediction. The definition is as follows:

$$\text{RMSE} = \sqrt{\frac{1}{|T|} \sum_{(u,i)\in T} (\hat{X_{ui}} - X_{ui})^2}, \tag{33}$$

where T is the dataset we want to evaluate on, $\hat{X_{ui}}$ denotes the predicted ratings, and X_{ui} is the ground truth. RMSE will give relatively high weight to large errors

[4] https://github.com/cheungdaven/DeepRec.
[5] https://www.librec.net/.
[6] http://surpriselib.com/.
[7] https://openrec.ai/.

as the errors are squared before averaged. It is most useful when large errors are particularly undesirable.

Mean Average Error (MAE) It is also commonplace in measuring the accuracy of rating prediction task. It measures the average magnitude of errors and is defined as follows:

$$\text{MAE} = \frac{1}{|T|} \sum_{(u,i)\in T} |\hat{X_{ui}} - X_{ui}|. \tag{34}$$

Individual differences are weighted equally in MAE.

Recall Recall at n is the proportion of items the user liked found in the top-n recommendation list (recommended items are ranked as a list where higher ranked items are the ones that the user will like the most):

$$\text{Recall@}n = \frac{\text{Number of items that } u \text{ likes among the top-}n \text{ list}}{\text{liked}(u)}. \tag{35}$$

The final result is the average recall over all users. liked(u) is the total number of items that user u liked.

Precision Precision at n is the proportion of recommended items in the top-n list that are liked by the user:

$$\text{Precision@}n = \frac{\text{Number of items that } u \text{ likes among the top-}n \text{ list}}{n}. \tag{36}$$

The overall precision is the average precision over all users.

Mean Average Precision (MAP) It is different from precision in that correctly recommended items in top ranks are prioritized.

$$\text{MAP@}n = \frac{1}{M} \sum_{u=1}^{M} \frac{\sum_{j=1}^{n} \text{Precision@}j \times \mathbf{1}_{\text{rel}}(j)}{\min\{n, \text{liked}(u)\}}, \tag{37}$$

where $\mathbf{1}_{\text{rel}}(j)$ is an indicator function equaling 1 if the item at rank k is liked by the user. Obviously, MAP is a rank-aware evaluation metric as it rewards the system for having the "correct" items higher ranked in the list.

Normalized Discounted Cumulative Gain (NDCG) It is also a rank-aware measure, where positions are discounted logarithmically. The definition of NDCG is as

$$\text{DCG@}n = \sum_{j=1}^{n} \frac{\mathbf{1}_{\text{rel}}(j)}{log_2 j + 1}. \tag{38}$$

And NDCG@n $= \frac{\text{DCG@}n}{\text{IDCG@}n}$ with IDCG@n denoting the DCG for perfect ranked list.

Mean Reciprocal Rank (MRR) It cares about the single highest ranked relevant item, and it calculates the reciprocal of the rank at which the first item is put.

$$\text{MRR} = \frac{1}{M}\sum_{u=1}^{M}\frac{1}{\text{rank}_u}, \tag{39}$$

where rank_u is the rank of the first correctly ranked item for user u.

4.2 Beyond Accuracy

Beyond accuracy, there are many other aspects such as coverage, privacy, diversity, novelty, robustness, scalability, explainability, and freshness that are important for recommender systems [24, 23]. For example, coverage describes the proportion of items that the recommender system can recommend or the proportion of users for which the recommender system can recommend items; privacy means the systems should not disclose user's preferences to third parties without permission; diversity might be very important in some recommendation scenarios where similar items may not be as useful for different users (e.g., recommendation for a household with people having different tastes). A robust recommender system should keep stable in the presence of attacks or misinformation. Explainable recommender system could provide users with intuitive explanations of why certain items are recommended to them. Scalability is also a key factor when making decisions as different recommendation scenarios may have different levels of tolerance on the computational overhead.

5 Conclusion

This chapter was structured around the techniques, applications, and evaluations of recommender systems. We introduced non-personalized methods, neighborhood method, factorization-based approaches, as well as recent deep-learning-based methods. These methods are applicable for a wide spectrum of recommendation tasks. We also discussed several specific recommendation applications and concerns of designing industrial-scale recommender systems. Moreover, we introduced a set of evaluation metrics that can be used for evaluating recommender systems.

The values and contributions of recommender systems cannot be overestimated. The development and advancement in the field are inspiring and enlightening. We hope that the panorama of recommender systems provided in this chapter can help

researchers and practitioners to get a deep understanding toward recommender systems.

References

1. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. WWW.
2. Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 8 (2009): 30–37.
3. Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. KDD. ACM, New York, NY, USA, 426–434.
4. Ma, Hao, et al. "Recommender systems with social regularization." WSDM. ACM, 2011.
5. Mnih, Andriy, and Ruslan R. Salakhutdinov. "Probabilistic matrix factorization." Advances in Neural Information Processing Systems. 2008.
6. Rendle, Steffen, et al. "BPR: Bayesian personalized ranking from implicit feedback." UAI. AUAI Press, 2009.
7. Hsieh, Cheng-Kang, et al. "Collaborative metric learning." WWW, 2017.
8. Quadrana, Massimo, Paolo Cremonesi, and Dietmar Jannach. "Sequence-aware recommender systems." ACM Computing Surveys (CSUR) 51.4 (2018): 66.
9. Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized ranking metric embedding for next new POI recommendation. IJCAI.
10. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT Press, 2016.
11. Zhang, Shuai, et al. "Deep learning based recommender system: A survey and new perspectives." ACM Computing Surveys (CSUR) 52.1 (2019): 5.
12. He, Xiangnan, et al. "Neural collaborative filtering." WWW, 2017.
13. Sedhain, Suvash, et al. "Autorec: Autoencoders meet collaborative filtering." WWW. ACM, 2015.
14. Wu, Yao, et al. "Collaborative denoising auto-encoders for top-n recommender systems." WSDM. ACM, 2016.
15. Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. ACM, 2016.
16. Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." arXiv preprint arXiv:1703.04247 (2017).
17. He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.
18. Zhang, Shuai, et al. "Next Item Recommendation with Self-Attentive Metric Learning." AAAI Conference. Vol. 9. 2019.
19. Ying, Rex, et al. "Graph convolutional neural networks for web-scale recommender systems." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2018.
20. Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).
21. Zheng, Guanjie, et al. "DRN: A deep reinforcement learning framework for news recommendation." WWW, 2018.
22. Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for YouTube recommendations." RecSys. ACM, 2016.
23. Ricci, Francesco, Lior Rokach, and Bracha Shapira. "Introduction to recommender systems handbook." Recommender Systems Handbook. Springer, Boston, MA, 2011. 1–35.
24. Ge, Mouzhi, Carla Delgado-Battenfeld, and Dietmar Jannach. "Beyond accuracy: evaluating recommender systems by coverage and serendipity." RecSys. ACM, 2010.

25. Pazzani, Michael J., and Daniel Billsus. “Content-based recommendation systems.” In The adaptive web, pp. 325–341. Springer, Berlin, Heidelberg, 2007.
26. Burke, R., 2002. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4), pp.331–370.
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention is all you need. arXiv preprint arXiv:1706.03762.
28. Barkan, O., Koenigstein, N., Yogev, E. and Katz, O., 2019, September. CB2CF: a neural multiview content-to-collaborative filtering model for completely cold item recommendations. RecSys (pp. 228–236).
29. Malkiel, I., Barkan, O., Caciularu, A., Razin, N., Katz, O. and Koenigstein, N., 2020. RecoBERT: A Catalog Language Model for Text-Based Recommendations. arXiv preprint arXiv:2009.13292.
30. Linden, Greg, Brent Smith, and Jeremy York. “Amazon.com recommendations: Item-to-item collaborative filtering.” IEEE Internet Computing 7, no. 1 (2003): 76–80.
31. Bennett, J. and Lanning, S., 2007, August. The Netflix Prize. In Proceedings of KDD Cup and Workshop (Vol. 2007, p. 35).
32. Koenigstein, N., Nice, N., Paquet, U. and Schleyen, N., 2012, September. The Xbox recommender system. In RecSys (pp. 281–284).
33. Barkan, O. and Koenigstein, N., 2016, September. Item2vec: neural item embedding for collaborative filtering. In 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP) (pp. 1–6). IEEE.
34. Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
35. Barkan, O., Fuchs, Y., Caciularu, A. and Koenigstein, N., 2020, September. Explainable recommendations via attentive multi-persona collaborative filtering. In RecSys (pp. 468–473).
36. Zhang, Yongfeng, and Xu Chen. “Explainable Recommendation: A Survey and New Perspectives.” Foundations and TrendsÁő in Information Retrieval 14, no. 1 (2020): 1–101.
37. Kang, W.C., Fang, C., Wang, Z. and McAuley, J., 2017, November. Visually-aware fashion recommendation and design with generative image models. ICDM (pp. 207–216). IEEE.
38. Phanich, M., Pholkul, P. and Phimoltares, S., 2010, April. Food recommendation system using clustering analysis for diabetic patients. In 2010 International Conference on Information Science and Applications (pp. 1–8). IEEE.
39. Goldberg, D., Nichols, D., Oki, B.M. and Terry, D., 1992. Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), pp. 61–70.
40. Vinh Tran, L., Tay, Y., Zhang, S., Cong, G. and Li, X., 2020, January. HyperML: a boosting metric learning approach in hyperbolic space for recommender systems. In Proceedings of the 13th International Conference on Web Search and Data Mining (pp. 609–617).
41. Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.